МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Мобільний ігровий додаток "Heroes of Eternal"»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ.м-91 Онищенко Сергій Вікторович

Кваліфікаційну роботу захищено на засіданні ЕК з оцінкою _____ «___» грудня 2020 р.

Науковий керівник _____ к.т.н., доц., Федотова Н.А.
(підпис)

Голова комісії _____ Шифрін Д.М.
(підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент _____
(підпис)

Суми-2020

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. секцією ІТП

_____ В. В. Шендрик
«___» _____ 2020 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Онищенко Сергій Вікторович
(прізвище, ім'я, по батькові)

1 Тема проекту *Мобільний ігровий додаток "Heroes of Eternal"*

затверджена наказом по університету від «___»_____2020 р. № _____

2 Термін здачі студентом закінченого проекту «_10__» ___грудня___ 2020 р.

3 Вхідні дані до проекту Правила гри "Heroes of Eternal"

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) Аналіз предметної області, постановка задачі, проектування, реалізація

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Мета та задачі, аналіз предметної області, порівняння аналогів, вибір програмного щабезпечення, контекстна діаграма роботи додатку, діаграма декомпозиції, діаграма варіантів використання, реалізація, головний ігровий цикл, мета гра, процедурна генерація рівнів, моделі, анімації та звук, інтерфейс, візуальні ефекти, висновки.

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Аналіз предметної області			
Постановка задачі			
Проектування			

Дата видачі завдання 01.09.20.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
	Ідентифікація ідеї додатку	До 07.09.20	
	Аналіз існуючих аналогів	До 09.09.20	
	Вибір засобів дослідження	До 15.09.20	
	Постановка мети та задач	До 22.09.20	
	Проектування	До 30.09.20	
	Реалізація базових механік	До 28.10.20	
	Реаліація процедурної генерації	До 06.11.20	
	Реалізація мета гри	До 20.11.20	
	Створення моделей та анімацій	До 23.11.20	
	Реалізація інтерфейсу	До 27.11.20	
	Представлення роботи	До 20.12.20	

Магістрант _____ Онищенко С.В.

Керівник роботи _____ к.т.н., доц. Федотова Н.А.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра «Мобільний ігровий додаток "Heroes of Eternal"».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 28 найменувань, додатків. Загальний обсяг роботи − 88 сторінки, у тому числі 71 сторінка основного тексту, 2 сторінки списку використаних джерел, 12 сторінок додатків.

Кваліфікаційну роботу магістра присвячено розробці мобільного ігрового додатку на Android, в жанрі action rougelike.

В роботі проведено аналіз предметної області з дослідженням існуючих аналогів, визначення мети та задач проекту, обрані методи та засобіи реалізації, виконане планування та проектування роботи.

У роботі виконано поетапну розробку ігрового додатка, реалізовані базові механіки, процедурна генерація рівнів, мета гра, створені 3D-моделі та анімації, візуальні ефекти, інтерфейс, локалізація, звук, та ігрові предмети.

Результатом проведеної роботи є готовий ігровий додаток "Heroes of Eternal". який орієнтований на масове використання серед користувачів Android пристроїв.

Практичне значення роботи полягає у тому, що ігровий додаток може бути корисним у розвитку моторики, уваги, концентрації стратегічного та креативного мислення у гравця. Крім того, як і будь яка гра, додаток несе у собі розважальну функцію.

Ключові слова: ігровий додаток, гра, action, rougelike, Unity, Android, процедурна генерація.

ЗМІСТ

ВСТУП

Актуальність дипломної роботи зумовлена величезною популярністю мобільних відеоігор на сьогоднішній день. Мобільні технології, вже дуже давно, стали невід'ємною частиною нашого життя. А разом зі зростанням ринку мобільних гаджетів, так само швидко почав зростати ринок цифрових розваг. Тому сьогодні, відеоігрова індустрія є найбільшою частиною світового ринку цифрового розважального контенту, та кожен рік залучає величезну аудиторію, й отримує чисельні доходи.

Об’єкт дослідження проекту – мобільні ігрові додатки.

Предмет дослідження проекту – мобільний ігровий додаток в жанрі екшн roguelike.

Мета дипломної роботи – розробити мобільний ігровий додаток в жанрі екшн roguelike.

Для досягнення поставленої мети потрібно вирішити наступні задачі:

- Аналіз предметної області та вибір способу реалізації;
- Розробка структури додатка;
- Розробка ігрового додатку;
- Провести тестування проекту.

Методи дослідження:

- компонентно-орієнтоване програмування;
- процедурна генерація контенту.

Практичне значення. Як і будь яка відеогра, «Heroes of Eternal» виконує насамперед розважальну функцію, але може також бути корисною в розвитку моторики, уваги, концентрації стратегічного та креативного мислення у гравця.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Останнє десятиліття світовий ринок відеоігор переживає інтенсивне зростання. Цього року його дохід в усьому світі складає 164,6 млрд доларів, що на 9,3% більше в порівнянні з 2019 роком [1]. А за прогнозами експертів, до 2025 року він досягне понад 300 млрд доларів [2].

Таке зростання пов’язане з технологічною еволюцією. Сьогодні люди грають не тільки на традиційних ПК та консолях (Microsoft Xbox, Sony PlayStation та Nintendo Switch), а й на мобільних платформах таких як смартфони та планшети.

На сьогоднішній день статистика мобільних ігрових додатків показує, що більше ніж 50% всього доходу від відеоігор у світі створюється за рахунок ігор саме для мобільних пристроїв, і ця цифра продовжує зростати [3]. Це стало можливим завдяки тому, що сьогодні понад дві п’ятих світового населення володіє смартфоном або планшетом, а також має доступ до швидкого мобільного інтернету. Також цьому посприяли технічні досягнення, завдяки постійній еволюції мобільних технологій геймери отримують покращений ігровий досвід.

У 2019 році відеоігрові додатки були найпопулярнішою категорією в магазині Google play, вони займали 13,35% всіх доступних додатків. За оцінками у 2018 році у світі налічувалося 2,2 мільярда мобільних геймерів, з яких 203 мільйони з США, а 459 мільйонів з Китаю [4].

На західному ринку найпопулярнішими жанрами мобільних ігор є: гіпер-казуальні ігри, екшн ігри, аркадні ігри та головоломки. На відміну від західних гравців, азіатські користувачі люблять грати в мультиплеєрні ігри: партійні екшн баталії та MMORPG ігри, закріплюючи за ними місця в топ 5 найпопулярніших мобільних ігрових жанрів [5].

1.1 Аналіз та загальна характеристика предметної області

Гра – структурована діяльність людини, яка полягає в моделюванні іншого виду діяльності з метою розваги. Від інших видів діяльності (таких як робота) вона відрізняється тим, що не ставить перед собою мету яка б була «корисною» для гравця. Хоча слід зазначити, що ця границя не є чіткою, адже існує багато ігор які можна розглядати як роботу [6].

В ігри можуть грати як просто для задоволення, так і для досягнення певної винагороди. В одні можна грати поодинці, в інші в команді та проти інших гравців. Можна виділити наступні ключові елементи гри: цілі, правила, виклик та взаємодія. Зазвичай ігри включають в себе фізичну або розумову стимуляцію гравця, а часто і те і інше.

Відеогра – гра, яка являє собою комп'ютерну програму та використовує мультимедійні можливості комп'ютера. Пристрої, які використовують для відеоігор – ігрові платформи, до них входять: ПК, консолі, смартфони та ін. Пристрої, які використовують для керування відеоіграми – ігрові контролери, до них входять: гейм-пади, мишки, клавіатури, сенсорні екрани, та ін.

Ігрова механіка – правило або обмеження, яким керується гравець, а також реакція самої гри на дії гравця. Сукупність всіх ігрових механік однієї гри формують певну організацію її ігрового процесу. Також вони можуть формувати певний жанр гри.

Ігровий цикл (або core-gameplay, core-loop) — зациклений набір головних ігрових механік, які в сумі формують ігровий досвід для гравця. Це ті дії які гравець буде виконувати під час всього ігрового процесу знову та знову. Core-loop – це головний ігровий цикл гри навколо якого побудована вся гра. У процесі він може змінюватись та ускладнюватись, коли до нього додаються нові ігрові механіки, або еволюціонують старі.

Ігровий цикл повинен [7]:

- бути зрозумілим;
- бути легко виконуваним;
- бути гнучким та різноманітним;
- мати зв’язок з іншими ігровими циклами гри;
- приносити задоволення гравцю.

Мета гра (або meta-gameplay) – набір ігрових механік та систем, що не входять до головного ігрового циклу, а існують навколо нього та створюють основні або додаткові цілі для гравця, що направленні на залучення його в ігровий процес. Це циклічні системи, що можуть впливати на core-loop, але не беруть участі в ньому [8].

Наприклад система щоденних завдань у багатьох мобільних іграх. Кожен день гравець отримує завдання в грі, за виконання якого отримує внутрішньо ігрові бонуси, які допоможуть йому виконати ще більше завдань. Ця система стимулює гравця заходити в гру кожен день.

Слід зазначити, що мета гра не є обов’язковою частиною відеоігор, адже існує багато відеоігор, які побудовані виключно навколо головного ігрового циклу.

Rougelike – піджанр відеоігор, який походить від гри «Rouge» (звідси назва Rouge-like), що була випущена розробниками Гленном Віхманом та Майклом Тойем в 1980 році. Гра завоювала велику популярність та культовий статус [9]. Можна виділити дві ключові характеристики даного жанру: процедурна генерація рівнів та «перманентна смерть» гравця. Останнє означає, що після програшу гравець втрачає весь свій накоплений прогрес та починає спочатку, а рівні генеруються заново. На мій погляд, основною перевагою даного жанру є висока реіграбельність таких ігор.

Ігровий рівень – це певний віртуальний простір, який доступний гравцю в ході виконання ігрових цілей. Після того, як гравець виконав цілі рівня — він переходить на наступний. Як правило, кожен наступний рівень гри складніший за попередній. Наповнення рівнів залежить від жанру гри, та його головних ігрових механік. Щоб підтримувати інтерес та залучення гравця, кожен наступний рівень мусить надавати новий контент та нові перешкоди. Гра вважається пройденою, коли гравець пройшов всі її рівні.

Процедурна генерація – створення відеоігрового контенту автоматично за допомогою алгоритмів, замість створення вручну. Таким «контентом» можуть бути текстури, предмети, квести, музика, але частіше всього даним методом створюють ігрові рівні. Алгоритм створення контенту залежить від його типу та бажаного результату. В процедурній генерації використовуються випадкові або псевдовипадкові значення, які призводять до непередбачуваного діапазону можливого ігрового контенту. Концепція «випадковості» є головною, процедурна генерація повинна гарантувати, що з декількох параметрів може бути згенеровано велику кількість можливого контенту. Згенерований контент повинен відповідати наступним характеристикам [10]:

- Надійність: контент повинен відповідати заданим критеріям (наприклад якщо це ігровий рівень, то він повинен мати можливість бути пройденим гравцем);
- Швидкість: залежно від цілей та алгоритму контент може створюватись від декількох мілісекунд до декількох місяців, але він повинен бути створеним вчасно;
- Контролепридатність: можливість контролювати згенерований контент виходячи з ситуації;
- Різноманітність: чим різноманітнішим буде контент, тим більш реіграбельною буде відеогра.

1.2 Аналіз існуючих аналогів

Archero – це аркадна екшен відеогра в жанрі roguelike з видом камери зверху. Розроблена компанією Habby та випущена 24 березня 2019 року на iOS та Android. Гра є безкоштовною, та має більше ніж 10 000 000 кількостей установок у Google Play та оцінку 4,4/5 [11].

Core-gameplay в Archero є аналогічним до інших проектів в жанрі Rougelike. Гравець керує одним лучником, що подорожує по різним рівням гри, на яких

розташовані вороги. Лучник може стріляти по ворогам тільки тоді коли не рухається. Гра має велику кількість різних типів ворогів, які після своєї смерті дають внутрішньо ігрові ресурси.

Крім цього в Archero також є meta-gameplay. За внутрішньо ігрові ресурси такі як «досвід» та «гроші» герой може отримати бонуси, та витратити їх між рівнями в магазині для купівлі предметів, що підсилять його характеристики. Крім того, у грі доступно декілька лучників, на вибір. Деякі з них доступні за замовчуванням, інших необхідно розблокувати за допомогою внутрішньо ігрової валюти.

Головним недоліком даної гри є відсутність процедурної генерації. Замість цього гра має певний набір заздалегідь створених вручну рівнів, які випадково обираються під час ігрового процесу. Але їх замало, і через певну кількість ігрових циклів починаєш помічати як рівні повторюються. Через це Archero має дуже низький рівень реіграбельності.

Можна виділити дві сильні сторони даного проекту. Перша — просте та інтуїтивне управління героєм, яке здійснюється буквально одним пальцем. Друга — мультиплеер, деякі рівні гри можна проходити в парі з іншим гравцем.

Archero має мінімалістичну візуальну складову. Персонажі гри та вороги є низько полігональними 3D-моделями, а все оточення двовимірне. Через це гра стабільно запускається навіть на старих смартфонах. Також Archero має простий та зручний інтерфейс.

Nuclear Throne – це 2D аркадна екшен відеогра в жанрі roguelike з видом камери зверху. Розроблена компанією Vlambeer та випущена 5 грудня 2015 року для Microsoft Windows, Linux, OS X, PlayStation 4 та Vita, а також 25 березня 2019 року для Nintendo Switch [12].

Core-gameplay в Nuclear Throne є аналогічним до інших проектів в жанрі Rougelike. Гравець керує одним з дванадцяти ігрових персонажів, та переміщується по 15 процедурно згенерованих рівням, які розбиті на сім тематичних областей. На кожному рівні гравець може підібрати випадкову зброю для боротьби з ворогами. Між рівнями гравець може обрати «мутації» для персонажа, які зроблять його сильнішим. Після програшу прогрес рівнів та мутації скидаються.

Слабкою стороною гри є її meta-gameplay, який є дуже примітивним. Він полягає у відкритті всіх дванадцяти ігрових персонажів, виконуючи завдання для кожного з них. Оскільки персонажів не так багато, цього не вистачить для залучення гравця у гру надовго.

Найсильніша сторона Nuclear Throne це процедурна генерація рівнів. Кожен наступний рівень не схожий на попередній, та має свою структуру. В кожній тематичній області спавняться різні типи ворогів, та змінюється загальна побудова: від широких арен, до вузьких коридорів. Через це після чисельних циклів ігровий процес не набридає.

Enter the Gungeon – 2D аркадна екшен відеогра в жанрі roguelike з видом камери зверху. Розроблена компанією Dodge Roll та випущена 5 квітня 2016 року для Microsoft Windows, OS X, Linux і PlayStation 4 Xbox One та Nintendo Switch [13].

Оскільки Enter the Gungeon це roguelike, його core-gameplay схожий на попередні аналоги. Гравець обирає одного з чотирьох персонажів, де кожен має свої здібності (Другий гравець може приєднатися до гри, у якості мультиплеерного режиму). Головна задача — пройти відповідну кількість рівнів (поверхів), з випадковими кімнатами та ворогами. Для перемоги над останніми гравець повинен використовувати зброю, яку можна знайти або купити на рівні. Самі кімнати створені заздалегідь, але їх розташування та наповнення генеруються процедурно.

Головним плюсом гри є те, що в грі представлено понад 300 типів різної зброї та предметів, які можуть бути об'єднані для досягнення більш потужних ефектів. Такі ефекти роблять кожен новий ігровий цикл унікальним. Meta-gameplay являє собою поступове відкриття гравцем всіх видів зброї та предметів.

Hades – аркадна екшен відеогра в жанрі roguelike з видом камери зверху. Розроблена та опублікована компанією Supergiant Games 17 Вересня 2020 року для Microsoft Windows, macOS і Nintendo Switch [14].

Core-gameplay гри схожий на попередні ігри в цьому жанрі. Гравець керує героєм який переміщується по рівням та змагається з монстрами. На початку кожного ігрового циклу гравець обирає один з п’яти видів зброї. Після проходження рівня герой отримує винагороду (інколи гравцю дають вибір яку саме винагороду

отримати). Значним мінусом є те, що в грі майже відсутня процедурна генерація рівнів (аналогічно як і в Archero).

Під час ігрового процесу герой отримує підсилення. Атаки гравця отримують додаткові ефекти, наприклад ефект який уповільнює ворогів, або або ефект який надає додаткові пошкодження блискавкою. Плюсом є те, що ці ефекти можуть взаємодіяти один з одним та створювати нові ефекти. Це спонукає гравця під час кожного нового ігрового циклу пробувати нові комбінації ефектів, та робить гру більш глибокою.

Ще одним плюсом Hades є її meta-gameplay. Між ігровими забігами гравець може розблокувати нову зброю, та взаємодіяти з різними персонажами які допоможуть йому під час ігрового процесу. Останній плюс гри – її візуальна складова. Персонажі гри та вороги є низько полігональними 3D-моделями, а все оточення двовимірне.

Pokémon Quest – мобільна пригодницька гра в серії Pokémon. Розроблена компанією Game Freak, та випущена 30 травня 2018 року Android, iOS та Nintendo Switch [15].

Core-gameplay гри сильно відрізняється від попередніх аналогів, адже жанр гри не rougelike. Головне завдання гри – пройти всі рівні на острові та перемогти всіх покемонів. На кожен рівень гравець може взяти з собою до трьох покемонів. Вони можуть атакувати та виконувати спеціальні прийоми. Завершення одного рівня відкриває доступ до наступного в локації. Meta-gameplay гри полягає в пошуку та тренуванні нових покемонів.

Найсильнішою стороною гри є її візуальна складова та оптимізація. Всі 3D-ассети гри мають блоковий, воксельний дизайн (схожий на популярну гру Minecraft). Через це вона виглядає стильно з дизайнерської точки зору, та має високу продуктивність на мобільних платформах.

Таблиця 1.1 – Порівняння аналогів

	Archero	Nuclear Throne	Enter the Gungeon	Hades	Pokémon Quest	Heroes of Eternal
3D-графіка	+/–	–	–	+/–	+	+
Процедурна генерація рівнів	–	+	+	–	–	+
Мультиплеер	+	+	+	–	–	–
Мобільна версія гри	+	–	–	–	+	+
Версія гри для ПК/консолей	–	+	+	+	+	–

1.3 Вибір програмного забезпечення

Ігровий рушій − середовище розробки програмного забезпечення, яке призначене для створення відеоігор. Сьогодні, ігрові рушії роблять складну задачу розробки відеоігор – простою, надаючи шар абстракції. Також вони дають змогу експортувати гру на різні ігрові платформи, з мінімальними змінами.

− Вхідні дані: ігровий рушій забезпечує підтримку різних ігрових контролерів. Існує багато різних способів обробки вхідних даних, найчастіше використовуються два з них: «events» та «polling». Перший фіксується комп’ютером (наприклад, клік правою кнопкою миші, натискання клавіші, тощо), і код запускається на основі отриманого вводу. Другий використовується для отримання значень позиції, (наприклад, координати на яких знаходиться курсор миші, кут нахилу смартфона, на якому запущена гра);

− Графіка: 3D-графіка створюється з використанням 3D-ресурсів, які роблять у зовнішніх програмах для 3D-рендерингу (таких як Maya, Blender тощо), а

потім імпортуються в сам ігровий рушій. Отже, ігровий рушій повинен підтримувати формати імпорту 3D-ресурсів. Він надає такі функції як: ефекти світла, тіні, анімації, карти нормалів і т.д;

− Аудіо: частина підсистеми ігрового рушія, яка використовується для управління звуковими ефектами;

− Фізика: в більшості ігрових рушіїв існує такий компонент як фізичний рушій. Він дозволяє виконувати доволі точне моделювання реальних фізичних систем (наприклад, гравітація, виявлення зіткнень, швидкість та прискорення тіла, зміна маси тіла, пружність тіла гідродинаміка, і т.д.);

− Штучний інтелект: сьогодні штучний інтелект відіграє значну роль у розробці ігор. Реалізація ШІ в іграх зазвичай здійснюється за допомогою готових сценаріїв [16].

Великі ігрові компанії можуть дозволити собі створити та використовувати власний ігровий рушій, але зараз існує достатньо вже існуючих програмних рішень.

Godot − це 2D та 3D, крос-платформний, безкоштовний і відкритий ігровий рушій, який дозволяє розробляти відеоігри, орієнтовані на ПК, мобільні та веб-платформи [17].

Архітектура рушія побудована навколо концепції "вузлів". Вузли організовані всередині "сцен", які є групами вузлів. Усі ігрові ресурси, включаючи скрипти та графічні ресурси, зберігаються як частина файлової системи комп'ютера, а не в базі даних. Таке рішення для зберігання даних полегшує співпрацю між групами розробників відеоігор, які використовують системи контролю версій програмного забезпечення.

Godot підтримує такі мови програмування як: C++, C# та будь-які інші мову з прив'язками GDNative, такі як Rust, Nim та D. Рушій має також свою скриптову мову GDScript, яка дуже схожа на Python, а також підтримує візуальне кодування за допомогою власної мови візуального програмування VisualScript.

Графічний рушій Godot використовує OpenGL ES 3.0 або OpenGL ES 2.0. Він підтримує: динамічні тіні, карти нормалей, дзеркальність, як запечене так і динамічне глобальне освітлення, а також різні ефекти постобробки. Godot також включає в себе

окремий 2D-графічний рушій, який працює незалежно від 3D-рушія. Також він містить систему анімацій з інтерфейсом для скелетної анімації, анімаційних дерев та анімацій змішування. Практично будь-яка змінна, що створена в ігровому об'єкті, може бути анімована.

Можна також виділити декілька недоліків Godot:

- недостатня кількість документації;
- проблеми з імпортом 3D-ресурсів;
- слабкий фізичний рушій.

Ігровий рушій Godot є повністю безкоштовним, та має ліцензію MIT.

Unity – це крос-платформенний ігровий рушій, що розроблений компанією Unity Technologies. Вперше його було представлено в 2005 році на конференції Apple, тоді він призначався тільки для розробки для OS X. На Сьогоднішній день Unity є одним з найпопулярніших ігрових рушіїв. Наприклад більше половини всіх нових мобільних ігор розробляються на Unity, а також більше 70% ігор віртуальної да доповненої реальності. Unity підтримується на таких платформах як Windows, macOS і Linux, а сам рушій на сьогоднішній день дозволяє створювати ігри для більш ніж 25 різних платформ, включаючи мобільні пристрої, ПК, консолі, та віртуальну реальність [18].

Рушій дає можливість розробникам створювати відеоігри як в в 2D, так і в 3D. Unity надає основний API сценаріїв на C# як для редактора Unity у формі плагінів, так і для самих відеоігор, а також забезпечує просте «drag and drop» середовище. В нових версіях Unity, стала доступною мова візуальних сценаріїв Bolt, яка дозволяє створювати ігрові механіки та інтерактивні системи без написання єдиного рядка коду (раніше Bolt була додатковим платним розширенням).

У 2D-іграх Unity дозволяє імпортувати спрайти та має вдосконалений 2D рендеринг. Для 3D-ігор Unity дає можливість визначити стиснення текстур, mip-мапи та налаштування роздільної здатності для кожної ігрової платформи, яку підтримує рушій, забезпечує підтримку рельєфного текстурування, відображення, паралакс, screen space ambient occlusion (SSAO), динамічні тіні з використанням карт тіней, ефекти рендерингу та ефекти пост обробки зображення.

Важливим плюсом Unity є Unity Asset Store. Він дозволяє розробникам продавати або ділитися своїми ассетами. Магазин був запущений у 2010 році, до 2018 року через нього було завантажено близько 40 мільйонів ігрових ассетів. Це допомагає одним розробникам зекономити час розробки своїх проектів, а іншим – заробити [19].

Головним мінусом Unity є те, що рушій значно відстає з графічної точки зору. Він не пропонує безліч інструментів для створення складної графіки, на відміну від інших ігрових рушіїв. Також проекти, що розроблені на Unity, споживають більше пам'яті, а через це в свою чергу, виникає багато помилок OOM та проблем налагодження в додатках.

Unity має чотири версії: Personal, Plus, Pro та Enterprise. Для даного проекту оптимальною є перша версія, яка є безкоштовною, за умовою, що дохід від проекту не перевищує 100 000$ на рік. Інші версії коштують від $399 до $1,800 на рік.

Unreal Engine 4 – це крос-платформенний ігровий рушій, що розроблений компанією Epic Games. Перша версія рушія була представлена в 1998 році грою в жанрі шутер від першої особи – Unreal. Остання версія рушія (Unreal Engine 4) випущена в 2014 році та має модель розповсюдження за допомогою підписки. З 2015 року Unreal можна завантажити безкоштовно, а також він має відкритий вихідний код [20].

Unreal Engine 4 має кращі характеристики для різних типів графіки. Даний рушій відомий створенням високотехнологічних та дорогих AAA проектів. Значною перевагою є технологія рендерингу Unreal Engine, а ефекти пост обробки швидкі та підтримують багато функцій. Крім цього рушій має редактор для створення розробниками власних матеріалів, а також купу інструментів для оптимізації. Але Unreal менше підходить для розробки невеликих або мобільних проектів, хоча й підтримує мобільні ігрові платформи. Це рушій переважно для проектів з великою командою спеціалізованих розробників, а не для новачків одиночок.

Unreal Engine підтримує мову візуального програмування Blueprint, яка підходить для створення прототипів. Blueprints являє собою графи з блоків, що з’єднані разом та утворюють певну логіку. Але для основної роботи все одно краще

використовувати C++. Також рушій володіє великою кількістю інструментів та функціоналу, яких немає в аналогах. Мова C++ дає значний контроль розробникам над всією системою.

Unreal Engine 4 є безкоштовним, але якщо доходи від створеної гри перевищують 3000$ за квартал, то розробник повинен віддавати 5% доходу компанії Epic Games.

Phaser – ігровий фреймворк для створення двовимірних HTML5 – відеоігор для ПК та мобільних пристроїв. Це вільне програмне забезпечення, яке розроблене компанією Photon Storm.

Phaser внутрішньо використовує як WebGL так і Canvas рендерер та може перемикатися між ними автоматично на основі підтримки браузера. Це дозволяє дуже швидко виконувати рендер на мобільних пристроях, використовуючи бібліотеку Pixi.js. Відеогра створена на Phaser не компілюється, а просто запускається як JavaScript в браузері. Рушій підтримує такі мови програмування як JavaScript та Typescript [21].

Плюси Phaser:

- Відмінно підходить для мобільних ігор;
- Стабільність;
- Легко розширюється за допомогою веб-технологій.

Мінуси Phaser:

- Тільки 2D графіка;
- Більш обмежений в порівнянні з аналогами.

GameMaker – крос-платформенний ігровий рушій, що розроблений компанією YoYo Games в 2007 році. Остання версія рушія - GameMaker Studio 2, випущена в 2017 році. Головна його ідея це дозволити розробникам початківцям мати можливість створювати відеоігри без глибоких знань програмування [22].

GameMaker Studio 2 дає можливість створювати крос-платформні відеоігри за допомогою мови візуального програмування або скриптової мови Game Maker, яка схожа на JavaScript.

Плюси GameMaker Studio 2:

- Підтримка широкої кількості платформ;
- Простота програмування.

Мінуси GameMaker Studio 2:

- Більше орієнтований на 2D графіку (хоча й підтримує 3D);
- Платний.

2. ПОСТАНОВКА ЗАДАЧІ

2.1 Мета та задачі дослідження

Відеогра Heroes of Eternal представляє собою 3D аркадний екшн в жанрі roguelike. Рівні гри створюються за допомогою процедурної генерації контенту.

Мета: Розробити мобільний ігровий додаток Heroes of Eternal на ігровому рушії Unity в жанрі екшн roguelike з використанням випадкової генерації рівнів. Для досягнення поставленої мети потрібно вирішити наступні задачі:

1) Аналіз предметної області та вибір способу реалізації;
2) Розробка структури додатка;
3) Розробка ігрового додатку;
4) Провести тестування проекту.

2.2 Методи дослідження

Головна особливість усіх відеоігор в жанрі roguelike це процедурна генерація рівнів. Вона так чи інакше впливає на всі інші аспекти ігрового процесу — від стратегії дослідження рівня до розташування на ньому різних предметів і ворогів.

Процедурно згенеровані рівні забезпечують гравцю нескінченну реіграбельність, та кожен раз ставлять перед ним різні завдання. Крім того, задоволення від гри посилюється тим, що прогрес гравця в roguelike іграх, в першу чергу залежить від його власного ігрового досвіду, а не просто від проб і помилок.

Проаналізувавши аналоги, стало зрозуміло, що найваріативніші рівні генеруються в грі Nuclear Throne. Тому вирішено було використовувати аналогічний метод процедурної генерації. Оскільки гра має закритий вихідний код, не можна на

100% сказати який саме метод використовується у грі. На мій погляд це варіація алгоритму «Drunkard Walk».

Випадкове блукання – це математичний об'єкт, що відомий як випадковий або стохастичний процес описуючий шлях, який складається з деякої послідовності випадкових кроків на математичному просторі [23]. Найпростішим прикладом випадкового блукання є переміщення по цілочисельному числовому рядку. Що починається з 0 і на кожному кроці переміщується на +1 або на -1 з рівною ймовірністю.

Drunkard Walk – тип алгоритму випадкового блукання, який використовується для генерації підземель в відеоіграх. Сам алгоритм є доволі простим, і полягає в наступному:

1) Обираємо випадкову точку на сітці та позначаємо її заповненою;
2) Обираємо випадковий напрямок в одну з чотирьох сторін;
3) Рухаємось в цьому напрямку по сітці та позначаємо його заповненим, якщо він вже не був;
4) Повторюємо кроки 2-3, доки не заповнимо стільки точок, скільки потрібно.

Даний алгоритм генерує дійсно варіативні рівні, але головна його проблема в тому, що ці рівні є занадто випадковими. Одною з характеристик алгоритму процедурної генерації повинна бути його контролепридатність. Розробник повинен мати змогу контролювати згенерований контент. Тому було вирішено вдосконалити алгоритм.

Створимо змінну ChanceDotChangeDir яка буде відповідати за шанс того, що точка змінить напрямок руху по сітці. Після кожного кроку, з випадковим шансом (який визначає змінна ChanceDotSpawn) може створюватись ще одна точка яка буде рухатись аналогічно до першої точки. Також, після кожного кроку, з випадковим шансом (який визачає змінна ChanceDotDestoy) точка може знищитись. Додамо також змінну MaxDots яка обмежує кількість точок які можуть бути створені. Наостанок створимо змінну PercentToFill, яка визначає відсоток на який повинна бути заповнена сітка, щоб генерація закінчилась.

Тепер, куруючи змінними ChanceDotChangeDir, ChanceDotSpawn, ChanceDotDestoy, MaxDots та PercentToFill можна створювати більш передбачувані рівні. Наприклад деякі рівні можуть бути вузькими коридорами, а деякі навпаки – великими широкими аренами (рис. 2.1).

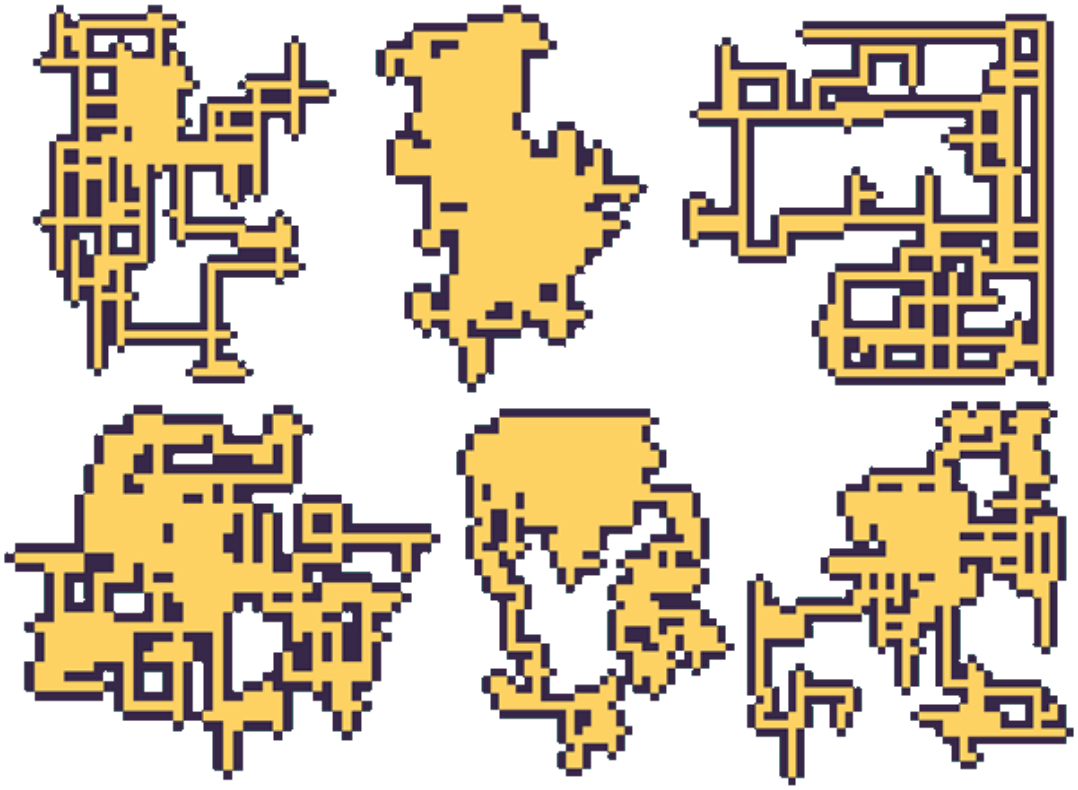

Рисунок 2.1 – Приклади карт згенерованих рівнів

Після цього відбуваєтья розміщення ігрових об’єктів: гравця, ворогів, та предметів. На початку кожного рівня гравець з'являється на краю рівня, а всі інші об’єкти розміщуються випадково, але подалі від гравця.

2.3 Вибір засобів реалізації

Мова програмування C# − об'єктно-орієнтована мова програмування загального призначення, яка була розроблена в в 1998-2001 роках компанією

Microsoft. Вона відноситься до мов з C-подібним синтаксисом, та найбільш всього схожа на Java та C++.

Сьогодні розробка додатків все більше тяжіє до створення програмних компонентів, що реалізують окремі функціональні можливості. C# надає мовні конструкції, які підтримують компонентно-орієнтоване програмування через такі концепції як: методи, події та атрибути, та дозволяє автономні та самоописові функціональні компоненти, які називаються збірками. Завдяки цьому дана мова програмування відмінно підходить для створення та застосування програмних компонентів.

C# має єдину систему типів, які успадковуються від одного кореневого типу. Тому, всі вони використовують загальний набір операцій, а значення будь-якого типу даних можна передавати, зберігати та обробляти схожим чином. Мова має сурову типізацію, що не дозволяє звертатися до неініціалізованих змінних або неконтрольовано виконувати приведення типів. Також, C# має опрацювання виняткових ситуацій, що дає структурований спосіб виявлення та обробки помилок.

Ігровий рушій Unity. Після аналізу існуючих ігрових рушіїв було прийняте рішення обрати Unity. Далі представлений більш детальний аналіз його архітектури. (рис. 2.2)

Unity це рушій, керований компонентами, тому саме за допомогою комбінацій компонентів відбувається розробка гри. Проаналізувавши схему на рисунку 2.1, ми побачимо, що існує ієрархія високого рівня, в якій одні сутності містять інші. Головними елементами даної структури є компоненти.

Найпростіший спосіб візуалізувати цю архітектуру полягає в тому, щоб вважати, що ігрова сцена це набір ігрових об’єктів (GameObjects). А вони в свою чергу є набором компонентів, які можуть реалізовувати і включати в себе:

- Системи (фізика, камери);
- Дані (текстури, анімації, конфігурації);
- Поведінки (сценарії подій, ігрові механіки).

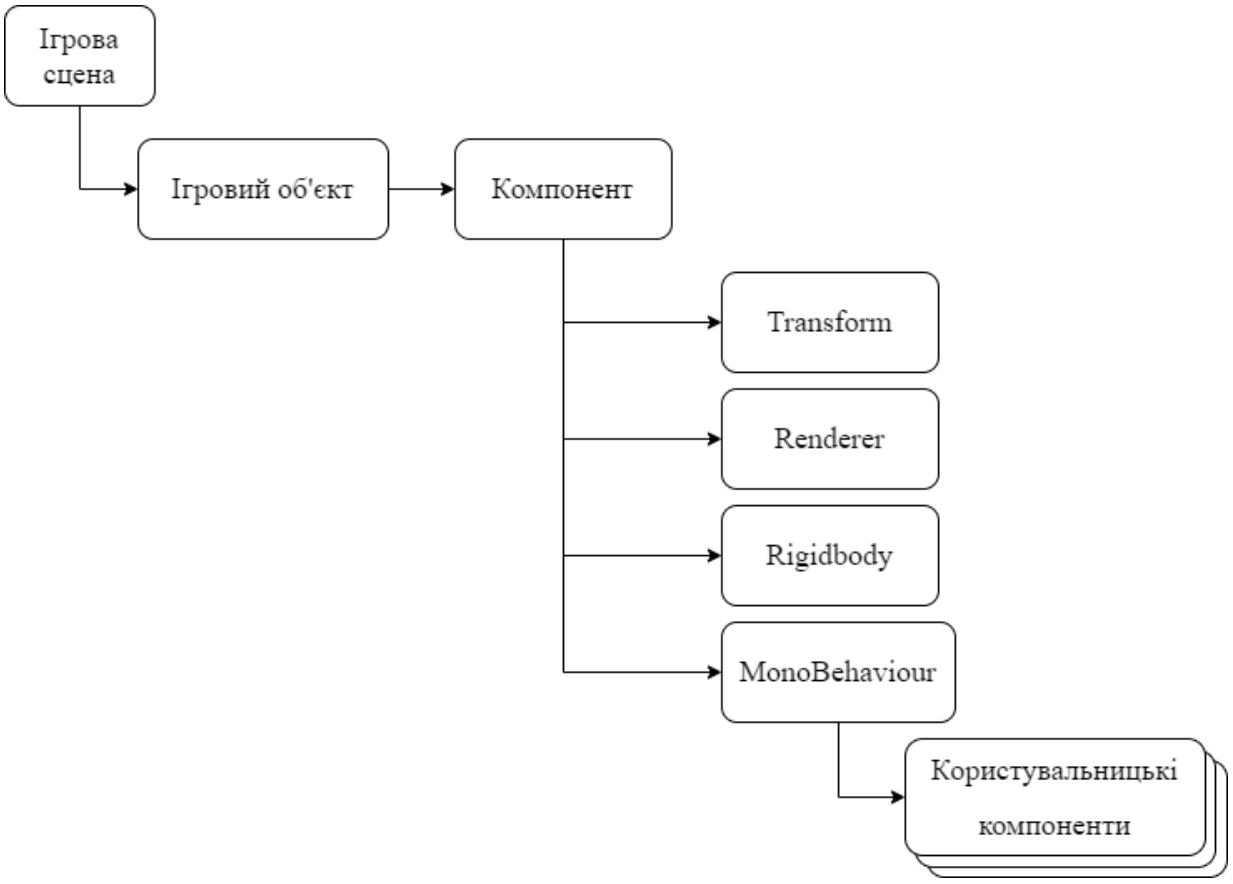

Рисунок 2.2 – Схема ігрової сцени Unity

Таким чином, при такому підході ми можемо швидко перетворити один ігровий об'єкт в інший, просто змінивши компоненти, які він містить. Це означає, що ігрові об’єкти складаються з компонентів, і в залежності від того, який тип компонента ми прикріплюємо до ігрового об’єкта, він перетворює його в певний вид сутності. Такий модульний підхід чудово підходить для розробки відеогри.

Visual Studio Code – це безкоштовний редактор вихідного коду, створений компанією Microsoft для Windows, macOS та Linux. Його функції включають в себе підтримку налагодження, інтелектуальне завершення коду, підсвічування синтаксису, рефакторинг коду та вбудований Git.

Visual Studio Tools for Unity – це безкоштовне розширення для Visual Studio Code, яке перетворює його в потужний інструмент для розробки додатків за допомогою ігрового рушія Unity. Воно надає змогу використовувати функції редагування коду, для створення сценаріїв редактора для Unity за допомогою мови C#, а також налагоджувати їх за допомогою Visual Studio Code.

Visual Studio Tools for Unity має глибоку інтеграцію з редактором Unity, так що розробник витрачає менше часу для виконання простих завдань, а також забезпечує підвищення продуктивності для Unity та надає відповідну документацію. Visual Studio Tools for Unity підтримує наступні функції:

- використання автозаповнення IntelliSense;
- швидке налагодження для Unity;
- потужні можливості рефакторингу.

Blender – це безкоштовний набір програмних засобів для створення 3D графіки. Він використовується для створення: 3D-моделей, візуальних ефектів та анімаційних фільмів. Blender має відкритий вихідний код, та підтримує наступні функції: 3D-моделювання, текстурування, UW-розгортання, редагування растрової графіки, система часток, анімацію та рендеринг.

В моєму проекті Blender використовується для створеня 3D-моделей та їх текстурування. Після чого вони експортуються в ігровий рушій Unity, який нативно імпортує всі файли Blender. Unity може імпортувати з Blender:

- Вузли, з їх положенням обертанням та маштабом;
- Меші, з вершинами, полігонами, трикутниками, та UW-розгортками;
- Анімації;
- Кістки анімацій;
- Меші з прив'язкою до кісток.

Для створення інтерфейсу гри, було використовувати растрову 2D графіку, замість векторної. Оскільки векторна графіка має суттєві недоліки в Unity:

- Ігровий рушій переводить вектор в Mesh;
- Обмежена функціональність;
- Відсутність прозорості;
- Не підтримує понад 65 тисяч вершин;
- Не можна фарбувати засобами Unity.

Adobe Photoshop – растровий графічний редактор, розроблений компанією Adobe Inc. для Windows та mac OS. Перша версія редактору була створена в 1988 році.

Сьогодні це програмне забезпечення стало стандартом в галузі не тільки в редагуванні растрової графіки, але і в цілому в цифровому мистецтві.

Файли Adobe Photoshop за замовчуванням мають розширення .PSD, у якому зберігається зображення яку має підтримку більшості параметрів візуалізації, що доступні в Photoshop. До таких параметрів відносяться: рисунки з прозорістю, масками, текстом, альфа-каналами та ін. На відміну від інших подібних форматів, які не зберігають подібні параметри.

PSD Importer – це спеціальний імпортер 2D-ассетів, який створений для імпорту PSD-файлів з Adobe Photoshop в ігровий рушій Unity. Importer підтримує 2 типи текстур - Default та Sprite.

3. ПРОЕКТУВАННЯ

3.1 Діаграми нотації IDEF0 по процесу роботи ігрового додатку

Функціональна модель IDEF0 – це структуроване представлення функцій, діяльності чи процесів у змодельованій системі чи предметній області. IDEF0 може бути використаний для визначення вимог та функцій, а потім для реалізації проекту, який їм відповідає. Двома основними компонентами моделювання є функції (представлені на діаграмі прямокутниками) та об'єкти, які взаємозв'язують ці функції (представлені стрілками) [24].

Після проведеного аналізу стосовно головних елементів контекстної діаграми «Робота ігрового додатку "Heroes of Eternal"», був сформований перелік даних:

- Вхідні дані: потреба гравця грати в гру, потреба налаштування додатку;
- Вихідні дані: завершена ігрова сесія;
- Управління: алгоритм генерації рівнів, вимоги до структури рівнів, ігрові ассети;
- Механізми: гравець, ігровий рушій та апаратне забезпечення.

При створенні даної моделі використовувався програмний продукт Erwin Process Modeler. На рисунку 3.1 зображена контекстна діаграма роботи ігрового додатку "Heroes of Eternal".

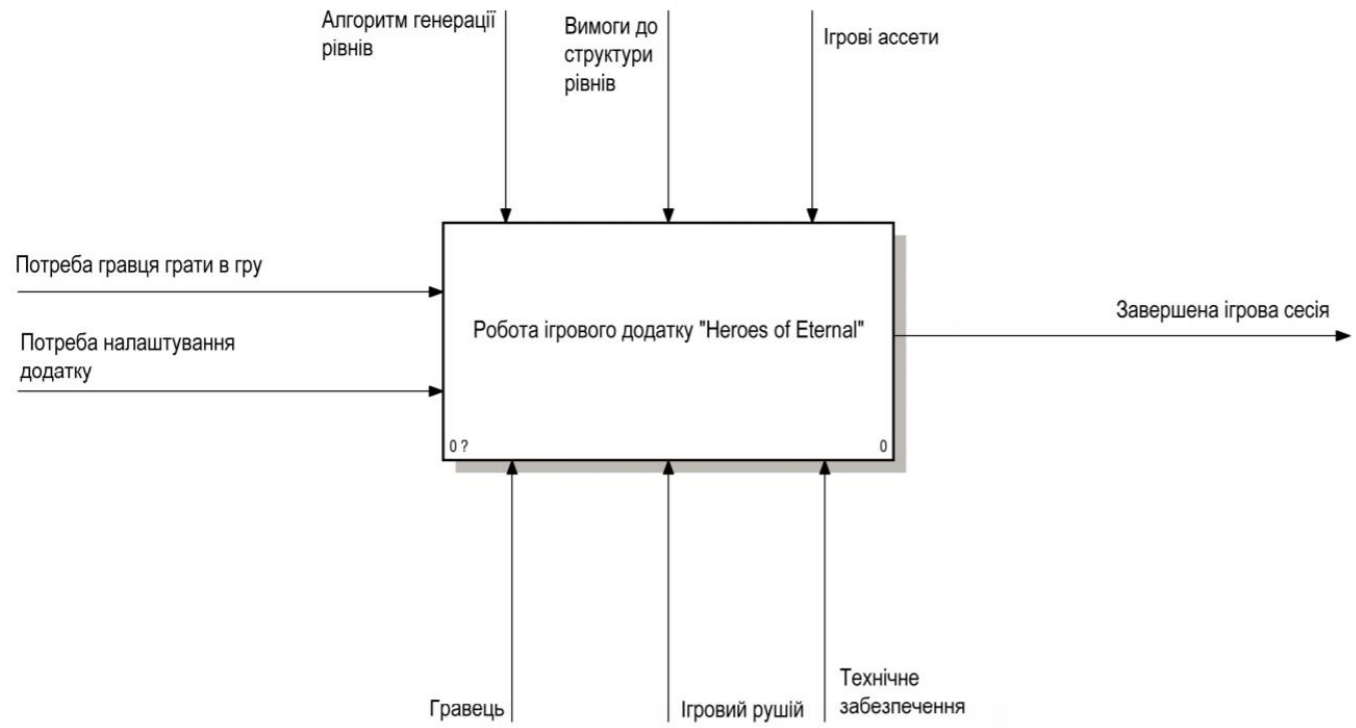

Рисунок 3.1 – Контекстна діаграма IDEF0

Оскільки контекстна діаграма містить у собі лише загальну інформацію про роботу додатку, то необхідно виконати її декомпозицію. Діаграма була поділена на три підрівні: зміна налаштувань додатку, ігровий процес та процес прокачування.

При описі зміни налаштуваннь додатку були сформовані такі дані:

- вхідні дані: потреба налаштування додатку;
- вихідні дані: дані про налаштування додатку;
- механізми: гравець, ігровий рушій та технічне забезпечення.

При описі ігрового забігу були сформовані такі дані:

- вхідні дані: дані про налаштування додатку, потреба гравця грати в гру;
- вихідні дані: завершена ігрова сесія, очки досвіду, внутрішньоігрова валюта;
- управління: алгоритм генерації рівнів, вимоги до структури рівнів, ігрові ассети;
- механізми: гравець, ігровий рушій та технічне забезпечення.

При описі процесу прокачування були сформовані такі дані:

– вхідні дані: очки досвіду, внутрішньоігрова валюта;
– вихідні дані: дані про героя;
– управління: ігрові ассети;
– механізми: гравець, ігровий рушій та технічне забезпечення.

Діаграма декомпозиції IDEF0 наведена на рис.3.2

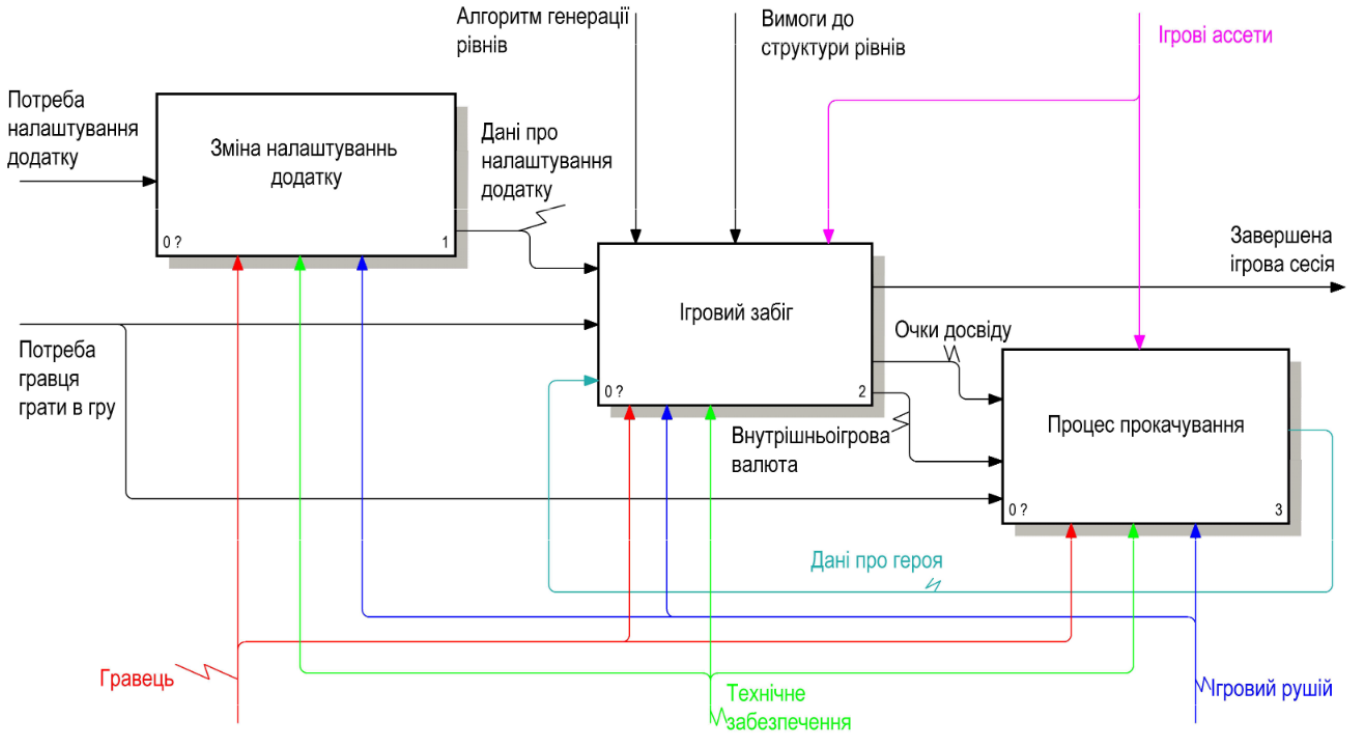

Рисунок 3.2 – Діаграма декомпозиції IDEF0

Після декомпозиції контекстної діаграми було виконано декомпозицію ігрового процесу. Він був розбитий на три підрівні: згенерувати рівень, налаштувати ігрову сцену, процес проходження рівня.

При описі генерації рівня були сформовані такі дані:

– вхідні дані: дані про пройдений прогрес гравця;
– вихідні дані: дані про рівень;
– управління: алгоритм генерації рівнів, вимоги до структури рівнів;
– механізми: ігровий рушій та технічне забезпечення.

При описі налаштування ігрової сцени були сформовані такі дані:

– вхідні дані: дані про рівень, дані про героя, дані про налаштування додатку;

– вихідні дані: ігровий рівень;

– управління: вимоги до структури рівнів, ігрові ассети;

– механізми: ігровий рушій та технічне забезпечення.

При описі процесу проходження рівня були сформовані такі дані:

– вхідні дані: ігровий рівень;

– вихідні дані: завершена ігрова сесія, очки досвіду, внутрішньоігрова валюта, дані про пройдений прогрес гравця;

– механізми: гравець, ігровий рушій та технічне забезпечення.

Вигляд процесу «Ігровий процес» після декомпозиції представлений на рис.3.3

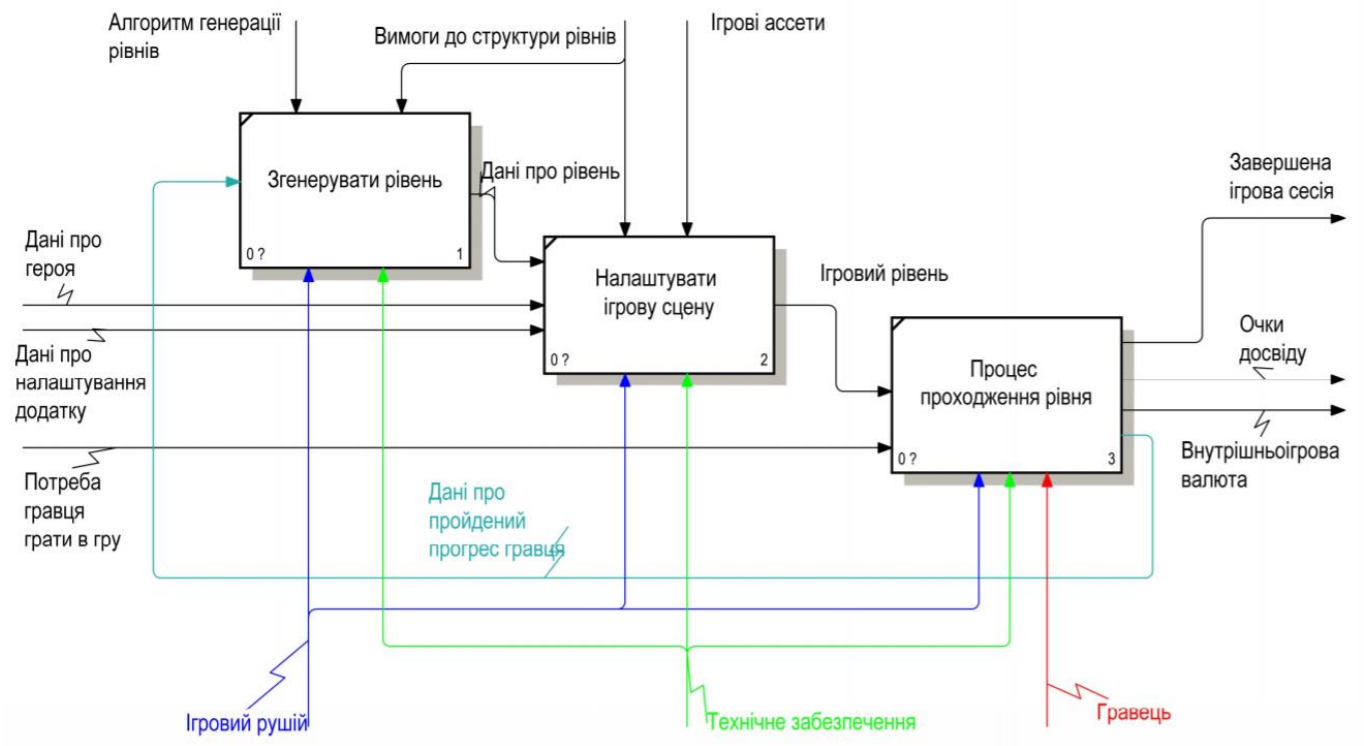

Рисунок 3.3 – Декомпозиція процесу «Процедура ігрової партії»

Після декомпозиції контекстної діаграми було виконано декомпозицію процесу прокачування. Він був розбитий на п'ять підрівнів: обрати героя, перейти на новий рівень, купити предмет, покращити предмет, надягнути предмет на героя.

При описі обрання героя були сформовані такі дані:

- вхідні дані: потреба гравця грати в гру;
- вихідні дані: дані про героя;
- управління: ігрові ассети;
- механізми: гравець, ігровий рушій та технічне забезпечення.

При описі переходу на новий рівень були сформовані такі дані:

- вхідні дані: Очки досвіду;
- вихідні дані: Внутрішньоігрова валюта;
- механізми: ігровий рушій та технічне забезпечення.

При описі купівлі предмету були сформовані такі дані:

- вхідні дані: внутрішньоігрова валюта;
- вихідні дані: ігровий предмет;
- управління: ігрові ассети;
- механізми: гравець, ігровий рушій та технічне забезпечення.

При описі покращення предмету були сформовані такі дані:

- вхідні дані: ігровий предмет, внутрішньоігрова валюта;
- вихідні дані: дані про предмет;
- управління: ігрові ассети;
- механізми: гравець, ігровий рушій та технічне забезпечення.

При описі надягання предмету на героя були сформовані такі дані:

- вхідні дані: дані про предмет, дані про героя;
- вихідні дані: дані про героя;
- управління: ігрові ассети;
- механізми: гравець, ігровий рушій та технічне забезпечення.

Вигляд процесу «Процес прокачування» після декомпозиції представлений на рис 3.4.

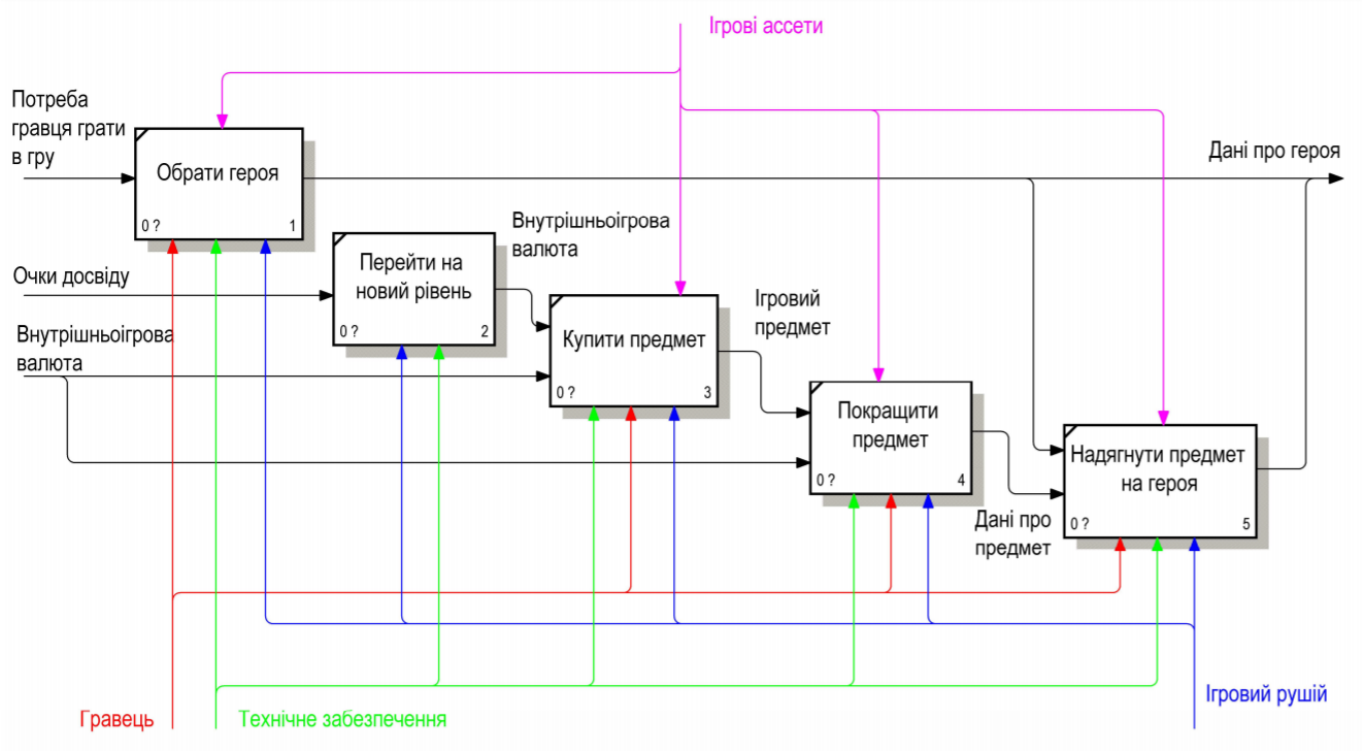

Рисунок 3.4 – Декомпозиція процесу «Процедура ігрової партії»

3.2 Діаграма варіантів використання Android-додатку

Діаграма варіантів використання підсумовує відомості про користувачів системи (також відомих як актори) та їх взаємодії з самою системою. Для її побудови, використовується набір спеціалізованих символів та стрілок. Діаграма варіантів використання може допомогти представити сценарії, в яких додаток взаємодіє з людьми, або зовнішніми системами та цілі, які він допомагає їм досягти [25].

При реалізації діаграми був визначений один актор – гравець. Далі було сформовано перелік варіантів використання:

- Розпочати гру;
- Обрати героя;
- Купити предмет;
- Змінити налаштування;
- Закрити додаток.

Варіант «Купити предмет» має продовження «Надягнути предмет» та «Покращити предмет», що у свою чергу призведе до процесу «Показ результату гри». Крім того, варіант використання «Зміна налаштувань додатку» має два варіанти використання:

- Обрати мову;
- Змінити гучність звуку.

На основі визначених варіантів використання та інформації про актора була розроблена діаграма варіантів використання, яка зображена на рис 3.5.

Рисунок 3.5 – Діаграма варіантів використання Android-додатку

4. РЕАЛІЗАЦІЯ

4.1 Реалізація головного ігрового циклу

На рисунку 4.1 представлений головний ігровий цикл гри "Heroes of Eternal".

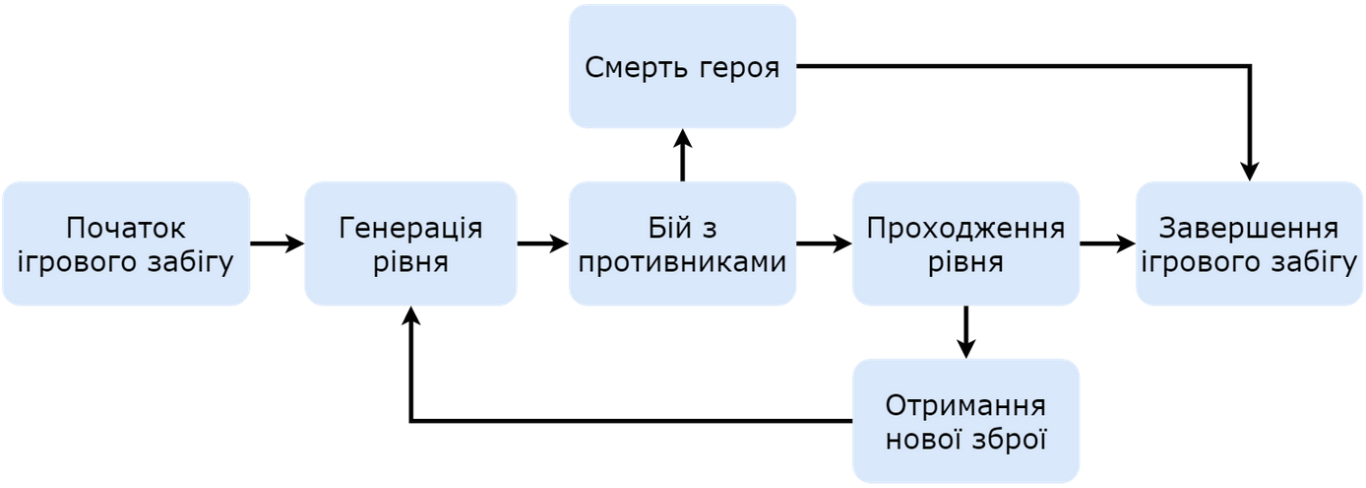

Рисунок 4.1 — Головний ігровий цикл гри

Гравець бере на себе управління героєм, який потрапляє на згенерований рівень, де знаходиться певна кількість противників. Герой пересувається коли гравець водить пальцем по екрану телефона, а коли зупиняється, то починає автоматично стріляти снарядами в найближчого противника. А противники, в свою чергу, стріляють в героя.

Кожен снаряд наносить "шкоду" та зменшує показник "здоров'я" того, в кого він потрапив. Показник "шкоди" снаряда та його швидкість, а також темп та точність стрільби залежить від зброї.

На початку ігрового циклу герой має лише одну стандартну зброю, але під час проходження рівнів може отримати нову. Герой може мати будь-яку кількість зброї, але одночасно стріляти можна лише з однієї. Змінити зброю можна в будь-який момент. Противники можуть мати лише одну зброю.

Коли показник "здоров'я" противника дорівнює нулю противник зникає з рівня. Після того, як всі противники переможені — на рівні з’являється портал, зайшовши в

який відбувається перехід на наступний рівень. Коли показник "здоров'я" героя дорівнює нулю, то герой "помирає", а ігровий забіг закінчується. Ціль гри — пройти 30 рівнів не померши. Отже, для реалізації головного ігрового циклу необхідно виконати наступні задачі:

- Реалізувати героя;
- Реалізувати ігрові характеристики;
- Реалізувати зброю.
- Реалізувати поведінку противників;
- Налаштувати ігрову камеру.

4.1.1 Реалізація героя

На початку, було в ігровій сцені було створено ігровий об’єкт зі стандартними компонентами Rigidbody та BoxColider. Перший — дозволяє об’єктам мати фундаментальні фізичні властивості, такі як сили та прискорення. Компонент Rigidbody з його налаштуваннями зображений на рисунку 4.2. Другий — реалізує колізії з іншими ігровими об’єктами.

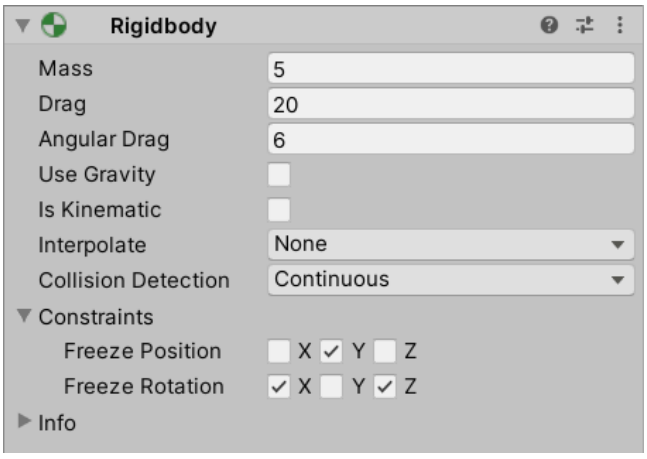

Рисунок 4.2 — Компонент Rigidbody ігрового об’єкта героя

Ігровий об’єкт героя має стани, які реалізовані в блок-схемі State Machine, за допомогою компонента Animation Controller. Дана схема представлена на рисунку 4.3. Було визначено чотири ігрові стани в яких може перебувати герой:

- **Idle** — герой стоїть на місці та не рухається. Виконується пошук найближчого противника.
- **Move** — герой переміщується по ігровому рівню у відповідному напрямку. Виконується пошук найближчого противника.
- **Attack** — герой стріляє снарядами по найближчому противнику або по противнику на якому сфокусувався.
- **Aiming** — герой стоїть на місці та обирає противника для того, щоб сфокусуватись на ньому.

Для реалізації даних станів були створені відповідні компоненти, які наслідують клас StateMachineBehaviour.

Між станами героя були визначені наступні переходи:

- **Idle** — **Move:** коли гравець починає водити пальцем по екрану смартфона;
- **Idle** — **Attack:** коли був визначений найближчий противник;
- **Idle** — **Aiming:** коли гравець натиснув на екран в одній точці, та тримає палець понад дві секунди;
- **Move** — **Idle:** коли гравець припинив водити пальцем по екрану смартфона;
- **Move** — **Attack:** коли гравець припинив водити пальцем по екрану смартфона та був визначений найближчий противник або противник на якому сфокусувався герой;
- **Attack** — **Move:** коли гравець починає водити пальцем по екрану смартфона;
- **Attack** — **Idle:** коли противник, по якому стріляв герой, помер;
- **Attack** — **Idle:** коли гравець натиснув на екран в одній точці, та тримає палець понад дві секунди;
- **Aiming** — **Idle:** коли гравець прибрав палець з екрана смартфона.

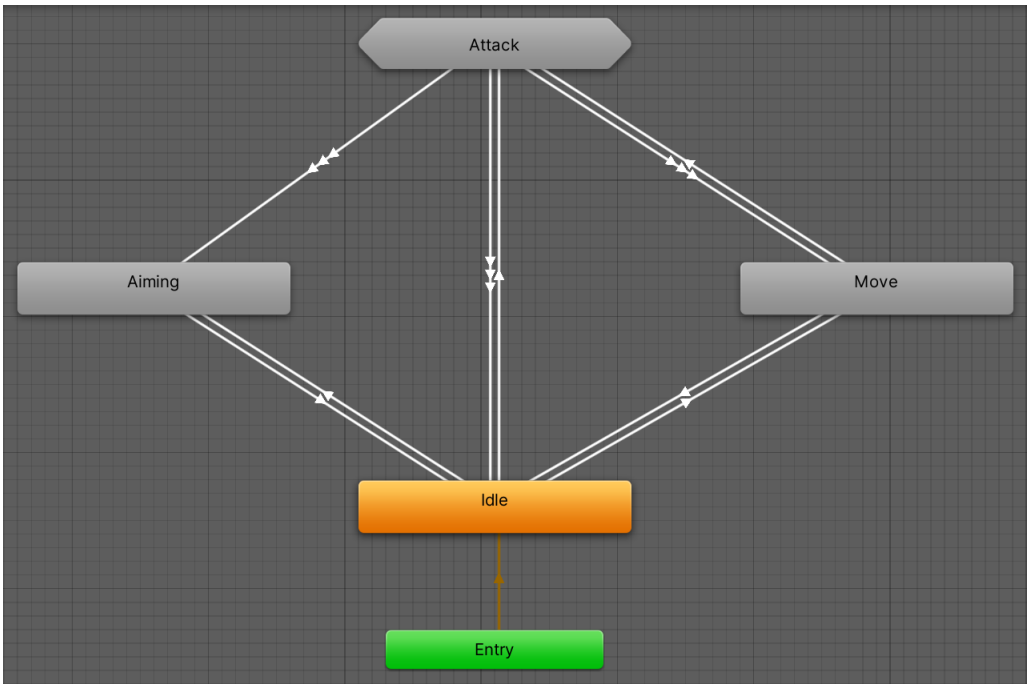

Рисунок 4.3 — State Mashine для ігрового об’єкта героя

Компонент PlayerController відповідає за зчитування вводу гравця, та відповідне переміщення героя. Даний компонент наслідує клас MonoBehaviour. У функції Update, яка викликається кожен кадр, перевіряється чи водить гравець пальцем по екрану смартфона, та в якому напрямку. Після чого, відповідно змінюється показник velocity в компоненті Rigidbody.

Компонент FieldofView визначає, яких противників герой "бачить", в цей момент часу. Спочатку він знаходить всі цілі у відповідному радіусі від себе. Після цього він вимірює кут між напрямком погляду персонажа та кожним з ворогів. Щоб визначити, які з них потрапляють в кут огляду героя. Далі, від положення героя до положення тих цілей, що залишились будуємо промені, щоб визначити чи є між героєм та противником певні перешкоди. В кінці, визначається яка з цілей ближче до героя.

Компонент PlayerAim контролює на яку саме ціль наведений герой, чи є герой сфокусованим на цій цілі. Останнє означає, що герой буде постійно атакувати одну ціль. Фокус на цілі зникає коли противник помирає, або коли гравець зробить подвійний тап по екрану.

4.1.2 Реалізація ігрових характеристик

Ігрова характеристика — це число, яке описує певний аспект ігрової сутності. В цьому випадку такими сутностями є: герой, противник, зброя та предмет. Сукупність характеристик визначає ігрові сутності в грі [26]. Під час ігрового процесу характеристики можуть збільшуватись або зменшуватись.

Далі описані основні характеристики героїв в "Heroes of Eternal":

- **MaxHp** — це сума шкоди, яку герой може отримати перед тим як помре;
- **Damage** — шкода яку герой наносить противникам;
- **Speed** — швидкість переміщення героя;
- **SpeedAttack:** швидкість темпу стрільби героя;
- **Dodge** — шанс того, що попадання ворожого снаряду по герою не зарахується;
- **CritRate** — шанс того, що герой нанесе противнику "критичну шкоду";
- **CritDamage** — розмір "критичної шкоди";
- **Mass** — визначає силу віддачі від стрільби.

Всього в "Heroes of Eternal" реалізовано три іграбельних героя, кожен з яких відрізняється як візуально, так і своїми характеристиками. Завдяки цьому, гравець може обрати того героя, який підходить його ігровому стилю. Характеристики героїв можна змінювати надягаючи на них предмети в меню, які відповідно збільшують їх, тим самим змінюючи ігровий процес.

Далі описані основні характеристики противників в "Heroes of Eternal":

- **MaxHp** — це сума шкоди, яку противник може отримати перед тим як помре;
- **Damage** — шкода яку герой наносить герою;
- **Speed** — швидкість переміщення противника;
- **CritRate** — шанс того, що герой нанесе герою "критичну шкоду";
- **CritDamage** — розмір "критичної шкоди";
- **Mass** — визначає силу віддачі від стрільби.

Далі описані основні характеристики зброї в "Heroes of Eternal":

- **Тип зброї** — на даний момент реалізовано два типи зброї: пістолет та автомат. Від типу зброї залежить анімація стрільби;
- **Додатковий ефект** — який з семи додаткових ефектів має зброя;
- **FireSpeedMultiplier** — на скільки змінюється швидкість анімації стрільби;
- **DamageMultiplier** — на скільки змінюється значення Damage гравця або противника;
- **CritRateMultiplier** — на скільки змінюється значення CritRate гравця або противника;
- **Spreading** — розкид траєкторій снарядів;
- **PushMultiplier** — визначає силу з якою снаряди штовхають ціль;
- **RecoilMultiplier** — визначає силу віддачі від стрільби.

Для реалізації ігрових характеристик було створено клас Stat. Оскільки гра передбачає можливість модифікації характеристик було також створено клас StatModifier. Stat містить в собі початкове значення характеристики та список модифікаторів. StatModifier містить значення на яке буде збільшуватись чи зменшуватись характеристика, посилання на об’єкт який застосував цей модифікатор та тип модифікатора. Всього є два типи модифікаторів:

- **Flat:** додавання числа до початкового значення;
- **Percent:** додавання відсотка від початкового значення;

В класі Stat, метод CalculateFinalValue() обчислює фінальне значення характеристики, додаючи до початкового значення всі модифікатори. А метод RemoveAllModifiersFromSource() видаляє всі методи від відповідного об’єкту.

Збереження всіх даних з характеристиками реалізоване за допомогою класу Unity — ScriptableObject. Скриптовий об'єкт це ігровий об'єкт, який може зберігати призначені для гравця дані будь-якого типу у вигляді файлу. Вони дозволяють значно скоротити обсяг пам'яті проекту, оскільки уникають дублювання копій даних. Адже, при кожному створенні нового екземпляра префаба ігрового об’єкта — створюються дублікати даних для кожного прикріпленого до нього скрипта Monobehaviour.

Також, скриптові об’єкти дозволяють відокремити ігрові дані від логіки гри, що в свою чергу дає структурувати код на незалежні модулі. Якщо дані зберігаються в об'єктах ScriptableObject, вони будуть доступними в різних ігрових сценах, без необхідності додаткової реалізації серіалізації або функції DontDestroyOnLoad в скрипті MonoBehaviour. Саме тому, було вирішено зберігати дані в об’єктах ScriptableObject та отримувати доступ до них з компонентів PlayerStats та EnemyStats.

4.1.3 Реалізація зброї

Під час ігрового процесу герой та противники стріляють один в одного снарядами. Кожен снаряд це ігровий об’єкт з компонентами Rigidbody та BoxColider. Компонент Rigidbody з його налаштуваннями зображений на рисунку 4.4.

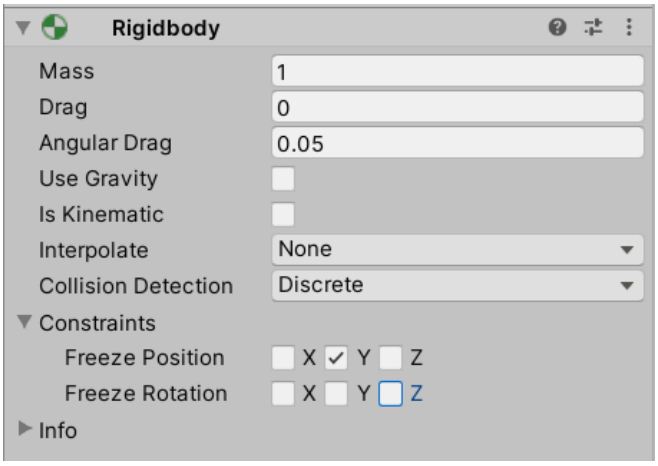

Рисунок 4.4 — Компонент Rigidbody ігрового об’єкта снаряда

Для створення моделей снарядів були використані стандартні 3D-моделі Unity: розтягнутий куб для снаряда героя, та сфера для снаряда противника. Крім того, до ігрового об’єкта снаряда противника було додано компонент TrialRenderer (Рисунок 4.5), який відображає слід полігонів коли ігровий об’єкт рухається. Це дозволяє надати ефект відчуття руху рухомому об'єкту.

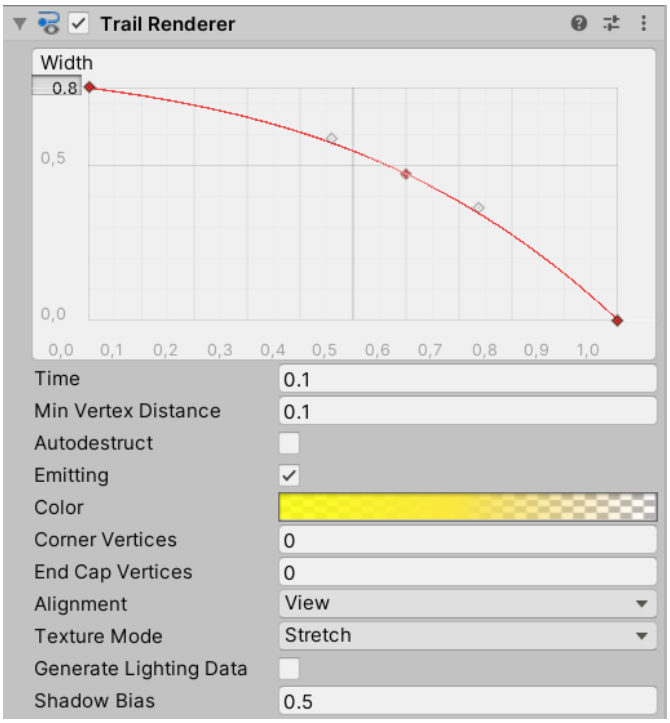

Рисунок 4.5 — Компонент Trial Renderer ігрового об’єкта снаряда

Компонент Projectile реалізує політ ігрового об’єкта снаряду до цілі, та наслідує клас MonoBehaviour. Швидкість польоту снаряду та його траєкторія залежить від зброї яка його випустила. Снаряд може мати пряму траєторію, якщо значення параметра швидкості повороту снаряду дорівнює нулю. Якщо значення параметра більше нуля, то під час польоту снаряд розвертається у сторону цілі. Також, компонент Projectile містить дані про:

- Шкоду яку наносить снаряд;
- Який додатковий ефект має снаряд;
- Який ігровий об’єкт випустив снаряд;
- В який ігровий об’єкт було випущено снаряд;
- Візуальний ефект знищення снаряда.

На початку створення снаряду функція CalculateDamage обчислює чи наносить снаряд критичну шкоду. Якщо ігровий об’єкт снаряду зіштовхується з будь-яким іншим ігровим об’єктом (який не є тим хто випустив снаряд), то на його місці створюється ефект знищення снаряда, а сам снаряд знищується.

Компонент TriggerEnter належить ігровим об’єктам як героя, так і противника, та реалізує зіткнення снаряда з об’єктом. Метод OnTriggerEnter викликається у FixedUpdate при зіткненні двох колайдерів ігрових об'єктів. Далі відбувається перевірка, чи є снаряд ворожим. Якщо це так, викликається метод ModifyHealth з компоненту PlayerStats, який змінює характеристику здоров’я, залежно від того яку шкоду наніс снаряд.

Наступні компоненти: Push, Shake, HitMaterialChange та ScreenShake реалізують ефекти які безпосередньо не впливають на ігровий процес. Дані компоненти виконують естетичну функцію та значно підсилюють відчуття від попадання снаряда.

Компонент Push реалізує штовхання цілі після того, як в неї потрапив ворожий снаряд. Сила поштовху залежить від характеристики Mass ігрового об’єкта цілі та характеристики PushMultiplier зброї. Поштовх відбувається у зворотному напрямку до ігрового об’єкта стрільця, шляхом зміни параметру velocity в компоненті RigidBody.

Компонент Shake реалізує ефект деформації ігрового об’єкта після того, як в нього потрапив ворожий снаряд. Ефект полягає в тому, що 3D-модель гравця або противника починає швидко трястись та змінювати розмір упродовж однієї секунди. Сила деформації може регулюватись за допомогою параметра shake_intensity.

Компонент HitMaterialChange реалізує ефект спалаху моделі гравця або противника після того, як в нього потрапив ворожий снаряд. Даний компонент належить 3D-моделі ігрового об’єкту, та містить в собі список всіх матеріалів моделі. Ефект полягає в тому, що 3D-модель на 0.2 секунди змінює всі матеріали моделі на матеріал яскравого червоного кольору.

Компонент ScreenShake реалізує ефект тряски камери після того як в ігровий об’єкт потрапив ворожий снаряд. Компонент належить об’єкту камери та реалізує патерн проєктування Singleton для простішого доступу до нього з будь-якого компонента.

Ігровий об’єкт зброї містить в собі 3D-модель зброї, а також два компоненти: SpawnProjectile та ProjectilePool.

Методи Instantiate та Destroy необхідні для створення та знищення ігрових об’єктів прямо під час ігрового процесу, бо вони вимагають мінімального часу CPU. Однак такі ігрові об’єкти як снаряди мають дуже короткий термін служби та знищуються у величезній кількості. Це вимагає значно більшого часу роботи процесора.

Крім цього, ігровий рушій Unity використовує Garbage Collection для звільнення пам’яті, а повторні виклики методу Destroy запускають його, що сповільнює роботу CPU та заважає ігровому процесу. Для мобільних пристроїв, ресурси яких сильно обмежені, це має вирішальне значення.

Через це, для створення снарядів було вирішено використовувати патерн Object pool, який реалізовано в компоненті ProjectilePool. Компонент попередньо створює всі ігрові об’єкти снарядів для кожної зброї під час завантаження сцени, та робить їх неактивними. А під час ігрового процесу, гра повторно використовує снаряди з "пулу", замість створення нових. Для кожної зброї задається своя кількість снарядів, в залежності від її скорострільності.

Компонент SpawnProjectile реалізує створення ігрових об’єктів снарядів. Він містить в собі посилання на: скриптовий об’єкт з даними про зброю, компонент RigidBody стрільця, компонент ProjectilePool, та на координати точки спавну снарядів. Останнім, є пустий ігровий об’єкт який знаходиться в ієрархії ігрового об’єкта героя. Даний компонент наслідує MonoBehaviour.

Однойменний метод SpawnProjectile у якості аргументу приймає посилання на ігровий об’єкт цілі, після чого створює снаряд за допомогою ProjectilePool, передаючи в нього дані про ціль та шкоду яку він наносить. Снаряд створюється на точці спавну, а його значення Rotation обчислюється функцією LookRotation, яка у якості аргументу приймає різницю векторів координат цілі та стрільця. Характеристика зброї Spreading визначає випадкову похибку значення Rotation при створенні снарядів, тобто якщо вона більше ніж 0, снаряди будуть летіти не точно в ціль.

Метод ShootPush реалізує віддачу від стрільби, та кожен раз викликається в SpawnProjectile. Поштовх відбувається у зворотному напрямку від ігрового об’єкта цілі, шляхом зміни параметру velocity в компоненті RigidBody. Сила віддачі залежить

від характеристики Mass ігрового об’єкта стрільця та характеристики RecoilMultiplier зброї.

Компонент WeaponManager реалізує роботу з різними типами зброї. Він належить тільки ігровому об’єкту героя, адже на відміну від нього, противник не може мати більше одного об’єкта зброї. Герой може мати будь-яку кількість зброї, але в момент часу активною може бути лише одна. Даний компонент містить в собі список всіх об’єктів зброї, які має герой та виконує дві основні функції: додавання нової зброї та зміна поточної зброї.

Метод CreateWeapon приймає в якості аргументу скриптовий об’єкт з даними про зброю. Такими даними є: 3D-модель зброї, префаб ігрового об'єкта снаряду, та характеристики зброї. На основі цього, за допомогою методу Instantiate створюється новий ігровий об’єкт зброї, який поміщається в ієрархію об’єкта героя.

Метод SetWeapon приймає в якості аргументу посилання на об’єкт зброї та робить її активною, а всі інші об’єкти зброї — неактивними. Після цього, в компоненті Animator ігрового об'єкта героя, метод змінює анімацію атаки та швидкість анімації (FireSpeedMultiplier) в залежності від характеристик зброї. Для цього було створено декомпозицію блоку Attack (Рисунок 4.5) в State Mashine, яка представлена на рисунку 4.6.

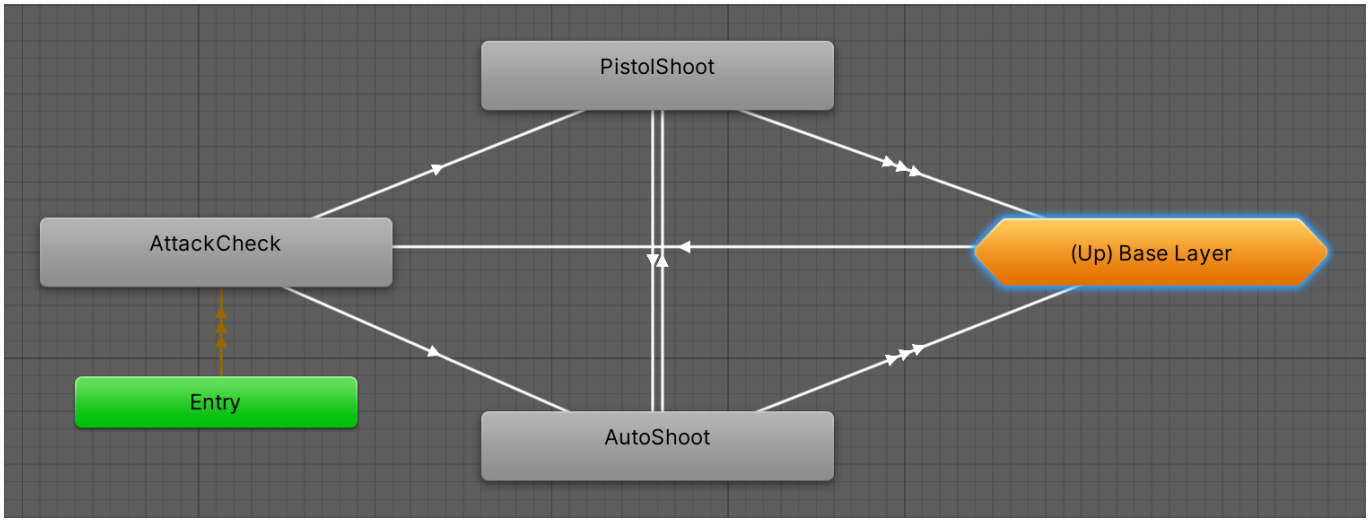

Рисунок 4.6 — State Mashine декомпозиції блоку Attack, для ігрового об’єкта героя

Кожна зброя може мати один з 7 додаткових ефектів. Ефекти з’являються коли снаряди зброї потрапляють у противника. Вони необхідні для створення більшої варіативності ігрового процесу. Всі ефекти та їх короткий опис представлений в таблиці 4.1.

Таблиця 4.1 — Додаткові ефекти зброї

Назва ефекту	**Опис ефекту**
Fire	Нанесення додаткової шкоди противнику кожні пів секунди, упродовж трьох секунд, якщо він не рухається. Розмір шкоди дорівнює 1/5 від шкоди, яку наносить снаряд. Якщо противник почне рухатись ефект припиняється.
Blood	Нанесення додаткової шкоди противнику кожні пів секунди, упродовж чотирьох секунд, якщо він рухається. Розмір шкоди дорівнює 1/10 від шкоди, яку наносить снаряд помножену на значення магнітуди velocity в компоненті RigidBody противника. (Тобто чим швидше рухається противник, тим більшу шкоду отримує. А якщо він не рухається, то не отримує шкоду, оскільки velocity дорівнює нулю)
Poison	Попадання кожного снаряду з даним ефектом поступово заповнює шкалу отруєння противника. Коли шкала стає повною, на нього накладається ефект отруєння, який полягає в поступовому нанесенні йому додаткової шкоди кожну секунду, доки противник не помре. Розмір шкоди дорівнює 3% від максимальної кількості здоров’я противника.
Freeze	Уповільнення противника на 5 секунд, шляхом зменшення швидкості його пересування, зменшення швидкості снарядів, та уповільнення анімацій у два рази. (Оскільки темп стрільби залежить від анімації стрільби, то він також сповільнюється).

Таблиця 4.1 — Додаткові ефекти зброї

Назва ефекту	**Опис ефекту**
Weakness	В два рази зменшення шкоди яку наносять снаряди противника на 6 секунд.
Electricity	Після потрапляння снаряду в противника, випускаються нові снаряди (у вигляді блискавок) по противниках, які знаходяться в його радіусі поля зору. Розмір шкоди від додаткових снарядів дорівнює 1/3 від шкоди, яку наносить основний снаряд.
Curse	Після потрапляння снаряду, противник отримує статус «проклятий» на 5 секунд. Одночасно с цим, всім противникам, які мають даний статус, наноситься додаткова шкода, навіть, якщо снаряд в них не потрапив. Розмір додаткової шкоди дорівнює 1/3 від шкоди, яку наносить снаряд.

Компонент ElementalDamage реалізує всі вищеописані ефекти. Компонент належить ігровому об’єкту противника та наслідує клас MonoBehaviour. В компоненті TriggerEnter, після того, як в колайдер потрапив снаряд, проводиться перевірка того, який ефект він має, і в залежності від цього викликається один з методів компоненту ElementalDamage: FireDamage, BloodDamage, PoisonDamage, FreezeDamage, WeakDamage, ElectricityDamage або CurseDamage.

4.1.4 Реалізація поведінки противників

Ігровий об’єкт противника, аналогічно до об'єкта гравця, має такі компоненти як: RigidBody, BoxColider, SpawnProjectile, ProjectilePool, TriggerEnter, FieldOfView та Push. Об’єкт противника має стани, які реалізовані в блок-схемі State Machine, за допомогою компонента Animation Controller. Дана схема представлена на рисунку 4.7.

Умовно поведінку противника можна поділити на дві частини: спокійна та режим бою. На початку кожного рівня противник просто переміщується по ігровому рівню. Але, коли в його поле зору потрапляє герой, він починає атакувати та переслідувати його. Тому, було визначено 6 ігрових станів в яких може перебувати противник:

- **Idle** — противник стоїть на місці та не рухається поки програється анімація Idle;
- **Move** — противник рухається у випадкову точку;
- **Alarm** — герой потрапляє у поле зору противника, і той переходить в режим бою;
- **Attack** — противник стріляє снарядами по герою;
- **Chase** — противник слідує за героєм, доки не наблизитися до нього на задану відстань;
- **Death** — програється анімація смерті противника, і його ігровий об’єкт знищується.

Між станами противника були визначені наступні переходи:

- **Idle** — **Move:** закінчилась анімація Idle;
- **Move** — **Idle:** противник перемістився до випадкової точки;
- **Будь-який стан** — **Alarm:** коли у поле зору противника потрапив герой;
- **Alarm** — **Move:** закінчилась анімація Alarm;
- **Move** — **Attack:** значення змінної CurrentAttackTime дорівнює 0;
- **Attack** — **Move:** закінчилась анімація стрільби;
- **Move** — **Chase:** герой знаходиться на занадто великій відстані від противника, або між ними є перешкоди;
- **Chase** — **Move:** герой знаходиться на нормальній відстані від противника, та між ними немає перешкод;
- **Будь-який стан** — **Death:** коли значення здоров’я противника менше або дорівнює нулю.

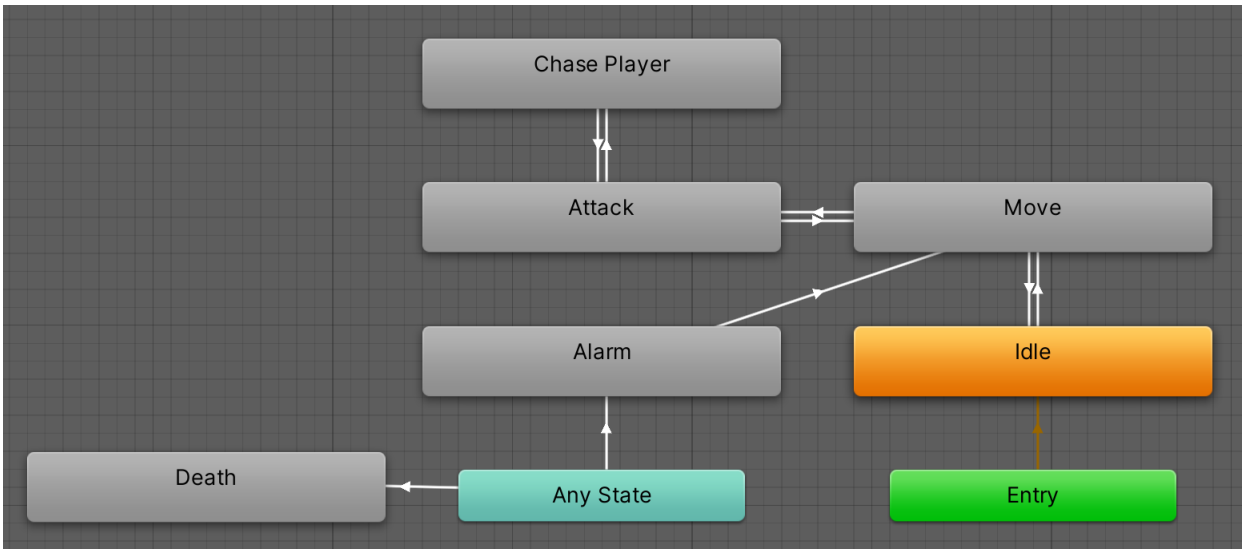

Рисунок 4.7 — State Mashine декомпозиції блоку Attack, для ігрового об’єкта героя

Для реалізації даних станів були створені відповідні компоненти, які наслідують клас StateMachineBehaviour: EnemyMove, EnemyAttack, та EnemyChase. EnemyMove реалізує процес переміщення у випадкову точку, який відбувається наступним чином:

1) Обирається випадкова точка в заданому радіусі від ігрового об’єкта;
2) Від ігрового об’єкта до точки пускається промінь (Raycast), щоб визначити чи є між ними перешкоди;
3) Якщо знайдено перешкоду — повернення до пункту 1, якщо ні — пункт 4;
4) Ігровий об’єкт противника переміщується до точки. Переміщення здійснюється за допомогою NavMesh.

NavMesh (навігаційна сітка) — абстрактна структура даних, яка використовується в ігровому рушії Unity для реалізації пошуку шляху між двома точками, в ігровій сцені. NavMesh робить так, щоб противник проходив через ігровий рівень слідуючи його точним межам та обходячи усі перешкоди та, наприклад не упирався в стіни рівня або не падав з нього.

В контексті NavMesh, рівень можна розглядати як сітку, яка утворена за допомогою полігонів та ліній. А сутності, які взаємодіють з цією сіткою називаються агентами. В нашому випадку, таким агентом є противник.

Основними компонентами для роботи з NavMesh є:

– **NavMeshSurface:** представляє область для переміщення агента NavMeshAgent, а також визначає частину рівня, де на початку гри повинна бути побудована навігаційна сітка. Даний компонент належить ігровому об’єкту підлоги рівня;

– **NavMeshAgent:** належить ігровому об’єкту противника, та реалізує його переміщення по сцені за допомогою NavMesh;

– **NavMeshObstacle:** належить ігровим об’єктам стін рівня, агент буде ігнорувати об’єкт з даним компонентом при побудові шляху, та буде його обходити під час руху.

На рисунку 4.8 представлені налаштування компонентів NavMeshSurface та NavMeshAgent. Після налаштування всіх компонентів, необхідно побудувати NavMesh. В "Heroes of Eternal" використовується процедурна генерація рівнів, то навігаційна сітка повинна будуватись під час ігрового процесу, що може значно збільшити час завантаження кожного рівня. Оскільки, в проєкті не має складних сцен, то дана проблема не є суттєвою. Побудова NavMesh відбувається за допомогою метода компоненту NavMeshSurface — BuildNavMesh. На рисунку 4.9 представлений приклад побудованої сітки, згенерованого рівня гри.

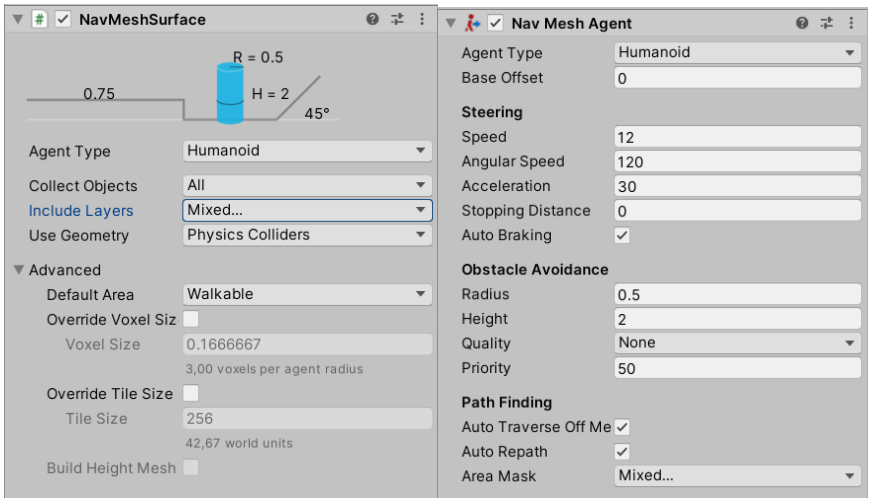

Рисунок 4.8 — Компоненти NavMeshSurface та NavMeshAgent

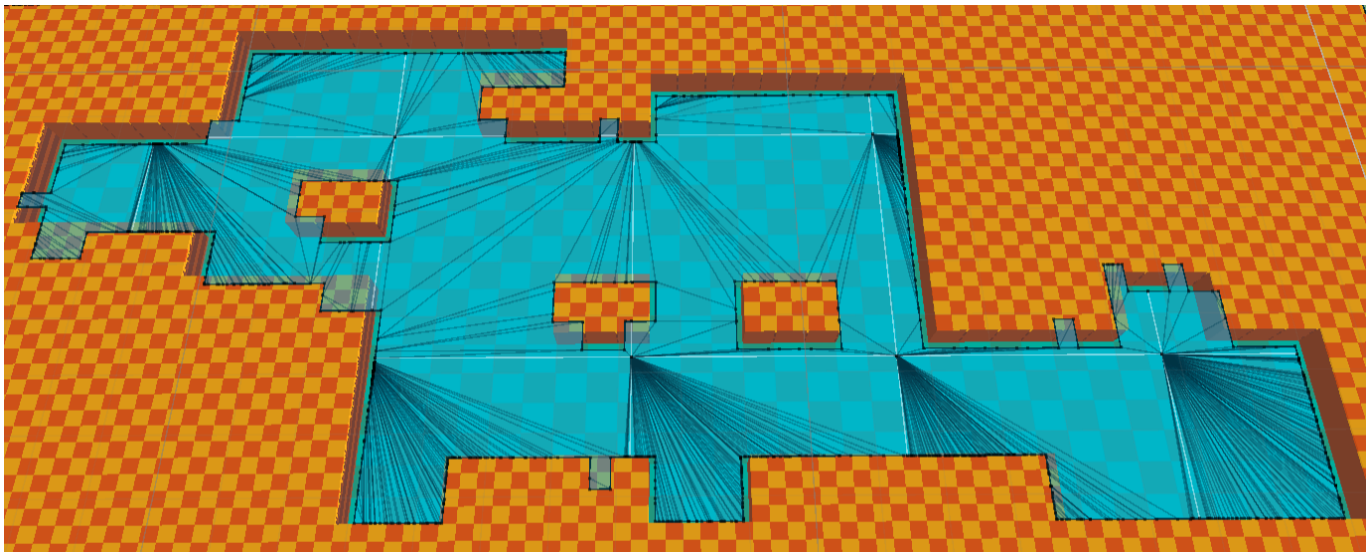

Рисунок 4.9 — Побудований NavMesh

Коли противник знаходиться в стані Attack, компонент EnemyAttack випускає промінь (Raycast) від координат положення противника, до положення героя, щоб перевірити, чи є між ними перешкоди у вигляді стін рівня. Якщо це так, противник переходить у стан Chase. За допомогою методу NavMesh SetDestination, який у якості аргументу приймає координати героя, противник переміщується до нього найкоротшим шляхом, обходячи всі перешкоди. Під час цього, в метді Update, кожну секунду випускається промінь для перевірки на наявність перешкод. Якщо їх більше немає, противник знову переходить у стан Attack.

Компонент EnemyStats належить ігровому об’єкту противника, та реалізує роботу з його характеристиками. Він містить посилання на скриптовий об’єкт з даними про характеристики, та працює аналогічно до PlayerStats.

Компонент EnemyManager контролює активність противників та належить пустому ігровому об’єкту на сцені. Даний компонент реалізує патерн проєктування Singleton, оскільки він існує в одному екземплярі на сцені, та містить список усіх об’єктів противників на рівні. Активність залежить від кількості противників які перейшли в режим бою, але не померли. Фактично, вона задає те, як часто противники будуть атакувати героя, зменшуючи чи збільшуючи значення їх AttackTime (час між атаками). У функції Update, значення змінної CurrentAttackTime постійно

зменшується з плином часу, і коли воно досягає 0 — противник атакує. Після чого CurrentAttackTime знову набуває значення AttackTime.

Коли противників в режимі бою багато, то вони будуть рідше атакувати героя, а якщо мало то частіше. Тобто якщо новий противник переходить в режим бою, то значення AttackTime збільшується, а коли противник помирає, то навпаки зменшується. Це необхідно для кращого балансу ігрового процесу.

4.1.5 Налаштування ігрової камери

В ігровому рушії Unity, ігрова камера це пристрій, який захоплює та відображає ігрову сцену для гравця. Для її реалізації було використано набір інструментів Cinemachine.

Cinemachine — це модульний набір інструментів для роботи з камерою в ігровому рушії Unity. Це офіційний плагін Unity, який дозволяє додавати додаткову функціональність до вже наявних ігрових камер. Cinemachine дозволяє створювати віртуальні камери, які мають дочірні компоненти, що реалізують такі функції як: плавне слідування камери за ігровим об’єктом, ефект тряски камери з процедурним шумом, задавання меж для камери та ін. Налаштування компонентів віртуальної камери представлене на рисунку

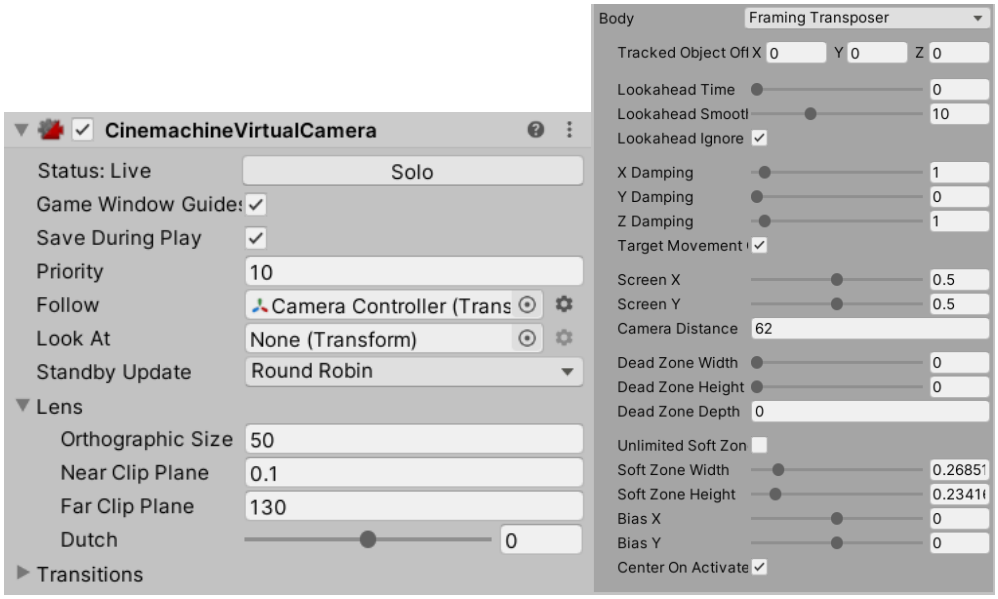

Рисунок 4.10 — Налаштування віртуальної камери

4.2 Реалізація процедурної генерації рівнів

Клас LevelData представляє шаблон для даних, які необхідні при генерації ігрового рівня. До цих даних належать:

– Розміри рівня;

– Кількість очків досвіду, які гравець отримує за проходження рівня;

– Список об’єктів класу EnemyData, який визначає які противники, та в якій кількості будуть створені на рівні;

– Чи отримує герой нову зброю в кінці рівня.

Скриптовий об’єкт LevelManager призначений для збереження та редагування даних про ігрові рівні гри. Даний компонент містить список об’єктів класу LevelData, кожен з яких представляє окремий ігровий рівень. Також він містить значення поточного рівня, на якому знаходиться гравець. На рисунку 4.11 представлений приклад заповнення скриптового об'єкта LevelManager в Unity Editor.

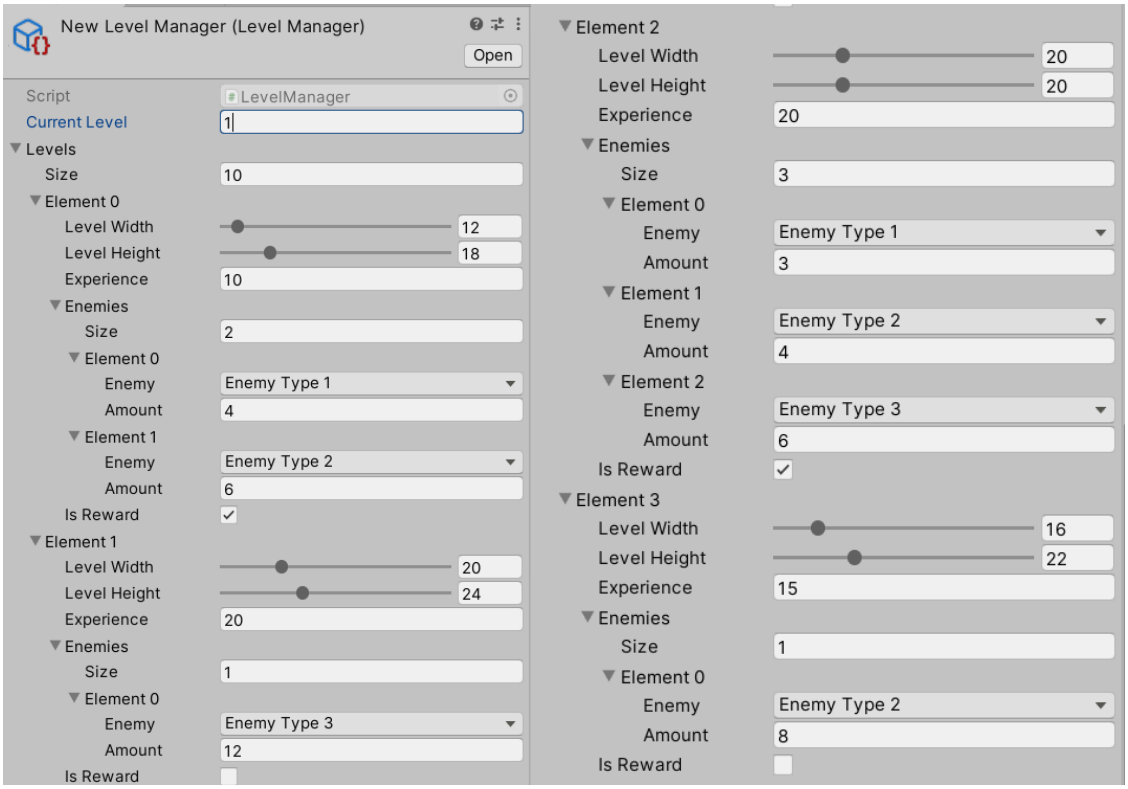

Рисунок 4.10 — Скриптовий об'єкт LevelManager в Unity Editor

Компонент CreateLevel реалізує створення та налаштування ігрового рівня "Heroes of Eternal". Процес створення рівня складається з наступних етапів:

1) Зчитування даних відповідного рівня LevelData;
2) Генерація даних ігрового поля, в компоненті LevelGeneration;
3) Налаштування ігрового об’єкта героя, та визначення його початкового місця на рівні;
4) Створення ігрових об’єктів противників, в компоненті EnemySpawn;
5) Створення ігрових об’єктів монет, в компоненті ObjectSpawn.

Компонент LevelGeneration реалізує процедурну генерацію даних ігрового рівня за алгоритмом. Він належить пустому ігровому об’єкту на сцені. Алгоритм процедурної генерації, який використовується в "Heroes of Eternal", описаний в пункті 2.2 "Методи дослідження". Дані рівня представлені у вигляді двовимірного масиву клітин, які належать до одної з двох констант: empty та wall.

Метод PrepareLevel створює масив ігрового поля, з усіма значеннями wall за замовчуванням. Після чого обирає випадкову точку, та створює на її місці змінну структури levelDot. Дана структура містить в собі значення її позиції, та напрямку руху. Якщо змінна levelDot знаходиться на місці клітини поля, клітина набуває значення empty.

Метод CreateEmpty виконує наступні дії в циклі:

- Рухає всі змінні levelDot в їхньому напрямку на одну клітину ;
- Випадково змінює напрямок змінних levelDot (шанс визначається змінною ChanceDotChangeDir);
- Випадково створює нову зміну levelDot (шанс визначається змінною ChanceDotSpawn, а максимальну можливу кількість точок визначає змінна MaxDots);
- Випадково знищує змінну levelDot (шанс визначається змінною ChanceDotDestoy).

Цикл закінчується, коли масив клітин, заповнений значеннями empty, на відповідну кількість відсотків, які визначаються змінною PercentToFill. Після генерації даних рівня, необхідно створити його на ігровій сцені. Для цього існує метод SpawnLevel, який за допомогою Instantinate спавнить ігрові об’єкти стін та підлоги.

Скриптовий об’єкт GameData містить в собі дані ігрового забігу, та виконує функцію їх збереження. Цими даними є: посилання на скриптовий об’єкт з даними обраного героя, список скриптових об’єктів з даними зброї яку має герой, та кількість монет та очків досвіду, яку гравець отримав під час ігрового забігу.

Метод SpawnPlayer виконує пошук випадкового стартового місця для ігрового об’єкта героя. Єдина вимога — герой завжди розпочинає на краю рівня. Після чого відбувається налаштування характеристик героя, та його зброї.

Компонент SpawnEnemies створює ігрові об’єкти противників на випадкових об’єктах підлоги на рівні. Єдина вимога — противник повинен знаходитись на значній відстані від об’єкта гравця.

Компонент ObjectSpawn створює об’єкти монет на випадкових об’єктах підлоги на рівні. Ігровий об’єкт монети містить в собі такі компоненти як: Rigidbody, та BoxColider.

Компонент GameManager реалізує переходи між ігровими рівнями. Перехід на наступний рівень відбувається після того, як герой переміг усіх противників на поточному рівні, на місті останнього померлого противника з’являється портал. Ігровий об’єкт порталу містить компоненти: RigidBody, SphereColider, та Portal. Останній викликає метод OnTriggerEnter коли герой входить в колайдер порталу, після чого надсилається запит в GameManager на перехід на наступний рівень (якщо він не останній).

4.3 Реалізація мета гри

Окрім основного ігрового циклу, в "Heroes of Eternal" є також елементи мета гри, яка відбувається між ігровими забігами. Вона відбувається в сцені головного меню, де гравець може купувати та покращувати предмети, які підсилюють його характеристики. Усі куплені предмети відображаються в інвентарі. На рисунку 4.11 представлений цикл мета гри (червоний), разом з головним ігровим циклом, та показаний їх взаємозв’язок.

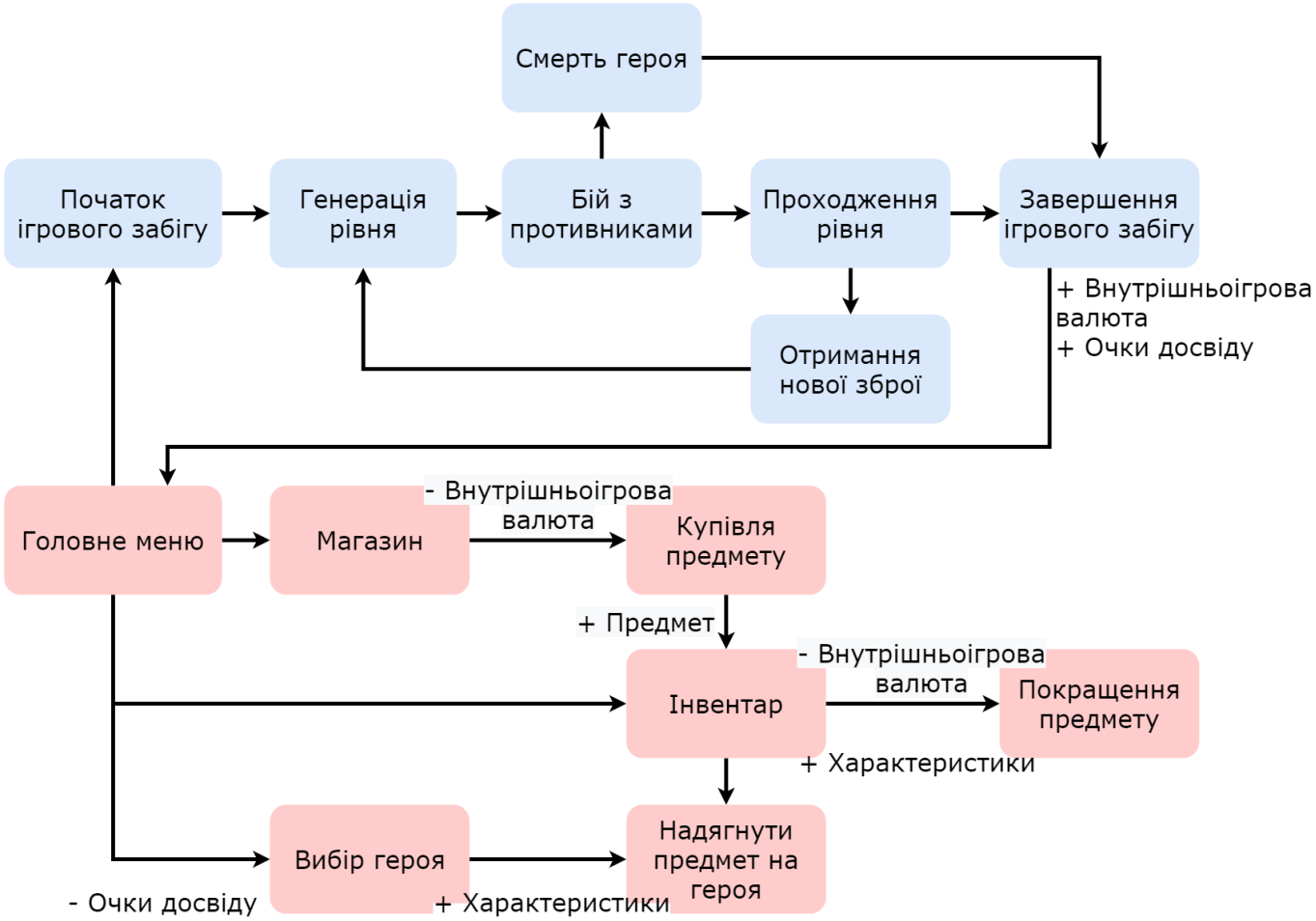

Рисунок 4.11 — Скриптовий об'єкт LevelManager в Unity Editor

Скриптовий об’єкт Item містить в собі дані про ігровий предмет. До них належать: назва предмета, 2D спрайт іконки предмета, та його опис. В методі OnValidate було згенеровано унікальне ID предмета, для копіювання скриптових об’єктів.

Скриптовий об’єкт EquipableItem наслідує Item, та містить в собі дані про ігровий предмет, який може бути надягнутим на героя. До них належать: посилання на скриптовий об’єкт героя, який надягнув предмет, рівень прокачування предмету, тип предмету, та колір предмету.

Кожен герой може надягнути по одному з чотирьох типів предметів: Helmet, Body, Ring та Boots. Від кольору предмету, залежить рідкість випадання предмета в магазині. Всього існує 5 типів кольорів предметів: Gray, Green, Blue, Violet, та Gold, де перший випадає найчастіше, а останній найрідше.

Також EquipableItem містить дані типу Stat про те які характеристики героя підсилює предмет, до них належать:

- **MaxHpBonus:** бонус до максимального здоров’я героя;
- **DamageBonus:** бонус до шкоди, яку наносить герой;
- **SpeedBonus:** бонус до швидкості переміщення героя;
- **SpeedAtackBonus:** бонус до швидкості атаки героя;
- **CritRateBonus:** бонус до шансу критичної шкоди;
- **CritDamageBonus:** бонус до критичної шкоди, яку наносить герой;
- **DodgeBonus:** бонус до шансу того, що попадання ворожого снаряду по герою не зарахується;
- **CoinsBonus:** бонус до кількості монет, які збирає герой під час ігрового процесу.

Метод Equip, у якості аргументу приймає скриптовий об’єкт даних героя, та реалізує процес надягання предмету на героя. Він додаючи модифікатори до відповідних характеристик героя типу Stat, тим самим збільшуючи їх. У свою чергу метод UnEquip робить абсолютно протилежну дію, видаляючи всі модифікатори цього предмету в об’єкта даних героя.

Метод ItemUpdate реалізує покращення предмету. За замовчуванням кожен предмет має рівень прокачування 1. При кожному виклику методу, значення характеристик бонусів які дають предмети збільшується на заданий відсоток, а рівень прокачування збільшується на 1. Максимальний рівень до якого можна прокачати EquipableItem — 10.

Компонент Shop реалізує купівлю випадкового предмета, в залежності від його кольору. Він має посилання на скриптовий об’єкт ShopData, який містить в собі усі скриптові об’єкти предметів гри. Метод CalculateItem обирає випадковий колір, що залежить від змінних частоти випадіння кожного кольору. Після чого, з ShopData обирається випадковий ігровий предмет цього кольору, та додається у інвентар гри, у вигляді його копії.

Компонент InventorySerializer реалізує збереження та завантаження даних інвентаря гри, що відбувається за допомогою методів LoadItemData та SaveItemData.

Збереження даних відбувається у бінарний файл, для цього використовується стандартний клас Unity — BinaryFormatter.

4.4 Реалізація моделей, анімацій та звуку

В 3D-редакторі Blender, були змодельовані три стилізовані моделі героїв, та одна модель противника. Оскільки "Heroes of Eternal" це мобільна гра, моделі є простими та мають мінімальну кількість полігонів, заради збільшення продуктивності роботи ігрового додатка. Після чого був виконаний їх імпорт в ігровий рушій Unity у форматі .obj. Моделі героїв представлені на рисунках 4.12 — 4.15.

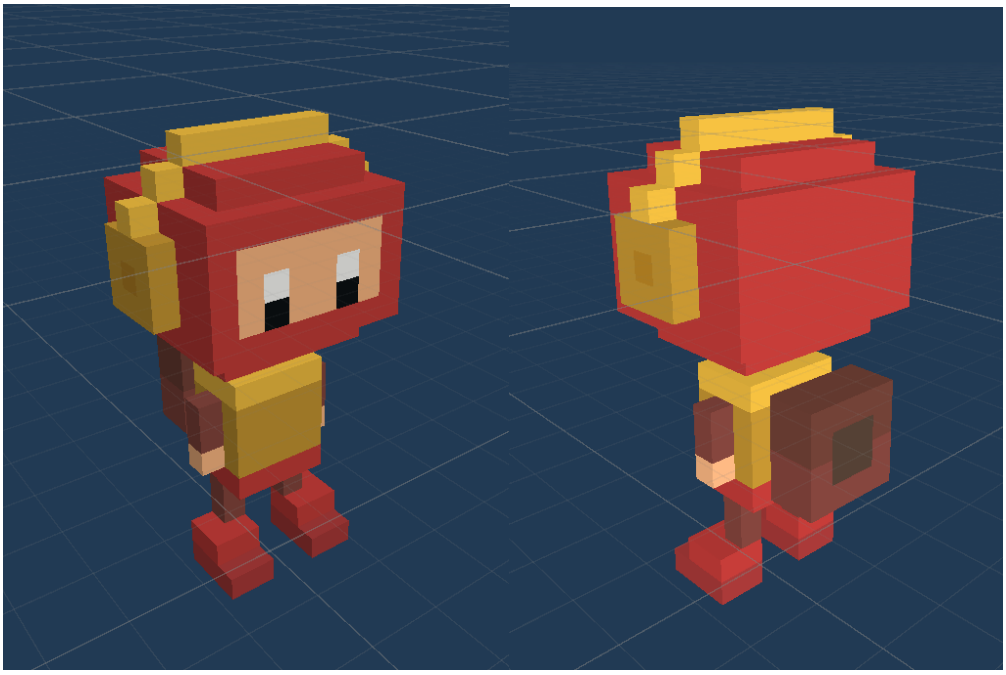

Рисунок 4.12 — Модель героя "Red"

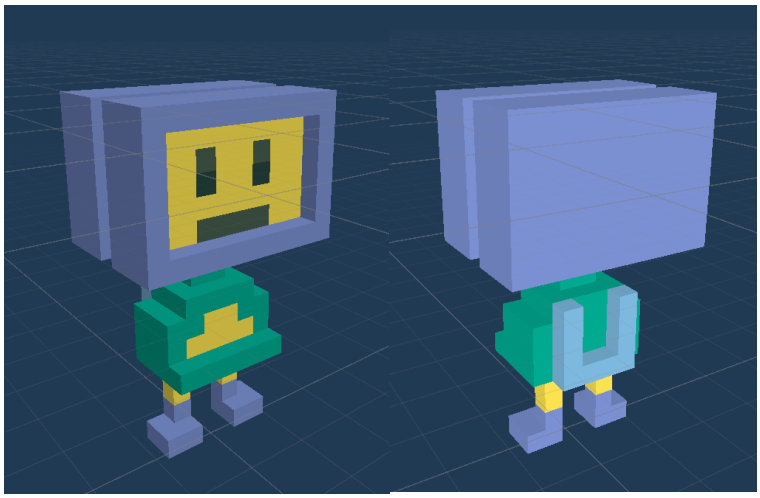

Рисунок 4.13 — Модель героя "Robot"

Рисунок 4.14 — Модель героя "Cat"

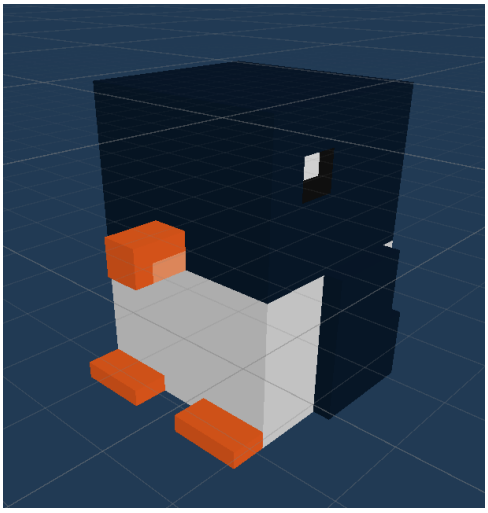

Рисунок 4.15 — Модель противника

Усі анімаційні кліпи були створені безпосередньо в ігровому рушії Unity. На рисунку 4.16 представлені таймлайни анімацій героїв, а саме анімацій переміщення та стрільби з двох видів зброї.

Створення снарядів відбувається за допомогою AnimationEvent, які дозволяють викликати методи ігрового об’єкта, під час програвання відповідного кадру анімації (AnimationEvent стрільби, позначений стрілкою на рисунку 4.16). Для моделі противника були створені анімації: переміщення, атаки, знаходження гравця та смерті

.

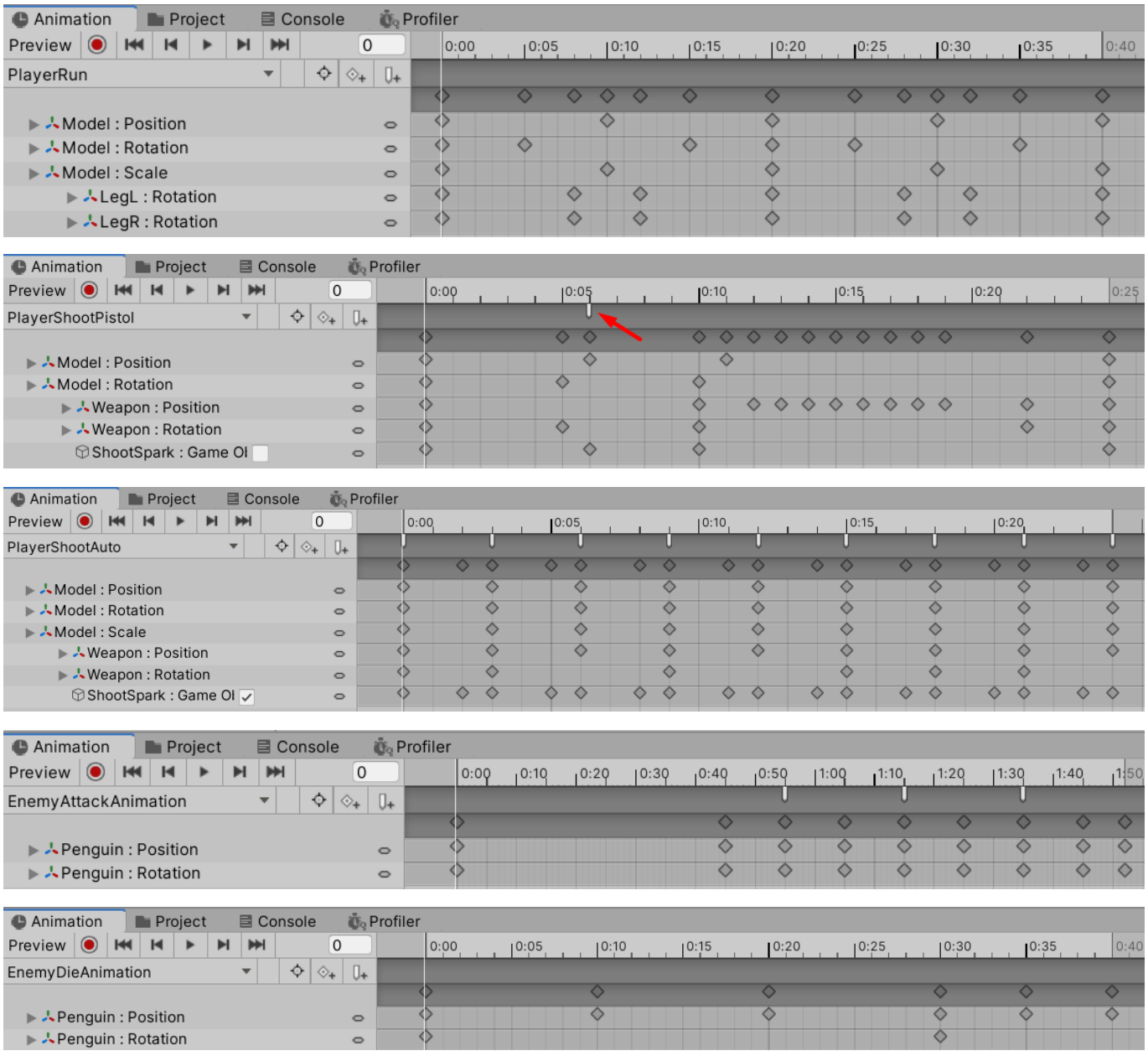

Рисунок 4.16 — Таймлайни анімацій героя та противника

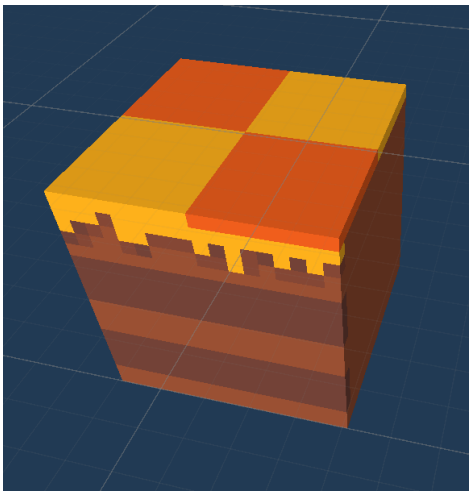

Рисунок 4.17 — Модель стіни рівня

На рисунку 4.17 представлена модель стіни рівня зі створеною текстурою.

Клас Sound описує звук, який програється під час ігрового процесу. Він містить такі змінні як: назва звуку, посилання на сам звуковий файл, а також значення гучності та висоти звуку.

Компонент SoundManager реалізує роботу зі звуком в ігровому додатку. Він містить список усіх об’єктів класу Sound та два методи: SetSoundVolume та PlaySound. Перший, змінює значення гучності звуку в усіх об’єктах класу Sound. Метод PlaySound, у якості аргументу приймає назву звуку, після чого запускає відповідний аудіокліп.

4.5 Реалізація інтерфейсу

4.5.1 Реалізація головного меню

Гравець потрапляє в головне меню гри "Heroes of Eternal" одразу після завантаження додатку. Меню складається з чотирьох панелей:

- **Головна:** на якій зображена назва гри, та кнопка старту ігрового забігу;
- **Налаштування:** де гравець може змінити мову, та гучність звуку;

– **Інвентар:** де зображені усі предмети, які має гравець, а також модель персонажа, його основні характеристики та надягнуті предмети. Натиснувши на предмет в інвентарі, відкривається вікно предмету, де гравець може переглянути його характеристики, покращити його та надягнути на обраного героя. Натиснувши на кнопку вибору героя, відкривається відповідне вікно, де гравець може змінити поточного героя;

– **Магазин:** де гравець може купити один з двох скринь з випадковими ігровими предметами.

На рисунку 4.18 представлена панель з кнопками навігації по головному меню, яка постійно знаходиться внизу екрану. Вона має 5 кнопок, 4 для кожної з перерахованих вище панелей, плюс одна додаткова, яка буде реалізована у майбутньому.

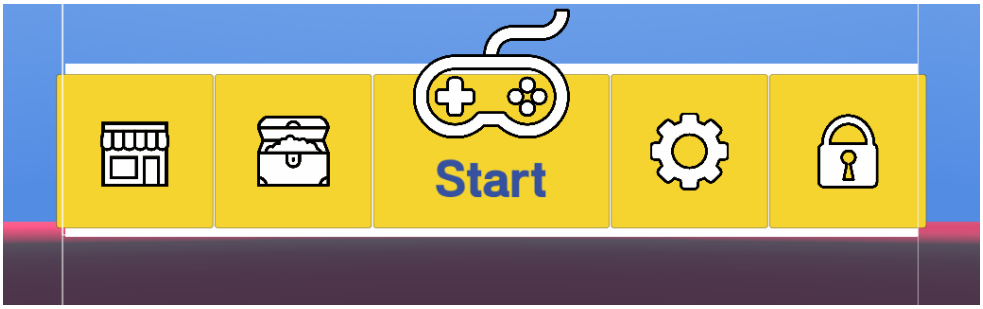

Рисунок 4.18 — Панель кнопок навігації

Компонент MainMenu реалізує навігацію по головному меню, за допомогою панелі кнопок навігації**,** а також їх анімацію. Анімація інтерфейсу створена за допомогою iTween. Це інструмент створення анімацій для ігрового рушія Unity, який являє собою систему інтерполяції, що приймає одне значення та протягом заданого проміжку часу анімує його на інше. iTween дозволяє створювати анімації в самому коді, без додавання компонента Animation Controller та без створення анімаційних кліпів.

Компонент Inventory реалізує інвентар гри. За допомогою компонента InventorySerializer він завантажує дані про всі ігрові предмети, що має гравець, після чого виводить їх в об’єкти комірок предметів, які мають компонент ItemSlot. Метод

Усі об’єкти комірок знаходиться в ієрархії ігрового об’єкта з компонентом Grid Layout Group, який поміщає свої дочірні об’єкти компонування в сітку. На рисунку 4.19 зображені налаштування сітки комірок предметів.

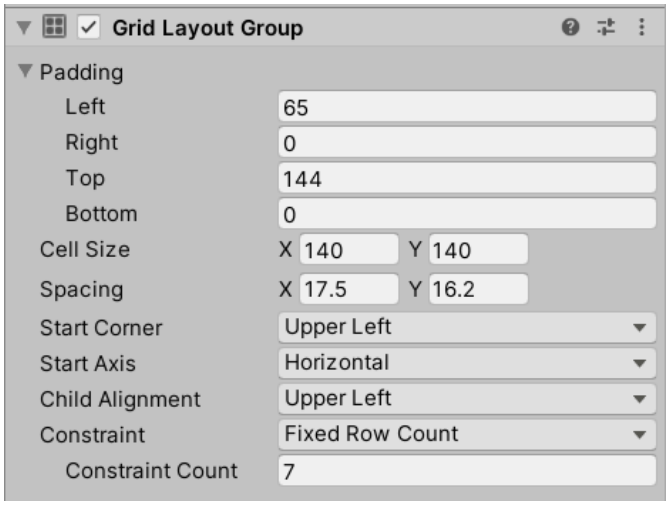

Рисунок 4.19 — Панель кнопок навігації

Компонент EquiqmentPanel реалізує панель які відображає предмети, які надягнуті на героя. Вона містить посилання на чотири комірки типу EquiqmentSlot, який наслідує клас ItemSlot, та містить значення типу предмету. Метод AddItem у якості аргументу приймає скриптовий об’єкт EquipableItem, та передає його дані в EquiqmentSlot відповідного типу. Метод RemoveItem аналогічно приймає EquipableItem, та робить пустим EquiqmentSlot який містить дані цього предмету (якщо такий є).

Компонент ItemPanel реалізує відображення вікна з даними про відповідний предмет, яке відкривається коли гравець тапнув по об’єкту ItemSlot. Вікно відображає: назву предмету, його опис, поточний рівень, іконку, бонуси які надає предмет та кнопки які виконують функції надягання та покращення предмету. Вікно представлене на рисунку 4.20.

Компонент StatsPanel реалізує відображення панелі з основними характеристиками героя, таких як MaxHp, Damage та Speed.

Компонент InventoryManager реалізує процес надягання предмету на героя. Метод Equip, який викликається в ItemPanel, у якості аргументу приймає скриптовий

об’єкт EquipableItem, викликає в ньому метод UnEquip для попереднього предмета в цьому слоті, після чого метод Equip для обраного предмета. Далі, він додає його на панель EquiqmentPanel, методом AddItem, та оновлює дані в StatsPanel.

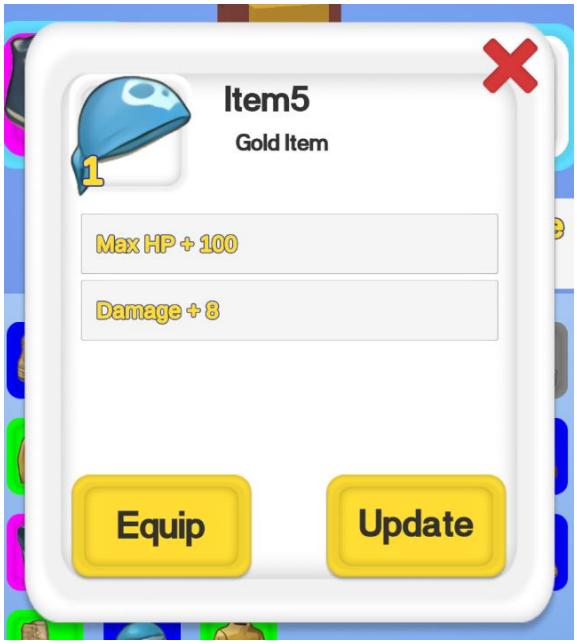

Рисунок 4.20 — Вікно ItemPanel

Компонент ItemPanel реалізує відображення вікна з героями та вибір поточного героя. Вікно відкривається коли гравець натискає кнопку "ChangeHero" на панелі інвентарю. Воно містить в собі: модель поточного героя, StatsPanel, та кнопки для вибору відповідних героїв.

Клас Localisation реалізує локалізацію тексту ігрового додатка трьома мовами: англійською, українською, та російською. Для цього було створено CSV файл, в якому зберігається всі текстові дані додатку. Кожен рядок файлу починається зі значення ключа, після чого йде відповідний текст англійською, українською, та російською. Клас Localisation має три колекції типу Dictionary для кожної мови. Клас CSVLoad завантажує файл, та розділяє текстові дані по відповідним словникам.

Компонент LocalisationUI додається до ігрових об’єктів, які мають компонент Text, та задає значення текстового поля відповідно до значення ключа, яке він приймає.

На рисунках 4.21 - 4.22 зображені всі панелі головного меню ігрового додатку.

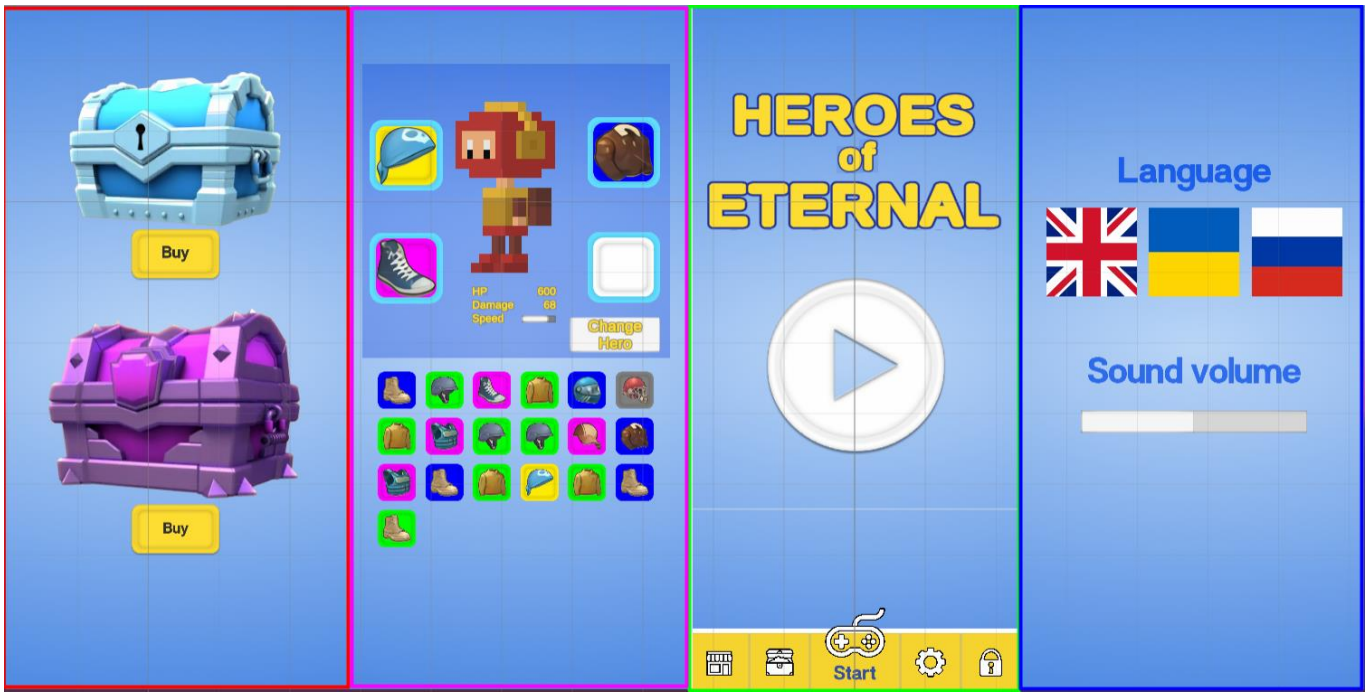

Рисунок 4.21 — Панелі головного меню: магазин, інвентар, стартовий екран, та опції

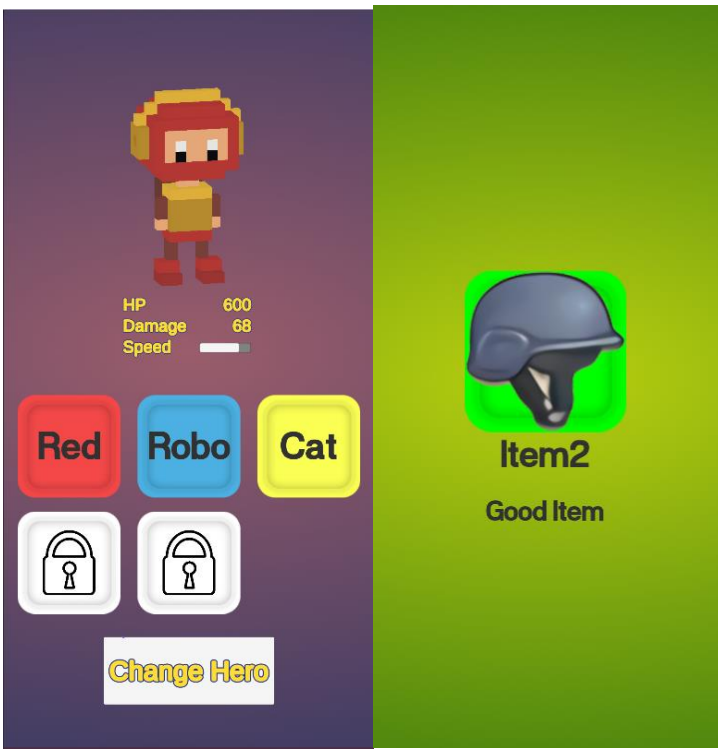

Рисунок 4.22 — Вікно вибору героя, та вікно анімації купленого предмету

4.5.2 Реалізація інтерфейсу під час ігрового процесу

Оскільки стандартний перехід з однієї ігрової сцени в іншу в ігровому рушії Unity є дуже різким, було вирішено створити відповідні анімації переходів. Було створено новий ігровий об’єкт з компонентом Canvas, та поміщено в нього 11 прямокутних спрайтів червоного то жовтого кольорів. Було зроблено дві анімації ShowLevelAnimation та HideLevelAnimation, які зображені на рисунку 4.23. Блок схема State Machine компонента Animator Controller представлена на рисунку 4.24.

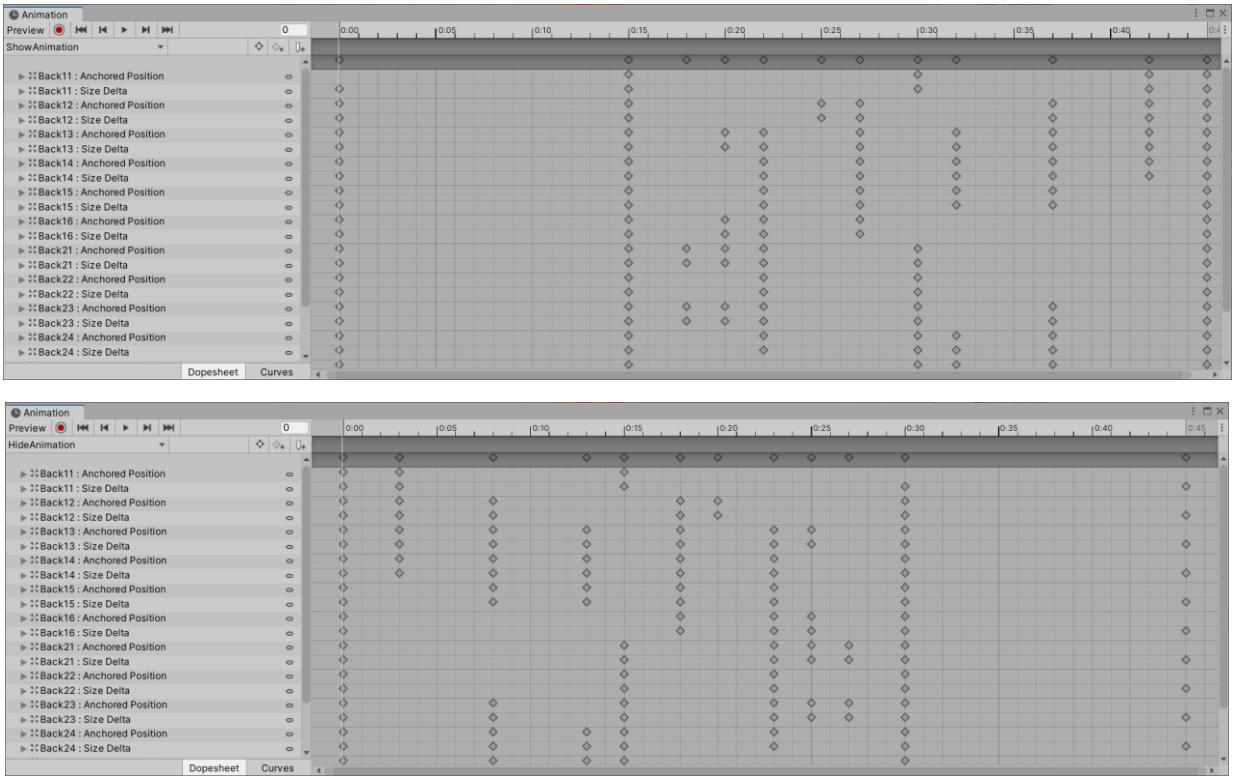

Рисунок 4.23 — Таймлайни анімацій переходу між рівнями

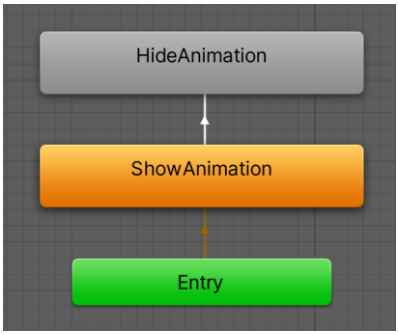

Рисунок 4.24 — Блок схема State Machine анімацій переходу між рівнями

Ігровий об’єкт HealthBar являє собою шкалу здоров’я героя та противника. Він містить в собі два компоненти Image (рисунок 4.25) та компонент HealthBar, який за допомогою метода HandleHealthModify змінює значення Fill Amount одного зі спрайтів, в залежності від значення здоров’я. Аналогічним чином, реалізована шкала отруєння противника.

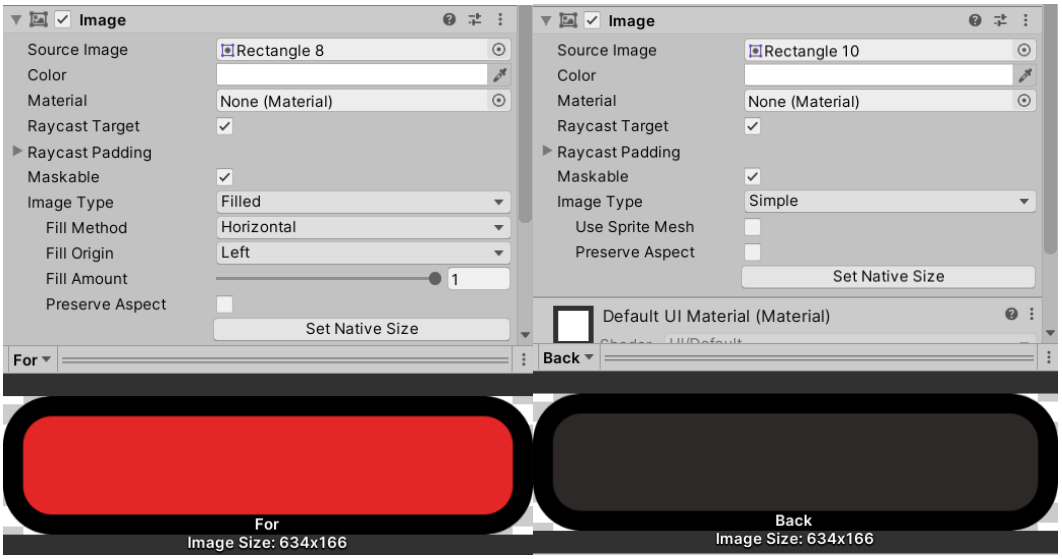

Рисунок 4.25 — Два компонента Image ігрового об’єкту HalthBare

Компонент DamageNumberSpawn створює вспливаючий текст, який з’являється після того, як противник отримав шкоду. Текст містить значення отриманої шкоди, та зникає через декілька секунд після закінчення анімації. Анімація створена за допомогою інструменту iTween, Даний ефект необхідний для того, щоб герой розумів яку шкоду він наносить противнику. На рисунку 4.26 зображена шкала здоров’я та шкода яка була нанесена противнику.

Рисунок 4.26 — Приклад роботи HealthBar та DamageNumberSpawn

Компонент WeaponsManagerUi реалізує інтерфейс для вибору поточної зброї під час ігрового процесу. Іконка поточної зброї знаходиться в об’єкті Canvas, у правому нижньому кутку. Після натискання на неї, справа наліво з’являються іконки усіх об’єктів зброї яку має герой. Натиснувши на потрібну іконку зброї, ця зброя стає основною, а іконки знову зливаються в одну.

Оскільки, потенціально герой може мати велику кількість зброї, іконки можуть почати виходити за межі екрану. Через це було вирішено реалізувати ефект прокрутки UI елементів, за допомогою компонентів Scroll Rect та Grid Layout Group. Їх налаштування представлені на рисунку 4.27. WeaponsManagerUi створює іконки зброї на початку рівня за допомогою методу Instantiate. Колір іконок залежить від типу додаткової шкоди, яку вони наносять. На рисунку 4.28 представлений приклад іконок зброї у згорнутому та розгорнутому стані.

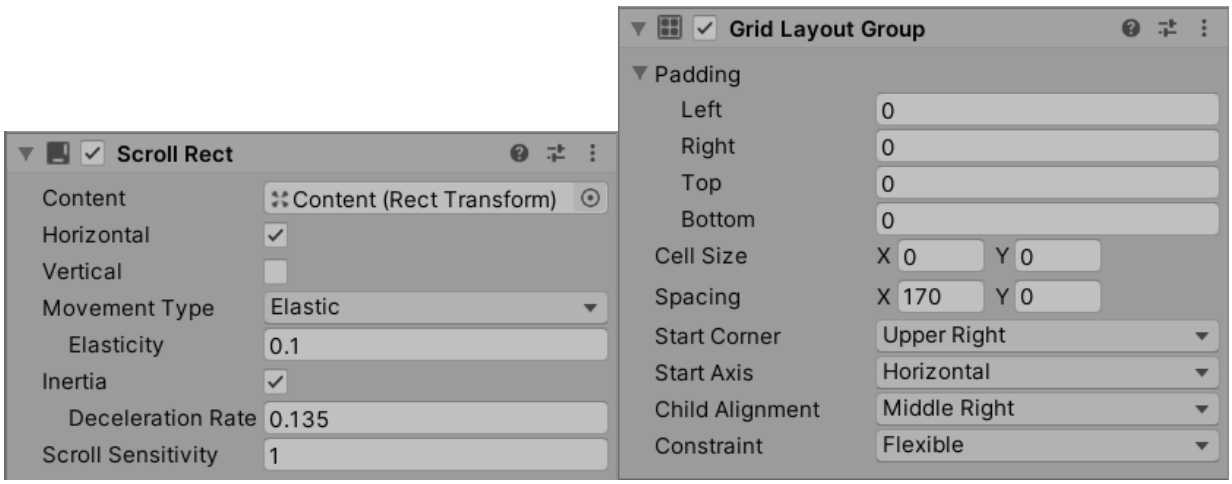

Рисунок 4.27 — Налаштування компонентів Scroll Rect та Grid Layout Group

Рисунок 4.28 — Іконки зброї

Після проходження рівня герой має шанс отримати нову зброю. Компонент WeaponGift реалізує інтерфейс вибору нової зброї, з трьох випадкових варіантів. Даний інтерфейс представлений на рисунку 4.29. Було також створено анімацію появи окремих іконок зброї по черзі, таймлайн якої зображений на рисунку 4.30.

Рисунок 4.29 — Інтерфейс вибору випадкової зброї

Рисунок 4.30 — Таймлайн анімації появи вибору випадкової зброї

Клас ExperienceSystem описує систему ігрового досвіду. За кожен ігровий забіг, гравець отримує певну кількість очків досвіду. Якщо він набирає достатню кількість досвіду, він переходить на новий рівень досвіду. Даний клас містить наступні змінні:

- Поточний рівень;
- Поточна кількість очків досвіду;
- Кількість очків досвіду до наступного рівня;
- На скільки з кожним новим рівнем збільшується кількість очків досвіду до наступного рівня.

Компонент ExperienceBar представляє шкалу підрахунку досвіду який герой отримав за ігровий забіг. Її реалізація аналогічна до HealthBar, але процес підрахунку супроводжується анімаціями, які створені за допомогою iTween. Компонент CoinsResult реалізує інтерфейс підрахунку монет, які герой заробив за ігровий забіг. На рисунку 4.31 представлений інтерфейс підрахунку результатів ігрового забігу.

Рисунок 4.31 — Інтерфейс підрахунку результатів ігрового забігу

4.6 Створення візуальних ефектів

Постобробка — це процес застосування різних ефектів та фільтрів до ігрового зображення. Гра рендерить своє вихідне зображення, однак перед його відправкою на монітор відбудеться ще один процес рендеринга, де на зображення накладаються ефекти та фільтри.

В "Heroes of Eternal" ефекти постобробки використовуються під час того, коли герой входить в режим прицілювання. В цей момент час сповільнюється, а на екран додаються ефекти зміни кольору. Для цього, в ігрову сцену було додано компонент Voleme. На рисунку 4.32 представлені налаштування ефектів постобробки даного компонента.

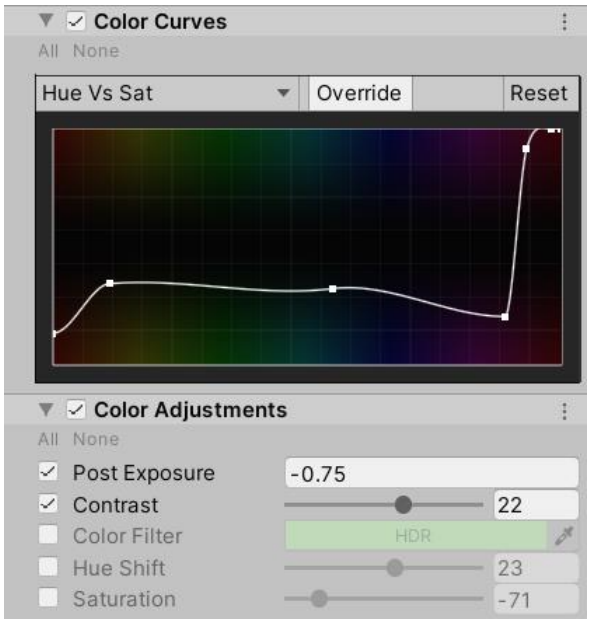

Рисунок 4.32 — Налаштування Volume

Шейдер — це невелика програма, яка містить в собі інструкції для графічного процесора комп'ютера, які описують те, як він повинен обчислити колір конкретного матеріалу. Раніше, написання шейдерів вимагало спеціальних мов програмування, таких як HLSL або Cg [27]. Unity Shader Graph дозволяє створювати шейдери візуально, за допомогою візуального програмування, та бачити результати в режимі реального часу. На рисунку 4.33 представлене створення шейдера порталу в Shader Graph. Готовий ігровий об’єкт порталу представлений на рисунку 4.34.

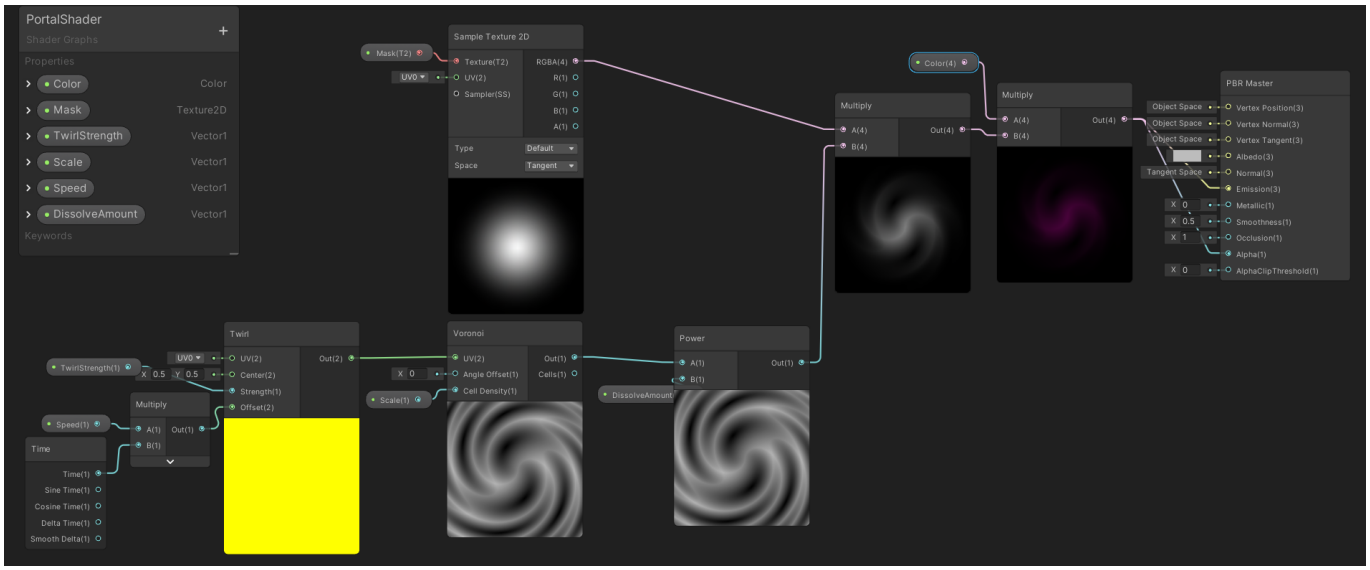

Рисунок 4.33 — Шейдер портал

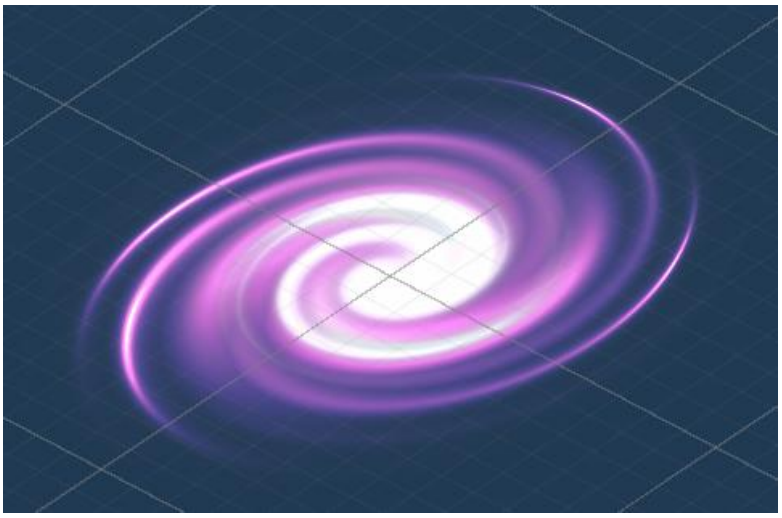

Рисунок 4.34 — Ігровий об’єкт порталу

Система часток дає змогу генерувати велику кількість частинок які мають коротку тривалість життя. Вони проходять окремий процес рендеринга, та дозволяють створювати тисячі об’єктів. Частинка — це будь-яка індивідуальна текстура, екземпляр матеріалу, або сутність, що породжена системою часток [28].

В "Heroes of Eternal" за допомогою Unity Particle System були створені наступні ефекти:

- Ефект попадання снаряду (Рисунок 4.35);
- Ефект вибуху противника (Рисунок 4.36);
- Ефект електричного снаряда (Рисунок 4.36).

На рисунку 4.37 представлені налаштування системи часток, для створення ефекту вибуху.

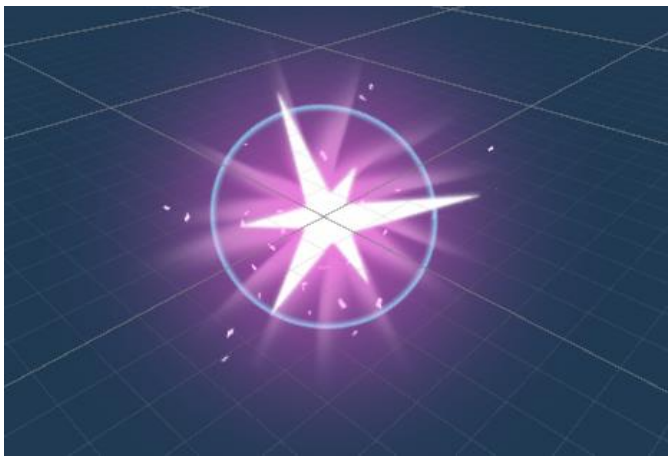

Рисунок 4.35 — Ефект попадання снаряду

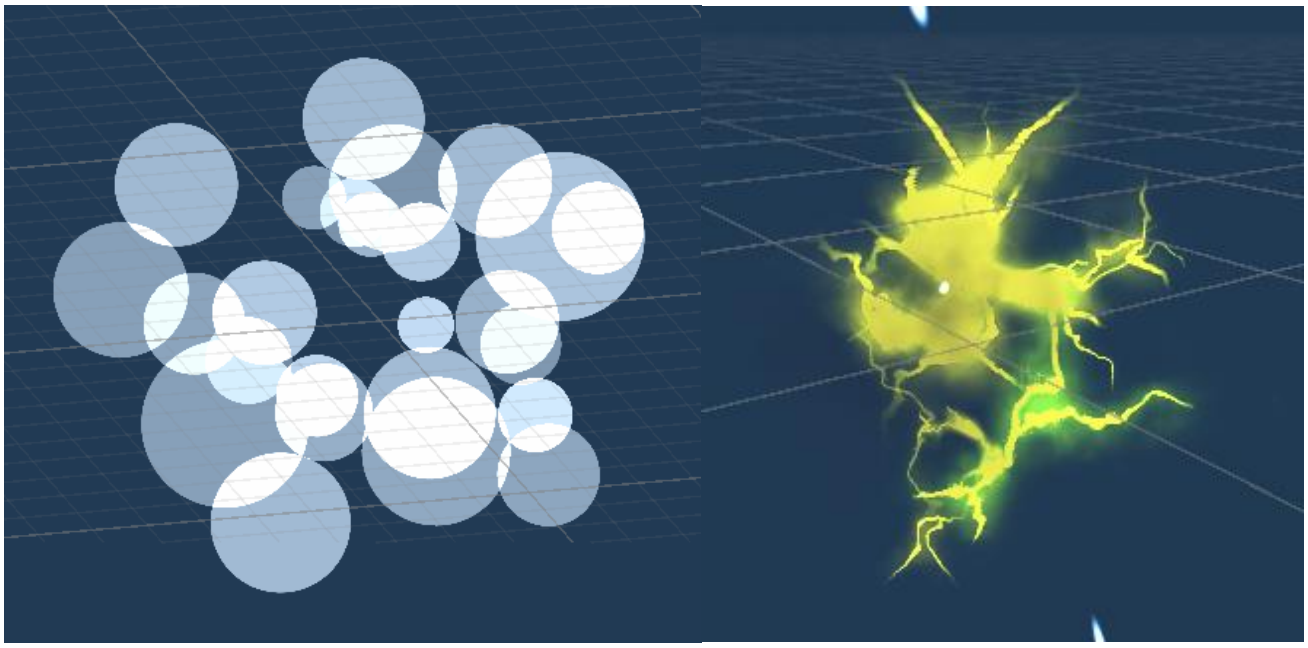

Рисунок 4.36 — Ефект вибуху та електричного снаряда

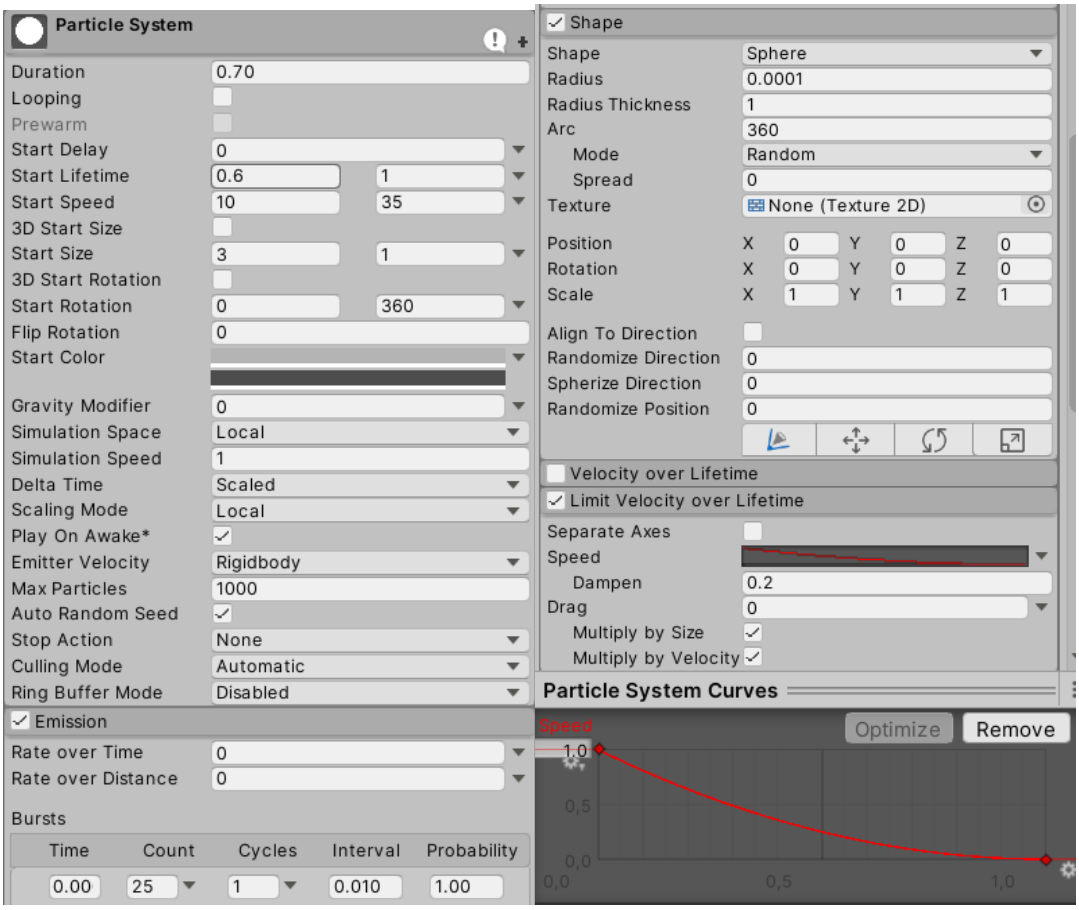

Рисунок 4.37 — Налаштування системи часток ефекту вибуху

ВИСНОВКИ

На початку розробки був проведений аналіз детальний предметної області мобільних ігрових додатків. Після чого, було вирішено розпочати розробку мобільного ігрового додатка, через високу актуальність даної теми.

Далі були проаналізовані існуючі аналоги мобільних додатків в жанрі action rougelike. На основі їх аналізу були визначені мета та задачі дослідження, що включали перелік завдань, які потрібно виконати під час розробки додатку. Після чого, були сформовані вимоги до програмного продукту. Було проведено дослідження методів та засобів реалізації.

Під час планування розробки додатку, були створені структурна декомпозиція робіт для організації командної роботи та організаційна структура проекту для відображення учасників та їх керівників. Після чого був побудований календарний графік розробки та описані можливі ризики при розробці проекту.

Наступним етапом проводилось проектування проекту ігрового додатка. Була створена контекстна діаграма роботи ігрового додатку та її декомпозиції, а також діаграма варіантів використання

У результаті був розроблений мобільний ігровий додаток в жанрі action rougelike. Були створені головний ігровий цикл, мета гра, процедурна генерація рівнів, 3D моделі, інтерфейс додатку, та візуальні ефекти.

СПИСОК ЛІТЕРАТУРИ

1. Video Game Industry - Statistics & Facts [Електронний ресурс] // Statista. – 2020. – Режим доступу до ресурсу: https://www.statista.com/topics/868/video-games/#dossierSummary__chapter1

2. Video Games Could Be a $300 Billion Industry by 2025 (Report) [Електронний ресурс] // Vervelogic. – 2019. – Режим доступу до ресурсу: https://variety.com/2019/gaming/news/video-games-300-billion-industry-2025-report-1203202672/

3. Infographic: Mobile Game Market Trends 2020 [Електронний ресурс] // Dot Com Infoway. – 2020. – Режим доступу до ресурсу: https://www.dotcominfoway.com/blog/infographic-mobile-game-market-trends-2020/#gref

4. Google Play Store Stats Of 2019-2020 You Should Know [Електронний ресурс] // Vervelogic. – 2020. – Режим доступу до ресурсу: https://www.vervelogic.com/blog/google-play-store-stats/

5. Games Rule The App Stores: Most Popular Genres Revealed 2020 [Електронний ресурс] // Localize Direct. – 2020. – Режим доступу до ресурсу: https://www.localizedirect.com/posts/most-popular-game-genres-revealed

6. Game [Електронний ресурс] // Wikipedia. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Game

7. Despain Wendy. 100 Principles of Game Design – New Riders – 2012 [Текст].

8. What Is A Meta-Game? [Електронний ресурс] // Gamasutra. – 2019. – Режим доступу до ресурсу: https://www.gamasutra.com/blogs/StanislavCostiuc/20190212/336413/What_Is_A_MetaGame.php

9. Roguelike [Електронний ресурс] // Wikipedia. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Roguelike

10. Julian Togelius, Mark J. Nelson. Procedural Content Generation in Games – Springer. – 2016 [Текст].

11. Archero [Електронний ресурс] // Google Play. – 2020. – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.habby.archero&hl=ru&gl=US

12. Nuclear Throne [Електронний ресурс] // Vlambeer. – 2015. – Режим доступу до ресурсу: https://vlambeer.itch.io/nuclear-throne

13. Enter the Gungeon [Електронний ресурс] // Steam. – 2016. – Режим доступу до ресурсу: https://store.steampowered.com/app/311690/Enter_the_Gungeon/

14. Hades [Електронний ресурс] // Steam. – 2020. – Режим доступу до ресурсу: https://store.steampowered.com/app/1145360/Hades/

15. Pokémon Quest [Електронний ресурс] // Nintendo. – 2018. – Режим доступу до ресурсу: https://www.nintendo.ru/-/-Nintendo-Switch/Pokemon-Quest-1382462.html

16. Game Engine and History of Game Development [Електронний ресурс] // Study Tonight. – 2020. – Режим доступу до ресурсу: https://www.studytonight.com/3d-game-engineering-with-unity/game-engine

17. Godot [Електронний ресурс] // Godot. – 2020. – Режим доступу до ресурсу: https://godotengine.org/features

18. Unity, whose software powers half of all new mobile games, lands $400 million from Silver Lake [Електронний ресурс] // Tech Crunch. – 2017. – Режим доступу до ресурсу: https://techcrunch.com/2017/05/23/unity-whose-software-powers-half-of-all-new-mobile-games-lands-400-million-from-silver-lake/

19. Unity’s asset store boss has big plans to fight Epic’s Unreal [Електронний ресурс] // Venture Beat. – 2018. – Режим доступу до ресурсу: https://venturebeat.com/2018/07/18/unitys-asset-store-boss-has-big-plans-to-fight-epics-unreal/

20. Unreal Engine [Електронний ресурс] // Wikipedia. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Unreal_Engine

21. Best Game Engines of 2020 [Електронний ресурс] // GameDev Academy. – 2020. – Режим доступу до ресурсу: https://gamedevacademy.org/best-game-engines/#Phaser

22. Game Maker Studio 2 [Електронний ресурс] // Yoyogames. – 2020. – Режим доступу до ресурсу: https://www.yoyogames.com/gamemaker/features

23. Random walk [Електронний ресурс] // Wikipedia. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Random_walk

24. IDEF0 as a Project Management Tool in the Simulation Modeling and Analysis Process in Emergency Evacuation from Hospital Facility: A Case Study [Електронний ресурс] // docplayer. – 2014. – Режим доступу до ресурсу: https://docplayer.net/9837160-Idef0-as-a-project-management-tool-in-the-simulation-modeling-and-analysis-process-in-emergency-evacuation-from-hospital-facility-a-case-study.html

25. UML Use Case Diagram Tutorial [Електронний ресурс] // lucidchart. – 2020. – Режим доступу до ресурсу: https://www.lucidchart.com/pages/uml-use-case-diagram

26. How to Make an RPG: Stats [Електронний ресурс] // HowtoMakeanRPG. – 2017. – Режим доступу до ресурсу: http://howtomakeanrpg.com/a/how-to-make-an-rpg-stats.html

27. Shader Graph in Unity for Beginners [Електронний ресурс] // raywenderlich. – 2019. – Режим доступу до ресурсу: https://www.raywenderlich.com/3744978-shader-graph-in-unity-for-beginners

28. Unity - The Particle System [Електронний ресурс] // tutorialspoint. – 2020. – Режим доступу до ресурсу: https://www.tutorialspoint.com/unity/unity_the_particle_system.htm

ДОДАТОК А

Планування робіт

1. Ідентифікація мети реалізації ігрового додатку

Мета проекту є орієнтиром при прийнятті рішень на протязі всього його життєвого циклу. Для її конкретизації був використаний метод SMART. Даний метод визначає: конкретність, вимірюваність, досяжність, реалістичність та обмеженість у часі.

Таблиця А.1 Деталізація мети методом SMART

S (Конкретність)	Розробити мобільний ігровий додаток в жанрі екшн roguelike.
M (Вимірюваність)	Залучити якомога більше гравців користуватися ігровим додатком.
A (Досяжність)	Для досягнення мети в наявності є всі необхідні навички та ресурси.
R (Доцільність)	На сьогоднішній день, ринок відеоігр зростає та приносить високі прибутки.
T (Обмеженість в часі)	Проект буде виконано вчасно, що підтверджується календарним планом.

2. Планування змісту структури робіт IT-проекту

Структура декомпозиції робіт (WBS) – це метод, який розбиває проект на ієрархію завдань, підзадач та результатів. Це дуже корисний інструмент, який визначає детальну оцінку часу або витрат та забезпечує контролю над роботою.

По суті, використання WBS структури дозволяє нам поглянути на наш проект зверху вниз та розбити його на завдання та підзадачі, які приведуть нас до його завершення. Це простий спосіб організації та розуміння обсягу проекту в менших, керованих компонентах =. WBS структура представлена на рисунку А.1.

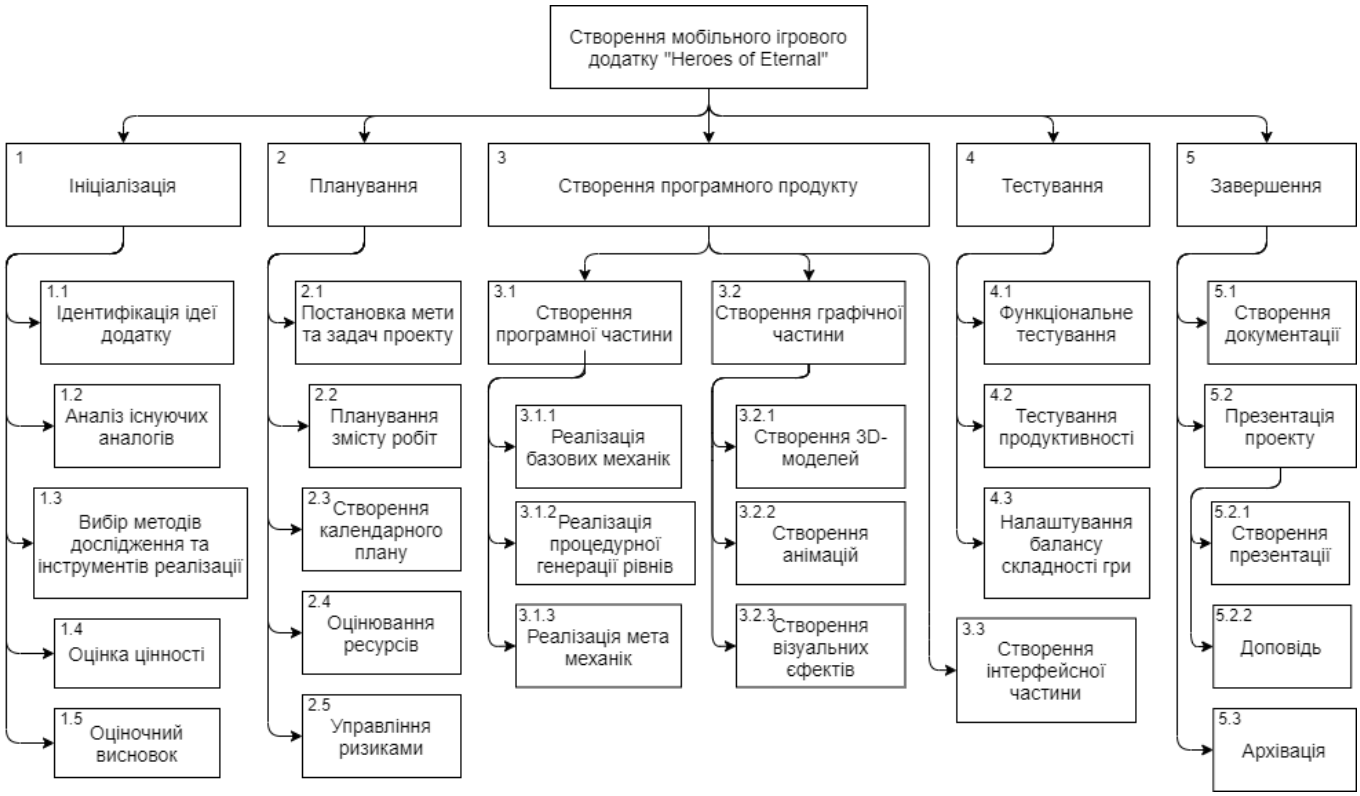

Рисунок А.1 – WBS структура проекту

Організаційна структура проекту (OBS) – використовується для відображення учасників проекту та відповідальних осіб, які залучені до проекту. Елементами даної структури можуть бути як цілі служби, так і окремі робітники, а взаємозв’язки між ними бути вертикальними та горизонтальними, та носити функціональний характер. OBS структура будується на основі WBS. Коли обов'язки по проекту визначені а роботи призначені, OBS та WBS забезпечують можливість для потужної аналітики продуктивності проекту та його робочої сили на дуже високому рівні. OBS структура представлена на рисунку А.2.

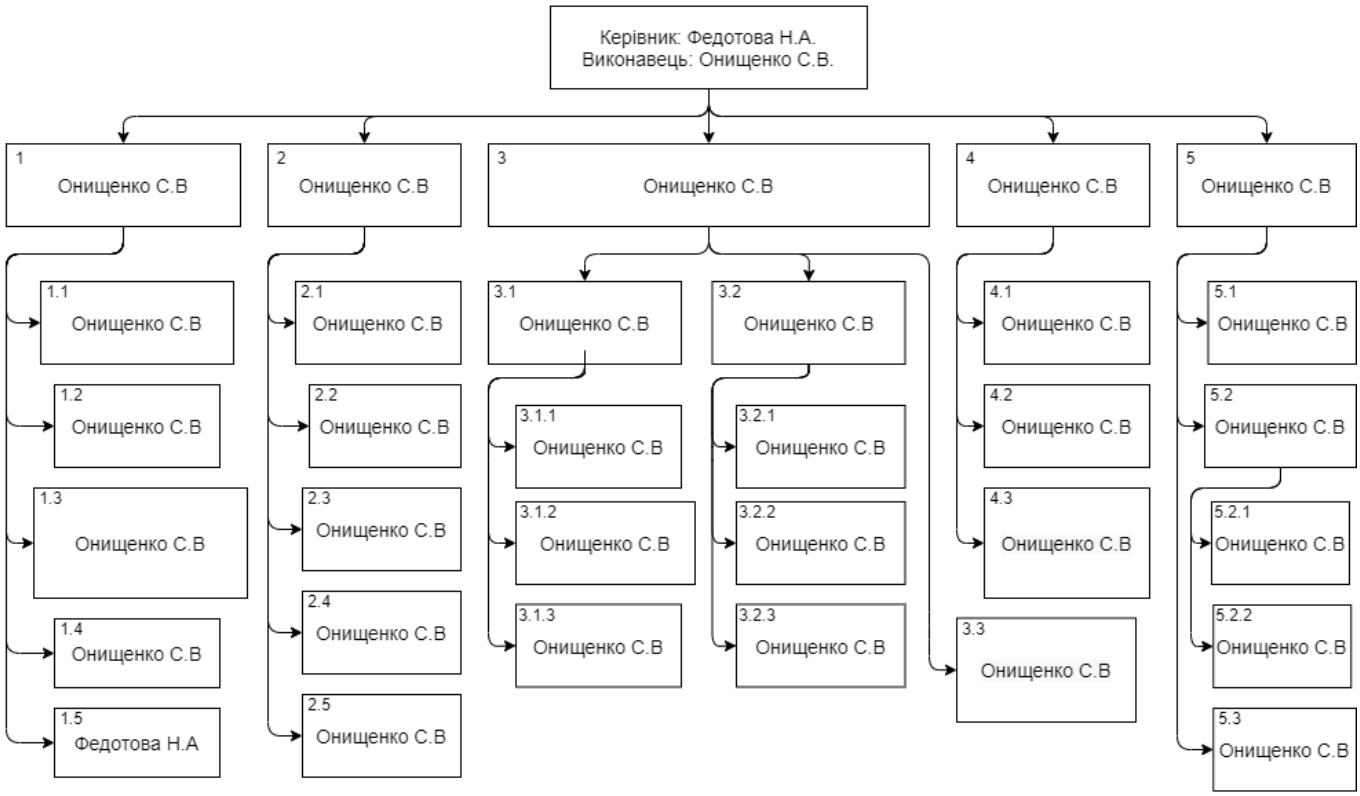

Рисунок А.2 – OBS структура

На основі WBS та OBS структур побудуємо матрицю відповідальності. Вона являє собою структурну схему, на якій візуально чітко видно, хто що повинен робити і які завдання та обов'язки кожного з членів команди. Матриця відповідальності представлена таблицею А.2.

Таблиця А.2 – Матриця відповідальності

	Виконавець	Керівник
	Онищенко С.В.	Федотова Н.А.
1. Ініціалізація	+	
1.1 Ідентифікація ідеї додатку	+	
1.2 Аналіз існуючих аналогів	+	
1.3 Вибір методів дослідження та інструментів реалізації	+	
1.4 Оцінка цінності	+	
1.5 Оціночний висновок		+

2. Планування	+	
2.1 Постановка мети та задач проекту	+	
2.2 Планування змісту робіт	+	
2.3 Створення календарного плану	+	
2.4 Оцінювання ресурсів	+	
2.5 Управління ризиками	+	
3. Створення програмного продукту	+	
3.1 Створення програмної частини	+	
3.1.1 Реалізація базових механік	+	
3.1.2 Реалізація процедурної генерації рівнів	+	
3.1.3 Реалізація мета механік	+	
3.2 Створення графічної частини	+	
3.2.1 Створення 3D-моделей	+	
3.2.2 Створення анімацій	+	
3.2.3 Створення візуальних ефектів	+	
3.3 Створення інтерфейсної частини	+	
4. Тестування	+	
4.1 Функціональне тестування	+	
4.2 Тестування продуктивності	+	
4.3 Налаштування балансу складності гри	+	
5. Завершення	+	
5.1 Створення документації	+	
5.2 Презентація проекту	+	
5.2.1 Створення презентації	+	
5.2.2 Доповідь	+	
5.3 Архівація	+	

3. Побудова календарного графіку виконання ІТ - проекту

Для побудови календарного графіку розробки проекту була використана Діаграма Ганта. Це інструмент, що допомагає у плануванні проектів будь-яких розмірів. Головна особливість діаграми Ганта полягає в тому, що вона є наочною. У лівій частині діаграми знаходиться список всіх завдань, необхідних для розробки проекту, а вгорі – шкала часу. Кожна задача представлена смугою, а її довжина представляє тривалість завдання. У той же час кожен бар представлений назвою дії, яка повинна бути завершена, разом з будь-якою відповідною інформацією, яка допоможе довести його до завершення [5].

Діаграма Ганта представлена на рисунку 1.3, для її побудови використовувалась програма MS Office Project 2020.

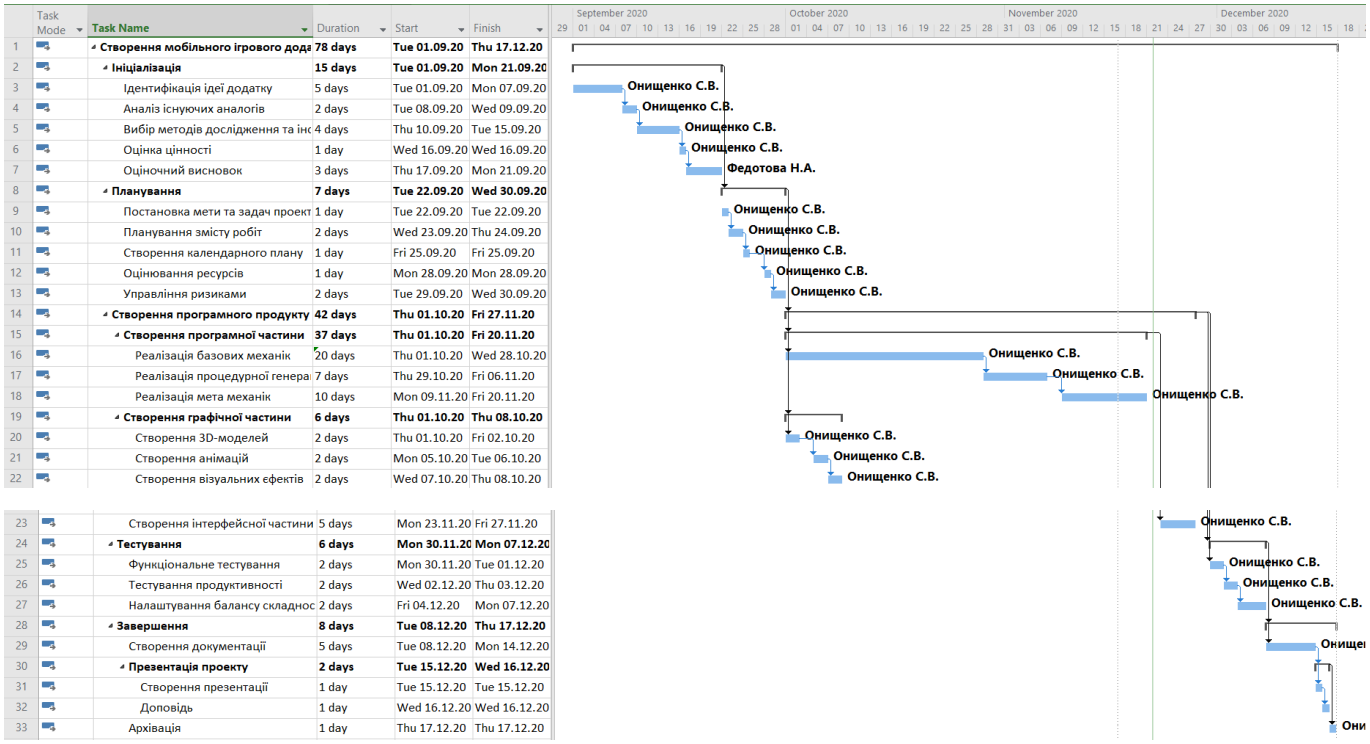

Рисунок А.3 – Діаграма Ганта

4. Планування ризиків проекту

Проведемо якісний та кількісний аналіз ризиків розробки мобільного ігрового додатку. Якісний аналіз надає пріоритет виявленим ризикам за допомогою заздалегідь визначеної рейтингової шкали. Ризики будуть оцінені в залежності від вірогідності їх виникнення та впливу на цілі проекту у разі виникнення. Кількісний аналіз ризику – це подальший аналіз найбільш пріоритетних ризиків [6].

До основних ризиків розробки мобільного ігрового додатку «Heroes of Eternal» можна віднести:

- брак досвіду роботи з Unity;
- слабке розуміння предметної області;
- погане планування та управління розробкою;
- людський фактор;
- зростання вимог під час розробки;
- проблеми з обладнанням;
- різке скорочення строків здачі роботи.

Таблиця А.3 – Ймовірність виникнення і величина ризику

№	Ризики	Виникнення	Втрати
Р1	Брак досвіду роботи з Unity	3	1
Р2	Слабке розуміння предметної області	2	2
Р3	Погане планування та управління розробкою	4	3
Р4	Людський фактор	3	2
Р5	Зростання вимог під час розробки	1	4
Р6	Проблеми з обладнанням	2	5
Р7	Різке скорочення строків здачі роботи	1	4

Таблиця А.4 – Матриця «Ймовірність – Втрати»

Ймовірність							
	-	5	5	10	15	20	25
	Р3	4	4	8	12	16	20
	Р1 Р4	3	3	6	9	12	15
	Р2 Р6	2	2	4	6	8	10
	Р5 Р7	1	1	2	3	4	5
			1	2	3	4	5
			Р1	Р2 Р4	Р3	Р5 Р7	Р6
			Втрати				

Таблиця А.5 – Втрати при виникненні ризиків

Втрати						
	5					
	4			Р3		
	3	Р1	Р4			
	2		Р2			Р6
	1				Р5 Р7	
		1	2	3	4	5
	Ймовірність					

Виходячи з цього, було визначено чотири критичних ризики, такі як:

- Р3: погане планування та управління розробкою;
- Р6: проблеми з обладнанням.

Таблиця А.6 – Варіанти запобігання та реакції на виникнення ризиків

Ризик	Наслідки та способи уникнення
Погане планування та управління розробкою	Своєчасне визначення вимог проекту з керівником
Проблеми з обладнанням	Розподіл свого часу та ресурсів заздалегідь

ДОДАТОК Б

Скріншоти ігрового процесу

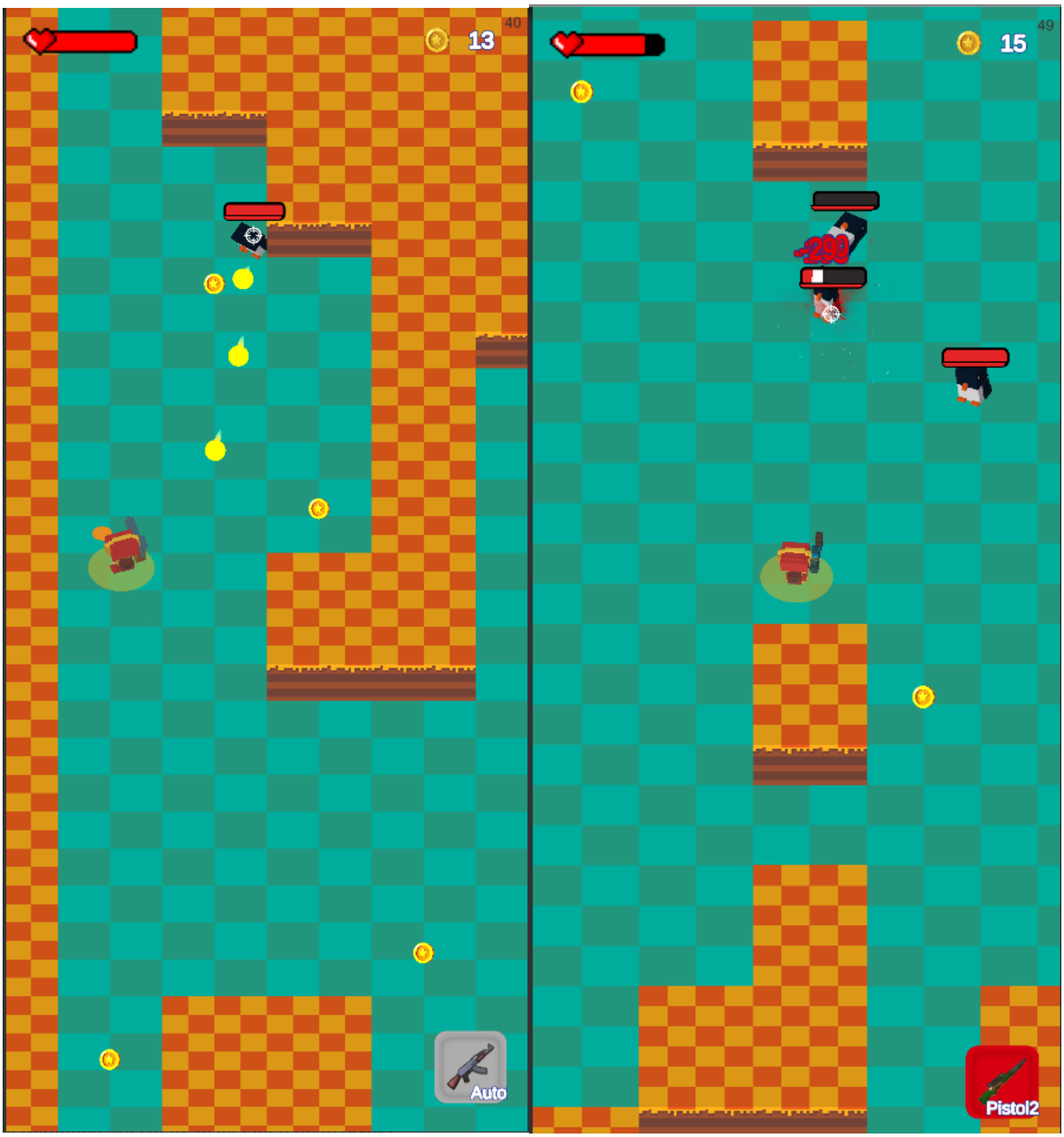

Рисунок Б.1 – Скріншоти ігрового процесу

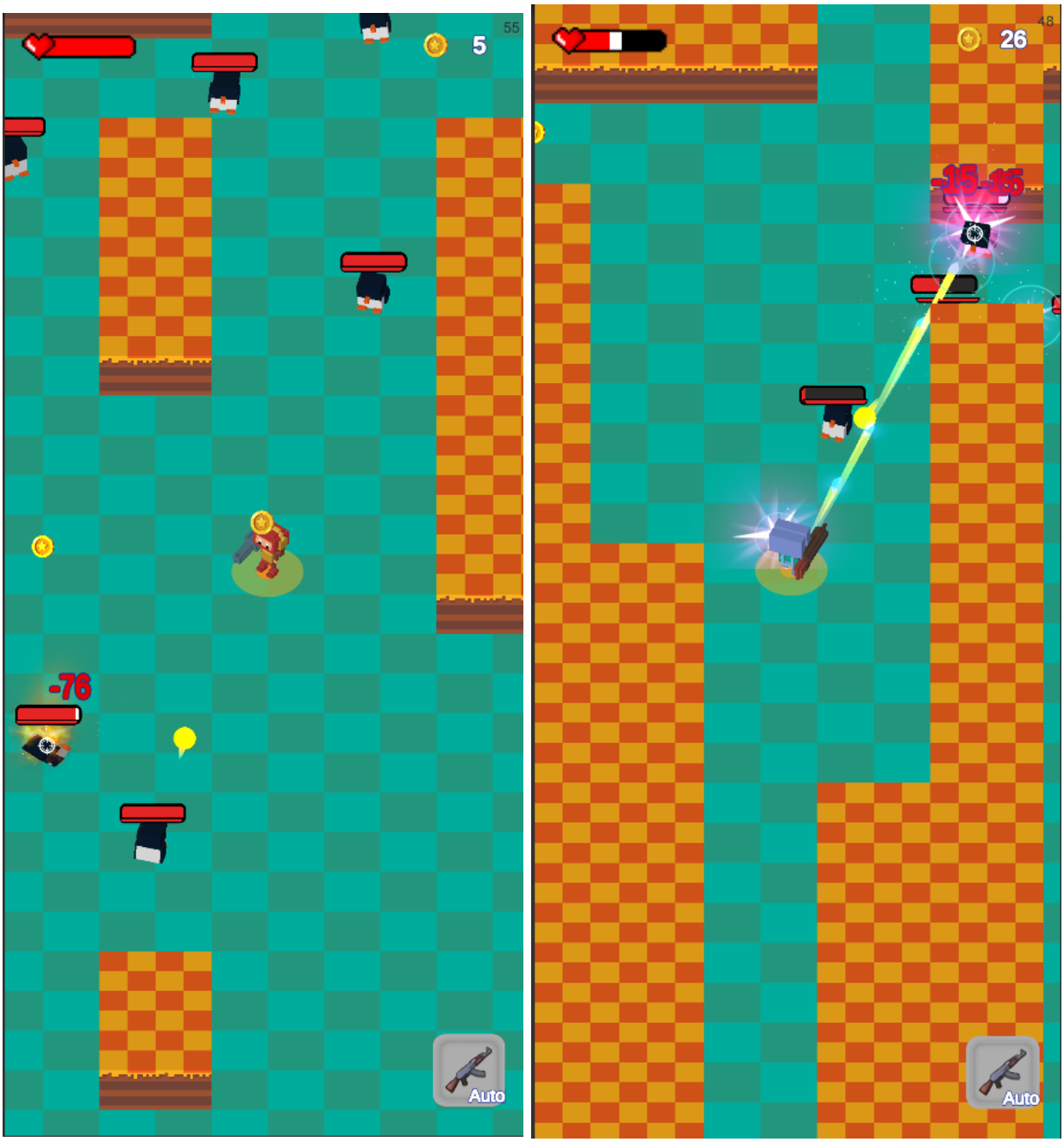

Рисунок Б.2 – Скріншоти ігрового процесу

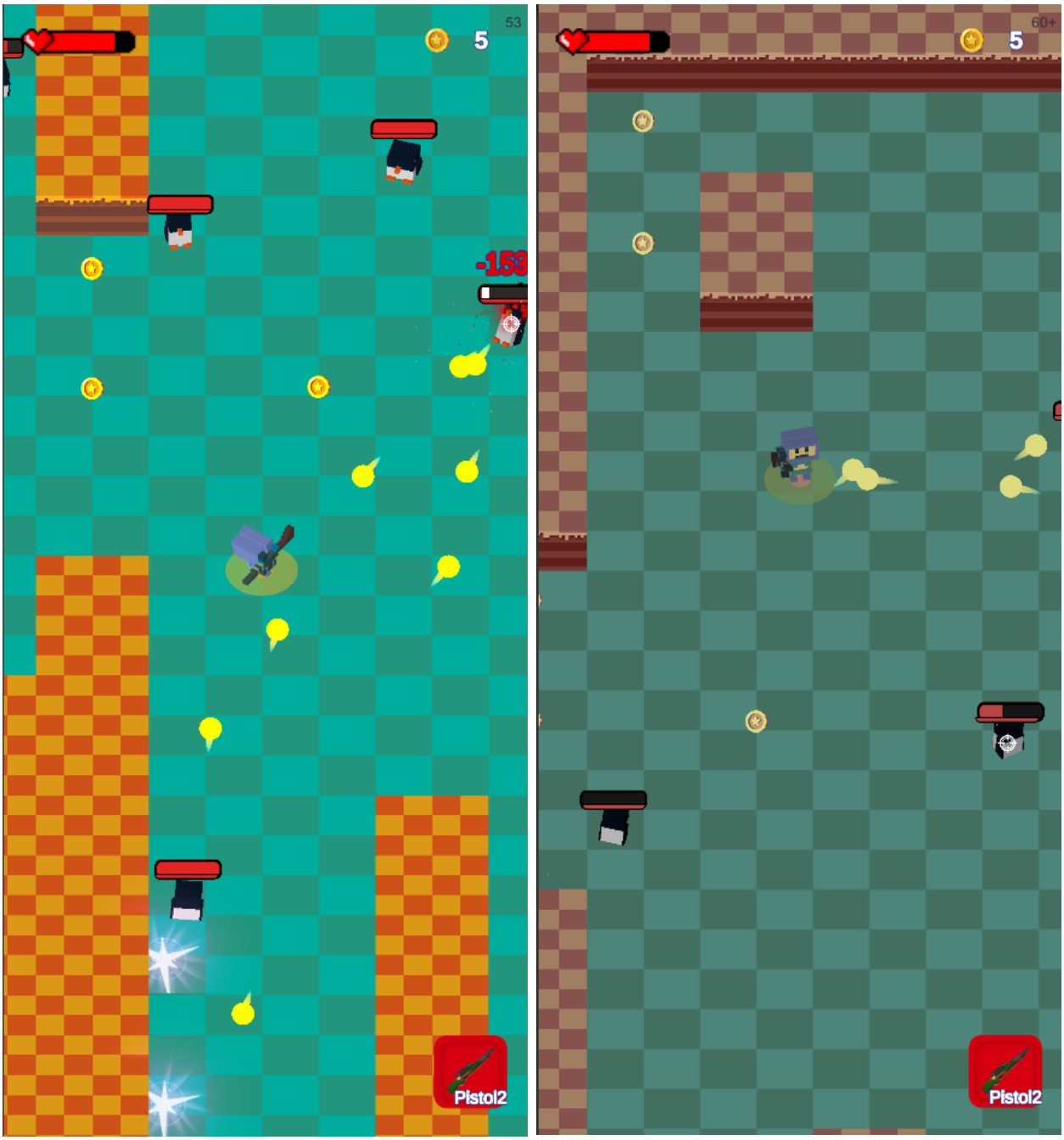

Рисунок Б.3 – Скріншоти ігрового процесу

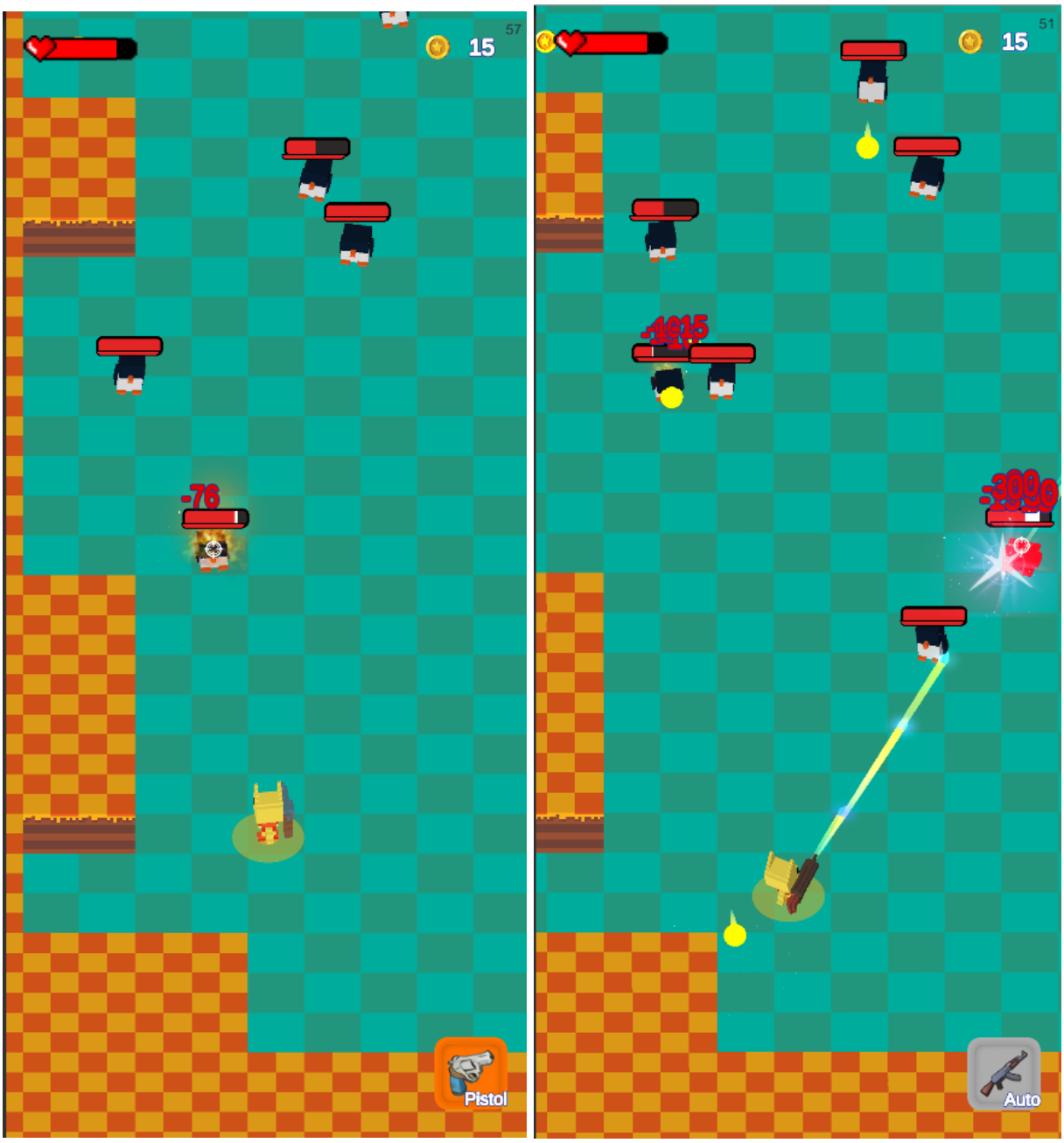

Рисунок Б.4 – Скріншоти ігрового процесу

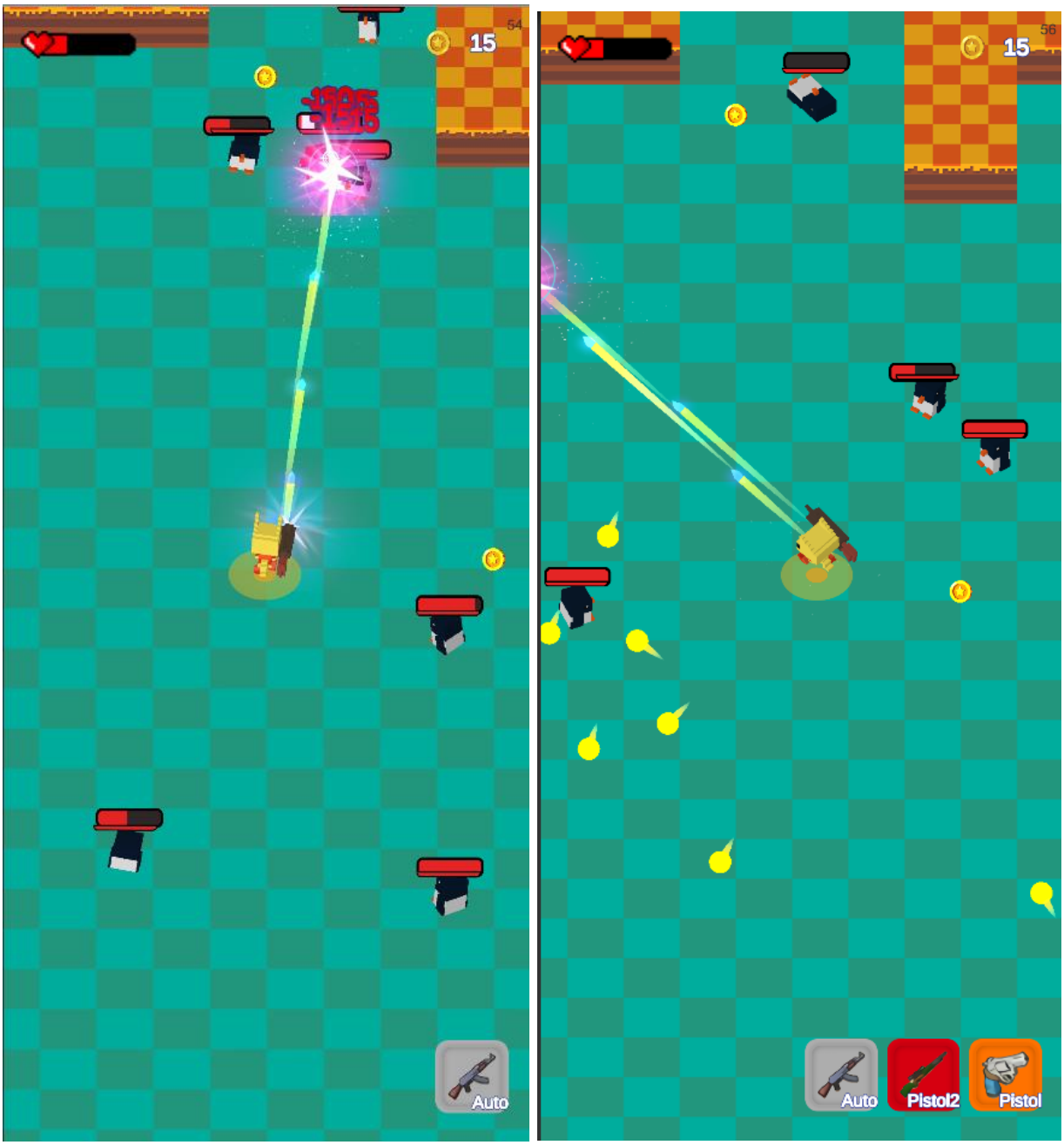

Рисунок Б.5 – Скріншоти ігрового процесу