НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
_____комп'ютерних наук_____
(назва кафедри)

_____ _____Голуб Б.Л._____
(підпис) (ПІБ)

"___" червня 2025р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Розробка 2D гри класу платформер за допомогою Unity»

Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми
____д.ек.н., професор_____ _____ __Руденський Р.А_____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
_____к.т.н. доцент_____ _____ __Бородкіна І.Л._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _Щербань Станіслав Олексійович_____
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОВИКОРИСТАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)
_____/ Голуб Б.Л. доцент, к.т.н./
(підпис)
"___" _____2025р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту

Щербаню Станіславу Олексійовичу

Спеціальність 122 – «Комп'ютерні науки»

1. Тема бакалаврської кваліфікаційної роботи «Розробка 2D гри класу платформер за допомогою Unity» затверджена наказом ректора НУБіП України №2246с від "16" грудня 2024 р.
2. Термін подання завершеної роботи на кафедру 2025 . 06 . ____ (рік, місяць, число)
3. Вихідні дані для роботи: опис програмного забезпечення.
4. Перелік питань, що розглядаються:
    1. Провести дослідження предметної області у розрізі ігрової індустрії.
    2. Обрати та опанувати середовище розробки, на основі якого будуть реалізовані запропоновані рішення.
    3. Засвоїти принципи роботи необхідних шаблонів проекування та реалізувати їх у власному проекті.
    4. Оцінити зручність та ефективність реалізованого підходу створення гри з позиції розробника.
    5. Розробити рекомендації щодо розвитку та масштабування підходу.

Керівник бакалаврської кваліфікаційної роботи_____/Бородкіна І.Л. /
(підпис) (прізвище та ініціали)
Завдання прийняв до виконання:_____/ Щербань С.О. /__
(підпис) (прізвище та ініціали)
Дата подання завдання 2025.01.05
Рік місяць число

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

LMB (Left Mouse Button) – ліва кнопка миші.

RMB (Right Mouse Button) - права кнопка миші.

AAA-проект (Triple-A) – ігри що створюються великими студіями зі залученням при розробці великого бюджету.

Кастомний (від англ. custom) – Використовується для опису речей, що були налаштовані спеціально під виконання конеретної задачи.

Скрипт – програмний код, що автоматизує виконання поставлених перед ним завдань.

UI (User Interface) - інтерфейс користувача, тобто спосіб взаємодії людини з програмамою.

Колайдер – фізичний компонент, що визначає форму об’єкта для зіткнення. Використовується у ігровій фізиці для перешкоджання перетинання фізичних тіл різних об’єктів.

Хітбокс – невидима область, яка визначає, чи об'єкт був атакований.

Колізія – механізм, який визначає зіткнення між об’єктами.

Рендер - процес виведення зображення на екран. Для Unity це означає перетворення 2D сцени у фінальне зображення, яке бачить користувач.

Спрайт - об’єкт, який представляє окреме зображення або фрагмент із текстури.

Дженерик - це шаблони типів у C#, вони дозволяють створювати класи або методи, що працюють з різними типами даних. Головною перевагою є написання гнучкого і повторно використовуваного коду.

Комбо – послідовність атак гравця за короткий проміжок часу.

Шар – спосіб групування об’єктів у різних цілях: для обробки порядку виння спрайтів, обробки фізики об’єктів.

ВСТУП

За останнє десятиліття відеоігри втратили статус виключно розважального продукту ставши важливим соціокультурним і навіть освітнім явищем. Так ігри на кшталт *Civilization* та *Assassin's Creed* допомагають учням глибше зануритися в історичні події та розвивати критичне мислення. Вони дозволяють студентам взаємодіяти з історичними контекстами, що сприяє більш глибокому розумінню матеріалу [1]. Їх також можна використовувати як інструмент соціалізації. Дослідження показують, що ігри, що вимагають співпраці та взаємодії між гравцями, можуть покращувати комунікативні навички та сприяти соціалізації. Однак надмірне захоплення підвищує ризик зворотнього ефекту, знижуючи рівень соціального розвитку, особливо у дітей [2]. Щодо професійної підготовки існує програма Command: Professional Edition, розроблена компанією Slitherine Software, що використовується Пентагоном, Військово-повітряними силами США та Британським стратегічним командуванням у військових цілях. Програма дозволяє військовим аналітикам та стратегам відпрацьовувати різні сценарії та покращувати прийняття рішень [3].

Актуальність теми полягає в тому що розробники початківці часто стикаються з труднощами при реалізації базових механік таких як: пересування, вірне зчитування та обробка колізій, узгодження патернів поведінки ворогів, взаємодія з об'єктами. Все це ускладнює створення повноцінної гри. Відсутність структурованих реалізацій та добре організованої архітектури є додатковим бар'єром. В результаті чого розробник витрачає неймовірну кількість часу на усунення багів та конфліктів спричинених спробою впровадження нової механіки. Двовимірний напрям розробки обрано через те що на відміну від тривимірного дає змогу сконцентруватися на розробці та реалізації шаблонів проектування, анімацій, фізики, поведінкової логіки, частково уникаючи проблему кропіткого пошуку або створення графічних ресурсів, не втрачаючи при цьому свій оригінальний стиль. Внаслідок чого можна сформувати основу

для створення сучасних, підтримуваних ігрових проектів, здатних відповідати як вимогам ринку, так і професійним стандартам розробки.

Об’єктом дослідження для цієї кваліфікаційної роботи є двовимірні ігри жанру платформер.

Предмет дослідження представляє з себе конкретну реалізацію програмного продукту в описаного жанру з використанням актуальних технологій та підходів.

Метою, роботи є створення підходу щодо розробки двовимірних платформерів для вирішення проблем, пов’язаних з масштабованістю, керованістю та повторним використанням коду шляхом створення 2D-платформера з використанням актуальних щодо архітектури програмного коду, зокрема реалізації шаблону проектування машини кінцевих автоматів (Finite State Machine) для поведінки гравця та неігрових персонажів. А також розробка гнучкої підсистеми модульного конструювання озброєння персонажа гравця (Multy Weapon System) для формування комплексних бойових механік.

Для досягнення представленої мети необхідно виконати такі завдання:

- Провести дослідження предметної області, визначити особливості ігрової індустрії.
- Обрати та опанувати середовище розробки на основі якого будуть реалізовані запропоновані рішення.
- Засвоїти принципи роботи необхідних шаблонів проекування та реалізувати їх у власному проекті.
- Оцінити зручність та ефективність реалізованого підходу створення гри з позиції розробника.
- Розробити рекомендації щодо розвитку та масштабування створеної гри.

Для розробки програмного додатку було використано наступні методи та технології**:**

- мова програмування C#;

- середовище розробки Unity;
- патерн Finite State Machine для організації логіки гравця та противників;
- компонентний підхід до побудови шаблонів озброєння Multy Weapon System;
- патерн Objective Pool для повторного використання створених об’єктів

Обсяг пояснювальної записки до дипломної роботи становить 70 сторінок, 13 використаних джерел літератури, три таблиці та чотири додатки. Записка складається з таких основних розділів:

- СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.
- МОДЕЛЮВАННЯ СИСТЕМИ.
- РОЗРОБКА СИСТЕМИ.
- РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ.

1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Введення в ігрову індустрію

З моменту появи перших відеоігор у 1970-х роках ігрова індустрія пройшла розвиток від низькобюджетних проектів ентузіастів до багатомільярдної екосистеми, що охоплює розробку, публікацію, розповсюдження та монетизацію інтерактивного контенту. Також по своїй суті ігрова індустрія тісно переплітається з різноманітними сферами та використовує їх напрацювання тим самим надаючи сотні тисяч робочих місць.

До таких галузей можна віднести:

- Дизайн. Широко використовуються всі його прояви від концепт артів персонажів та локацій, до навіть розвитку в нові напрями такі як геймдизайн. Головною задачею геймдизайну є проектування і наповнення продукту інтуїтивно зрозумілими механіками та завданнями. Узгоджуючи тим самим очікуваний гравцем результат та реально отриманий.
- Література. Переважна більшість одно користувацьких проектів наймають в команду сценаристів задача яких написати хороший сюжет та прописаний глибокий лор світу. Внаслідок чого вдається максимально залучити користувача в ігровий процес.
- Кіноіндустрія. Використовується для створення якісних катсцен. Переймаються техніки професійного позиціонування камери та об’єктів в кадрі.
- 2D та 3D графіка. Свого роду будівельники для ігрових світів. Будь яка модель розміщена на ігровій сцені була створена графічними художниками.

- Музика. Для створення звукового супроводу музичні студії записують унікальні та автентичні композиції, що позитивно впливає на досягнення потрібної атмосфери.
- Маркетинг. Використовуються різні способи монетизації. Як класичні такі як купівля особистої копії або підписки на певний термін, так і більш незвичні у вигляді мікротранзакцій, тобто покупки цифрового контенту безпосередньо всередені гри.

Згідно з аналітичним звітом компанії *Newzoo*, у 2023 році глобальний дохід від ігрової індустрії становив понад 184,4 млрд доларів США, важливо зазначити що понад половину з цієї суми забезпечив мобільний сектор. Сегменти консолей та ПК також продовжують утримувати стабільні позиції, маючи при цьому не лише фінансову стабільність, а й зростання аудиторії [4].

До ключової рушійної сили в розвитку ігрового ринку можна віднести стрімке вдосконалення рушіїв на основі яких створюють відеоігори. Найбільш популярними є Unity, Godot та Unreal Engine. Серед представлених середовищ Unity активно використовується як для розробки двовимірних, так і тривимірних проектів. Рушій підтримує експорт на більш ніж 25 платформ таких як Windows Android, iOS, , WebGL, Xbox, PlayStation тощо. Завдяки своїй відкритій архітектурі дозволяє інтеграцію сторонніх модулів. Все це робить його універсальним інструментом для студій різного масштабу.

Важливою особливістю відеоігор є тісна інтеграція в них сучасних технологій. Так в наслідок впровадження методу процедурної генерації контенту, що дозволяє автоматично створювати великі й варіативні ігрові світи, *No Man's Sky* отримала понад 18 квінтильйонів унікальних планет. Це дає змогу суттєво розширити користувацький досвід і забезпечивши його неповторність для кожного.

Іншою помітною технологією є використання штучного інтелекту. Його застосування можна побачити на існуючих прикладах адаптивної поведінки ворогів, згенерованих квестів, автоматично анімованих рухів персонажів, навіть

синтезованих звуків. Наприклад, в *The Last of Us Part II* штучний інтелект супротивників вміє взаємодіяти з навколишнім середовищем, координувати дії між собою та реагувати на мінливу тактику гравця.

Розвиток соціального аспекту є не менш важливим. Основні онлайн-платформи, такі як *Steam*, *Epic Games Store*, Xbox, *PlayStation Network*, забезпечують функції не лише купівлі та встановлення ігор, але й спілкування між гравцями, участі в спільних подіях та оновлення контенту. Різноманітні спільноти, платформи стрімінгу (*Twitch, YouTube*) та змагання з кіберспорту створюють навколо ігор справжні екосистеми, що виходять за межі так званої «гри заради гри».

Таким чином, сучасна ігрова індустрія – являє собою складний багатофункціональний комплекс, що поєднує в собі мистецтво, інновації, науку та бізнес. Індустрія має величезний вплив на культуру, освіту, економіку і є потужним простором для реалізації креативних ідей.

1.2 Постановка завдання

1. Створення контролеру персонажа гравця з реалізованими механіками:
    - Пересування по горизонталі
    - Стрибок, мультистрибок
    - Зістрибування з платформи проходячи через неї
    - Зісковзування зі стін
    - Стрибок від стіни
    - Зависання, підйом на виступ
    - Пересування навприсядки
    - Ривок у 8 різних напрямків
    - Реакція гравця на отримання шкоди
    - Інвентар для екіпіровки
    - Підбирання да заміна зброї

2. Гравець повинен мати та використовувати коректно всі необхідні анімації.
3. Створення модульного контролеру для керування поведінкою ворогів.
4. Створення модульного рішення для формування озброєння для гравця. На основі якого скомпонувати мінімум два типи обладнання серед яких зброя ближнього ураження у вигляді меча та дальнього - лука.
5. Створити обробник для польоту снарядів, що використовують вороги та гравець.
6. У грі повинні бути реалізовані механіки:
   - Побудова рівнів через тайлмапи
   - Збереження в форматі .json
   - Завантаження на основі збереженого рівня
   - Завершення рівня
7. Програма повинна формувати звітну інформацію в форматі .xlsx на основі даних зі збережень та пройдених рівнів.
8. Камера сцени повинна переміщатися за гравцем та надавати йому зручний огляд на те що знаходиться перед ним.
9. Гра повинна мати стартовий екран та можливість переходу на сцену рівня та назад.

1.3 Огляд існуючих рішень

Для створення програмного продукту та виконання запропонованих завдань потрібно спочатку проаналізувати існуючі рішення. Це дозволить більш чітко уявити орієнтовний вигляд якому повинна частково відповідати реалізована система. Мій проект надихається двома успішними іграми – це Hollow Knight та Dead Cells. Обидві гри користуються авторитетом серед поціновувачів двовимірних світів. Цікавим є те що хоч обидві гри є 2D

платформерами проте концептуально вони демонструють різні підходи до побудови ігрового процесу, взаємодії світу з гравцем та дизайну рівнів.

Hollow Knight – двовимірна гра платформер в жанрі метроїдванія [5]. Основною ідеєю жанру є концентрація на дослідженні світу та атмосфері. Передбачається дослідження раніше недоступних частин пройдених локацій доступ до яких відкривається з отриманням нових здібностей. Ця гра є ідеальним прикладом реалізації контролеру гравця та роботи з камерою.

Контролер гравця в Hollow Knight має такі особливості:

- Стрибок за довжиною та висотою має часткову залежність до часу утримання клавіші. На цій механіці засноване як проходження платформенних участків так і битв з унікальними ворогами, так званих «босів» рівнів.
- Тіло героя інертне, тобто після завершення введення руху персонаж не зупиняється моментально, натомість впродовж дуже короткого проміжок часу продовжує свій рух, формуючи тим самим більш реалістичні та інтуїтивно логічні відчуття від керування.
- Має додаткові навички для пересування, такі як додатковий стрибок та ривок що відкриваються не одразу, а по проходженню відповідних сюжетних подій. Це дозволяє новачку не бути перевантаженим всіма можливостями контролеру, а освоювати та використовувати їх поступово.
- Збалансований стрибок від стіни. Розробник передбачив що гравець схоче спробувати пробратися в недоступні на певному етапі гри локації шляхом стрибку від стіни та затисненням клавіші руху в сторону стіни для повернення персонажа назад на стіну але уже будучи вище від стартової позиції. Тому реалізована така логіка стрибку від стіни щоб при спробі скористатися цією хитрістю гравець в найкращому випадку повертався на стартову позицію.

Доступ до цієї техніки відкривається з розблокуванням вмінь пересування описаних вище.

- Використовує механіку «койотівого часу». Поширений прийом завдяки якому можна відштовхнутися від землі навіть якщо гравець щойно втратив поверхність під ногами. Застосовується як інструмент дружнього досвіду, так як завдяки нього у гравця більше часу на прийняття рішень в ситуації постійно мінливих умов. Цікавим є те що назва прийому походить з мультфільму «Хитрий койот і Дорожній бігун", де персонаж Койот замість того щоб впасти з урвища міг ще деякий час бігти по повітрю.
- Має механіку бойового пересування. При атаці по повітрю персонаж пересувається вперед, якщо атака досягає противника то подальша взаємодія залежить від його розмірів. Так якщо противник великий персонажа гравця буде відкидувати трохи назад, симулюючи тим самим віддачу. Унікальною фішкою бойової системи є можливість атаки в повітрі в напрямку під собою. У разі якщо дія уразить ворога або пастку то персонажа підкине вверх. Цю процедуру можна повторювати безліч разів. Ця проста механіка породжує безліч ігрових ситуацій та викликів. Приклад виконання цієї техніки можна побачити на рис.1

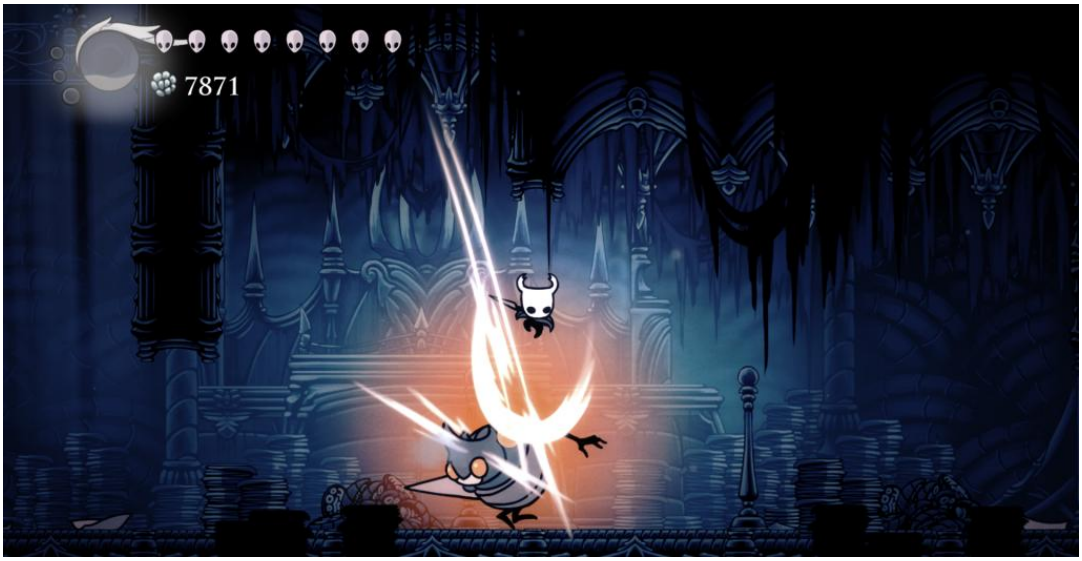


Рис1. Демонстрація геймплею та атаки під собою

Dead Cells - двовимірна гра платформер в жанрі рогалик [7]. Основною ідеєю жанру є процедурна генерація рівнів, концепт «одне проходження одна спроба», швидкий темп, концентрація на бойовій системі.

Важливо зазначити що Dead Cells також має чудовий контролер персонажа що частково реалізовує механіки попередньо розібраного проекту. Проте в контексті аналізу існуючих рішень він має другорядне значення адже представлений проект розглядається з позиції прикладу реалізації бойової системи.

Бойова система має наступні особливості:

- Розблокування вмінь також відбувається поступово. Але через те що ігровий процес має формат окремих невзаємопов'язаних сесій на згенерованих локаціях, а не у заздалегідь підготованому відкритому світі, то на освоєння всіх здібностей витрачається всього декілька забігів, після чого вони стають доступними завжди, в незалежності від етапу сесії.
- Розміщення зброї. Зброя розміщена на рівні та з'являється в випадкових місцях при його генерації.
- Інвентар складається з двох комірки для основної зброї та двох для гаджетів. Використання кожної комірки прив'язане до певної кнопки. Геймплей та інтерфейс гри можна розглянути на рис.2
- Збирання та заміна екіпіровки. Щоб підібрати зброю потрібно підійти до предмету та натиснути кнопку взаємодії, після чого обладнання екіпірується у вільну комірку відповідного типу. Якщо вільних комірок нема, то з'являється вікно вибору в якому потрібно визначитися яку зброю залишити, а від якої відмовитись. Після прийняття рішення зайвий предмет не пропадає, а викладається з інвентаря, залишаючись на тому ж місті, стаючи доступним для повторного підбору.

- Кожний предмет має свою унікальну здібність та модифікатор. Особливості та умови для їх спрацювання описані у відповідному вікні яке активується автоматично при перебуванні поряд з об’єктом або при підборі у селекторі екіпіровки Для більш чіткого розуміння описаного на рис.3 наведено приклад.
- Після проходження локації гравець має змогу отримати додаткову винагороду за умови виконання побічних випробовувань таких як проходження локації за певний проміжок часу або за факт відсутності отриманої шкоди. Це стимулює гравця відточувати навички володіння та розуміння бойових механік, даючи йому ціль. Завдяки цьому подовжується середню тривалість залучення клієнта в проекті.

Рис.2 Кадр з геймплею Deadcells

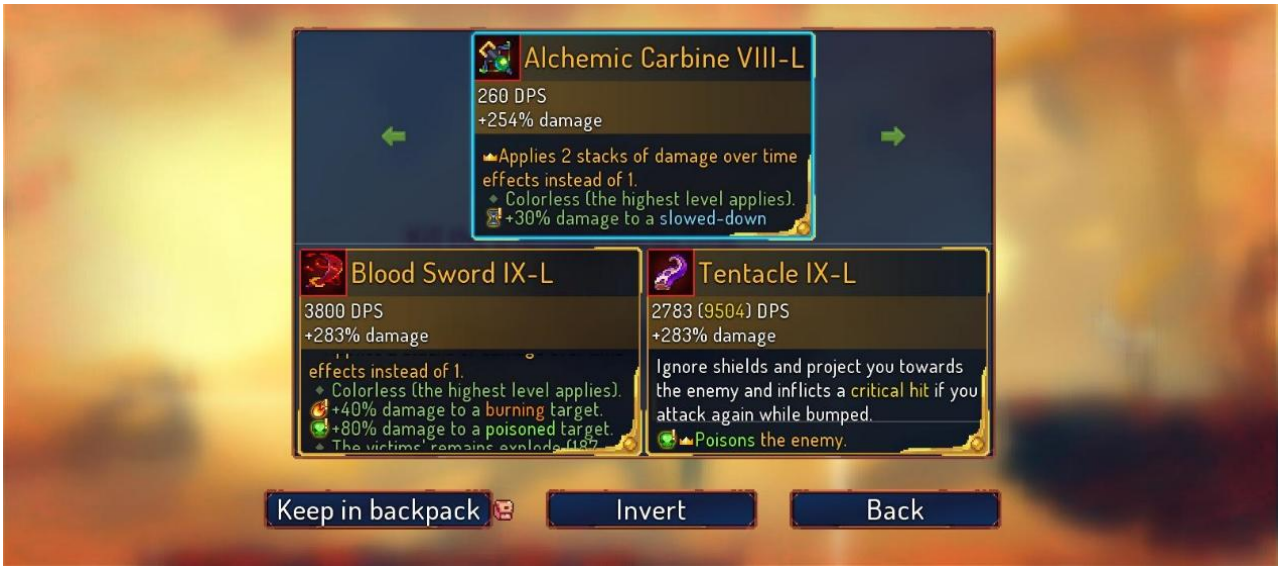


Рис.3 Селектор вибору зброї(у грі відсутня українська локалізація)

Також важливо провести порівняння цих проектів. На табл.1, табл.2, табл.3 опрацьовано протиставлення запропонованих відеоігор у трьох основних напрямах: загальна інформація, функціональність, переваги та недоліки.

Таблиця 1.1

Загальна інформація

Назва гри	Розробник	Рік випуску	Жанр	Використані технології
Hollow Knight	Team Cherry	2017	2D Екшен метроїдванія	Unity
Dead Cells	Motion Twin	2018	2D Екшен рогалик	Haxe и Heaps

Таблиця 1.2

Порівняння за функціональністю

Характеристика	Hollow Knight	Dead Cells
Геймплей	Дослідження світу, система пересування, боси	Динамічні бої, процедурна генерація, перманентна смерть
Графіка	Стилізована 2D-анімація ручної роботи	Піксель-арт з використанням великої кількості ефектів

Таблиця 1.2 (закінчення)

Аудіо	Атмосферна музика, озвученне середовище	Енергійний саундтрек, зосередження на ефектах
Збереження прогресу	Класичне збереження при взаємодії в контрольних точках	Автоматичне збереження по проходженню локації.
Складність	Помірна з поступовим зростанням	Висока, із необхідністю повторного проходження
Розвиток персонажа	Дерева здібностей, чари	Випадковість в отриманні та поліпшенні зброї

Таблиця 1.3

Переваги та недоліки

Гра	Переваги	Недоліки
Hollow Knight	- Висока якість графіки - Атмосферність та сюжетна глибина - Тривалість гри	- Відсутність інструкцій для новачків. Гравець повинен сам знайти вірний шлях
Dead Cells	- Продумана бойова система - Динаміка та реіграбельність - Підтримка модифікацій	- Надто складна для непідготовленого гравця - Малий вплив сюжету на гру

1.4 Середовище розробки

Для розробки програмного продукту в межах кваліфікаційної роботи було обрано рушій Unity. В ігровій індустрії є два основних ігрових рушія - це Unreal Engine та Unity. Хоч вони і є прямими конкурентами, проте займають зовсім різні ніші індустрії.

Так перший концентрується на іграх від середнього до AAA сегменту. Проекти на його основі розробляються великими корпораціями. Здебільшого в них намагаються досягти добре деталізованих текстур та великого за об’ємом світу. Як мову програмування використовують C++, що має доволі високий поріг для розуміння принципів його роботи. До особливостей також можна віднести те що на Unreal Engine для реалізація простих задач може знадобитися надлишкова кількість робіт в порівняні з іншими рушіями, проте зі складними ніяких проблем немає. Взявши до уваги перелічену інформацію та узагальнивши її можна сказати що ідеальним сценарієм для використання Unreal Engine є масштабний тривимірний проект з фотореалістичними текстурами та великим відкритим світом. З цього випливає що описаний рушій в контексті одноосібної розробки є черезмірним, та не задовільняє реалізацію поставлених завдань.

Unity є його повною протилежністю. Розробка скриптів відбувається на мові C#, що має значно нижчий поріг складності в порівнянні з C++. Закономірно через це інструментарій для оптимізації масштабних тривимірних проектів є набагато обмеженішим та здебільшого в цьому плані показує гірші результати в порівнянні з конкурентом. Проте рано відносити це до недоліків, як було сказано раніше обидва рушія займають різні ніші індустрії. Протилежним до AAA сегменту є інді. Інді-ігри розроблюються без фінансової підтримки великих видавців, здебільшого це невелика компанія інтузіастів, або навіть одинокі розробники. Особливе місце для рушія становлять двовимірні ігри, адже він має потужні засоби для роботи з 2D графікою та покадровою анімацією, а проблеми з обмежено оптимізацією в описаних умовах є повністю неактуальними. Виходячи з цього це є ідеальним простором для навчання та входження в індустрію. Можна сконцентруватися на безпосередній реалізації різноманітних ідей. Варто лише згадати що одне з вже розібраних рішень, а саме Hollow Knight створене за допомогою Unity.

1.5 Висновки до першого розділу

Провівши дослідження предметної області було досягнуто розуміння, що ігрова індустрія є багатогранною та в умовах нинішніх реалій не може існувати окремо від інших індустрій через те, що в своїй основі вона є результатом роботи десятків тисяч працівників з різноманітних галузей.

Було розглянуто існуючі реалізацій поставлених задач на прикладах відеоігор розроблених командами відомих проектів та прийнято рішення відтворити у власній роботі контролер героя подібний до представленого в Hollow Knight, з Dead Cells натомість взяти основний концепт механік бойової системи.

На роль ігрового рушія проаналізувавши можливі варіанти було обрано Unity. На відміну від його прямого конкурента в образі Unreal Engine є дружелюбним до малих проектів з невеликими командами розробки. І що найголовніше надає потужний, неперевантажений інструментарій для розробки двовимірних ігор в будь-якому жанрі.

2. МОДЕЛЮВАННЯ СИСТЕМИ

2.1 Взаємодія з системою

Взаємодія з програмним продуктом відбувається через персонажа гравця. Тому важливо є заздалегідь змоделювати функціонал притаманний для нього. На рис.4 зображено діаграму прецедентів для гравця.

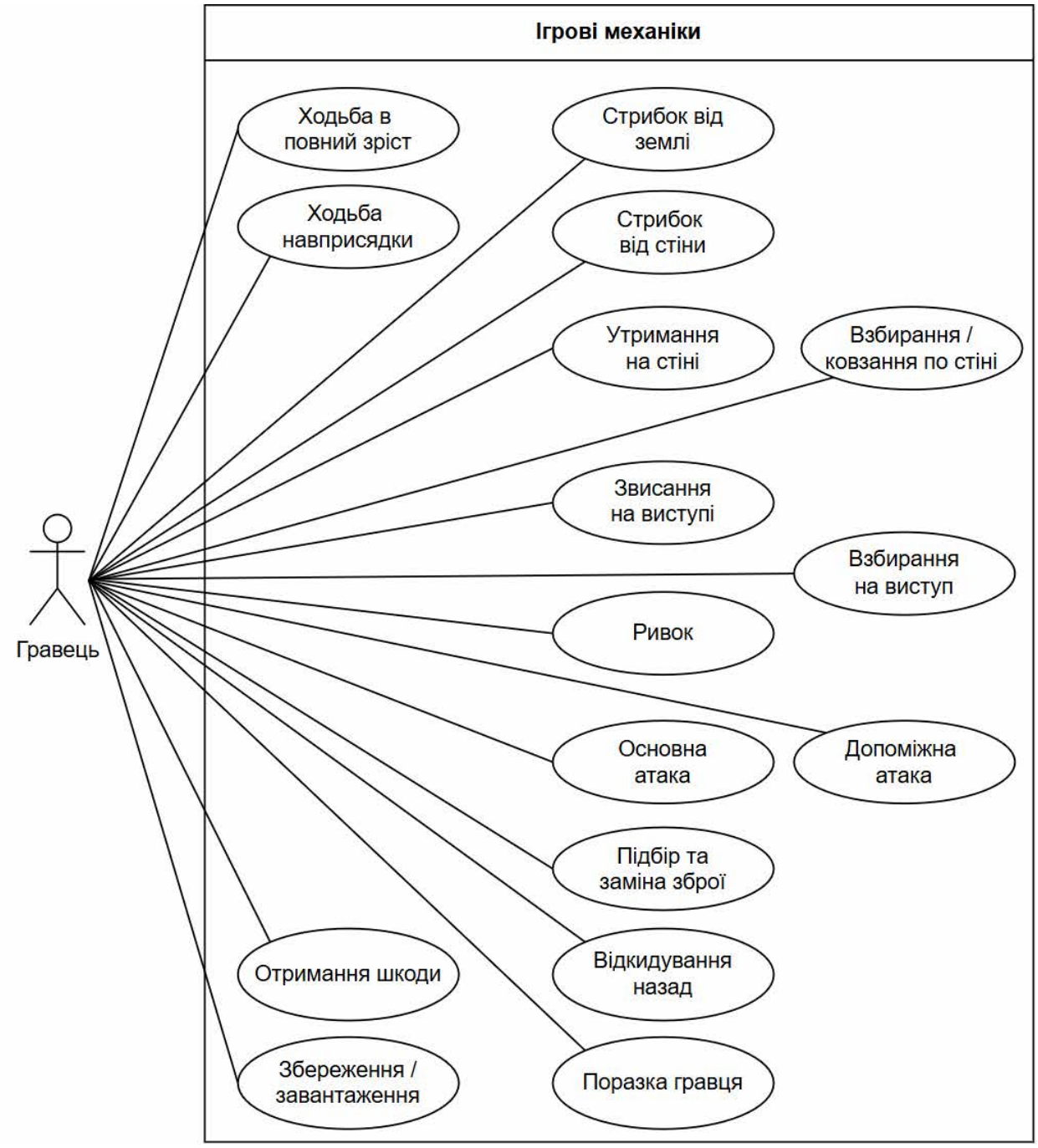


Рис.4 Діаграма прецедентів ігрових механік гравця

Вказані механіки поділяються на два типи. Перший є результатом введення користувача. Кожен представник має прив’язку до певної клавіші. В подальшому для уникнення плутанини буде використовуватись англійська розкладка. Другий тип це реакція персонажа на зовнішні фактори.

Механіки першого типу:

- Ходьба в повний зріст. Клавіші взаємодії «A» для переміщення вліво та, «D» вправо.
- Ходьба навприсядки. Клавіші взаємодії це комбінація з затиснутої «S» + «A» або «D». При активації зменшує розміри гравця по вертикалі в два рази при цьому перемасштабовуючи колайдер.
- Стрибок від землі. Клавіша взаємодії «Space». При утриманні збільшує висоту стрибка до двох разів.
- Стрибок від стіни. Активується при натисненні «Space» перебуваючи на стіні. Персонаж виконає стрибок в протилежному стіні напрямку. Якщо з іншого боку в радіусі стрибка знаходиться ще одна стіна то передбачити можливість повторного виконання прийому.
- Утримання на стіні. Клавіша взаємодії «Ctrl». Поки утримується персонаж перебуває на одному рівні стіни.
- Взбирання / ковзання по стіні. Взбирання активується комбінацією клавіш «Ctrl» + «W», а ковзнання «Ctrl» + «S». При цьому персонаж рівномірно пересувається у відповідній площині.
- Взбирання на виступ. Доступне якщо активне звисання на виступі після чого необхідно натиснути на «A» або «D». Під час відпрацювання блокуються всі інші механіки.
- Ривок. Клавіша взаємодії «Shift». Ривок виконується в одному з восьми напрямків. Для вказання напрямку використовується позиціонування миші по відношенню до гравця. При утриманні «Shift» час сповільнюється в чотири рази протягом половини

секунди, а біля гравця з’являється індикаторна стрілка, що візуалізує напрямок в якому буде виконано переміщення.

- Підбір та заміна зброї. Клавіша взаємодії «E». Для виконання потребує наявність предмету в радіусі підбору. У разі успіху розміщує обладнання у вільній комірці інвентаря. Якщо інвентар переповнений, то викликається вікно яке пропонує замінити наявний предмет на новий. Після вибору з представлених варіантів старе озброєння викладається на землю.
- Основна атака. Клавіша взаємодії «LMB». Доступна лише за умови екіпірованої зброї в першій комірці.
- Допоміжна атака. Клавіша взаємодії «RMB». Доступна лише за умови екіпірованої зброї в другій комірці.
- Збереження / завантаження. «F5» відповідає за збереження сесії активного рівня. Натомість «F9» завантажує релевантні дані в обраний рівень.

Механіки другого типу:

- Звисання на виступі. Активується автоматично при виявленні персонажем виступу за який можна ухватитися. При цьому позиція фіксується з очікуванням подальших дій.
- Отримання шкоди. Активується за умови влучення хітбоксу атаки ворога в колайдер гравця.
- Відкидування назад. Наслідок отримання шкоди. Переміщує гравця у напрямку атаки ворога на визначені значення.
- Поразка гравця. Тригером для цієї події є наявність у персонажа нульового або від’ємного значення здоров’я.

Для обробки введення з пристрою Unity має декілька рішень. В межах даного проекту цікавим є лише підхід під назвою New Input System [7]. Він дозволяє налаштовувати карт дій, що відповідають за обробку введення з різних пристроїв. На рис.5 зображено карту дій сформовану на основі розібраних

прецедентів. Вона також має дві схеми, що призначаються для роботи клавіатури та геймпаду.

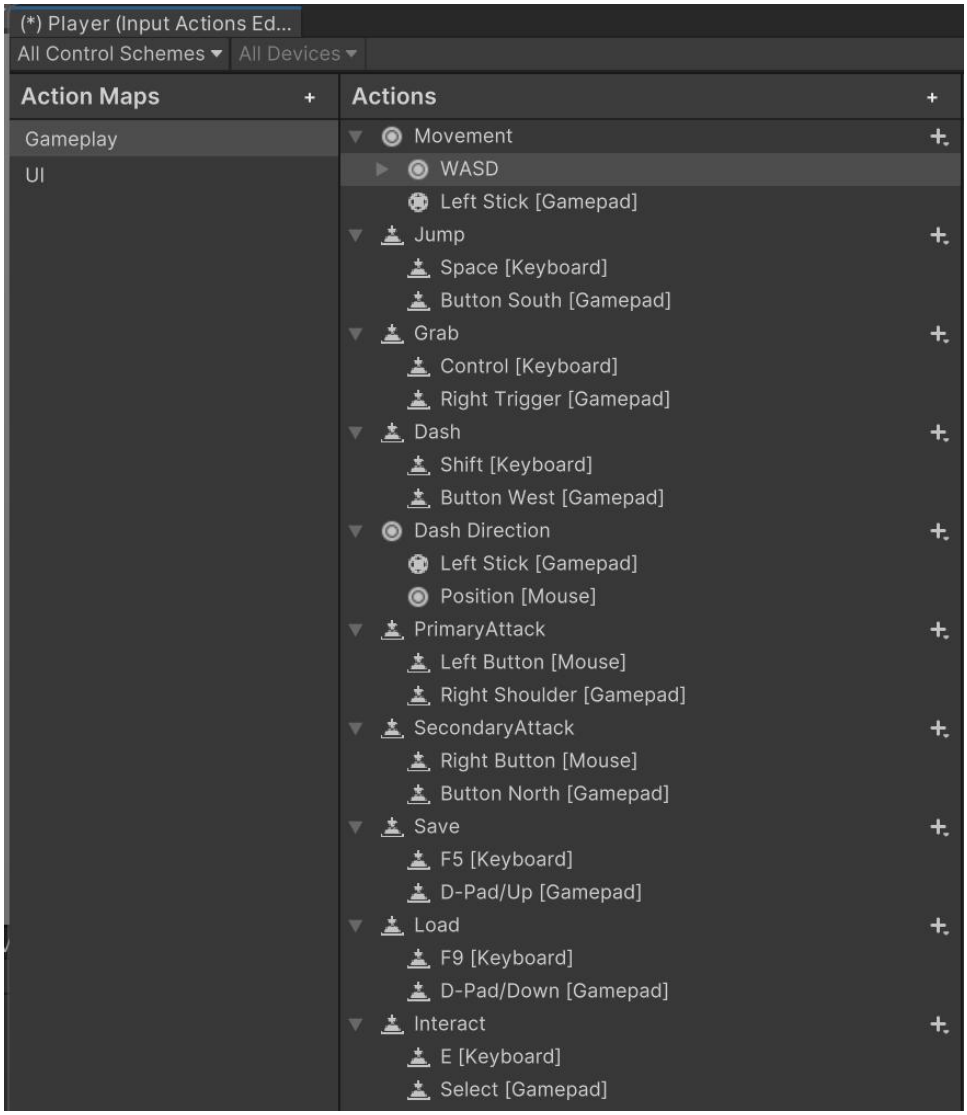


Рис.5 Карта обробки активних дії.

2.2 Структура організації даних

Проект є однокористувацькою програмою без серверної взаємодії, вся обробка та оновлення інформації виконується на пристрої користувача. Використання повноцінної бази даних є черезмірним для даного проекту, тому структура організації даних представлена у вигляді папки «Ресурси»(Assets) так як вона у собі містить всі об’єкти взаємодії проекту. Її структура являє собою

ієрархію папок, кожна з яких відповідіє за певний набір даних, об’єктів та реалізацій взаємодії. Так на рис.6 показано її вміст всередині редакора Unity.

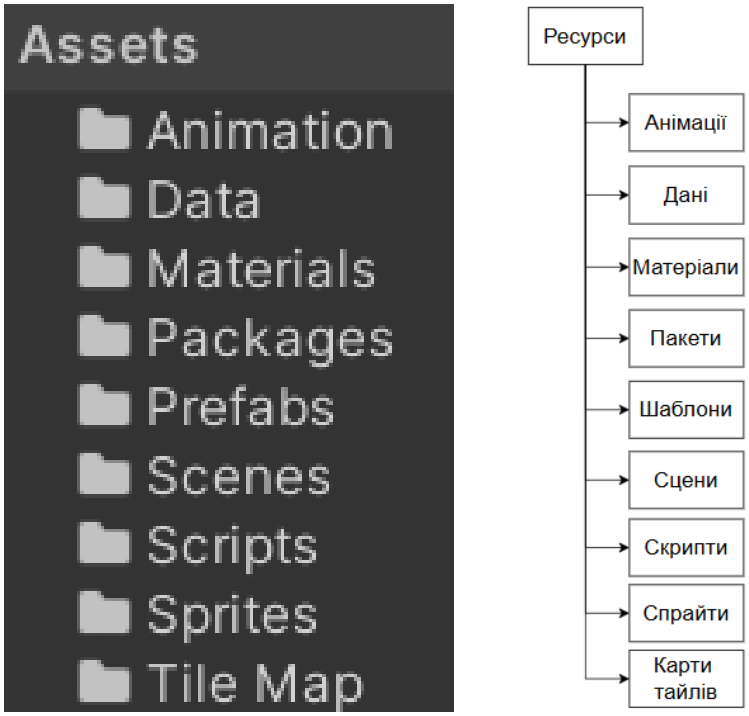


Рис.6 Ієрархія папки «Assets» та локалізована версія.

Для більш чіткого розуміня значення цих папок та об’єктів що вони містять, потрібно розглянути їх детальніше.

2.2.1 Анімації(Animation). Поділяється на дві підпапки, одна містить готові анімації, інша - контролери.

Анімація – файл з даними, які описують зміну властивостей компонентів об’єкту протягом певного часу. Властивостями можуть бути: спрайти, позиція, масштаб, кут обертання і тому подібне, розмір колайдеру. Анімації можуть відпрацьовуватися циклічно, по завершенню останнього кадру наступним буде перший. Час виконання одного циклу залежить від кількості кадрів що показується за одну секунду, цей показник називається Samples і встановлюється для кожного кліпу окремо. На рис. 7-9 показано вміст однієї з аніміцій гравця. В даному випадку властивістю є лише спрайти гравця.

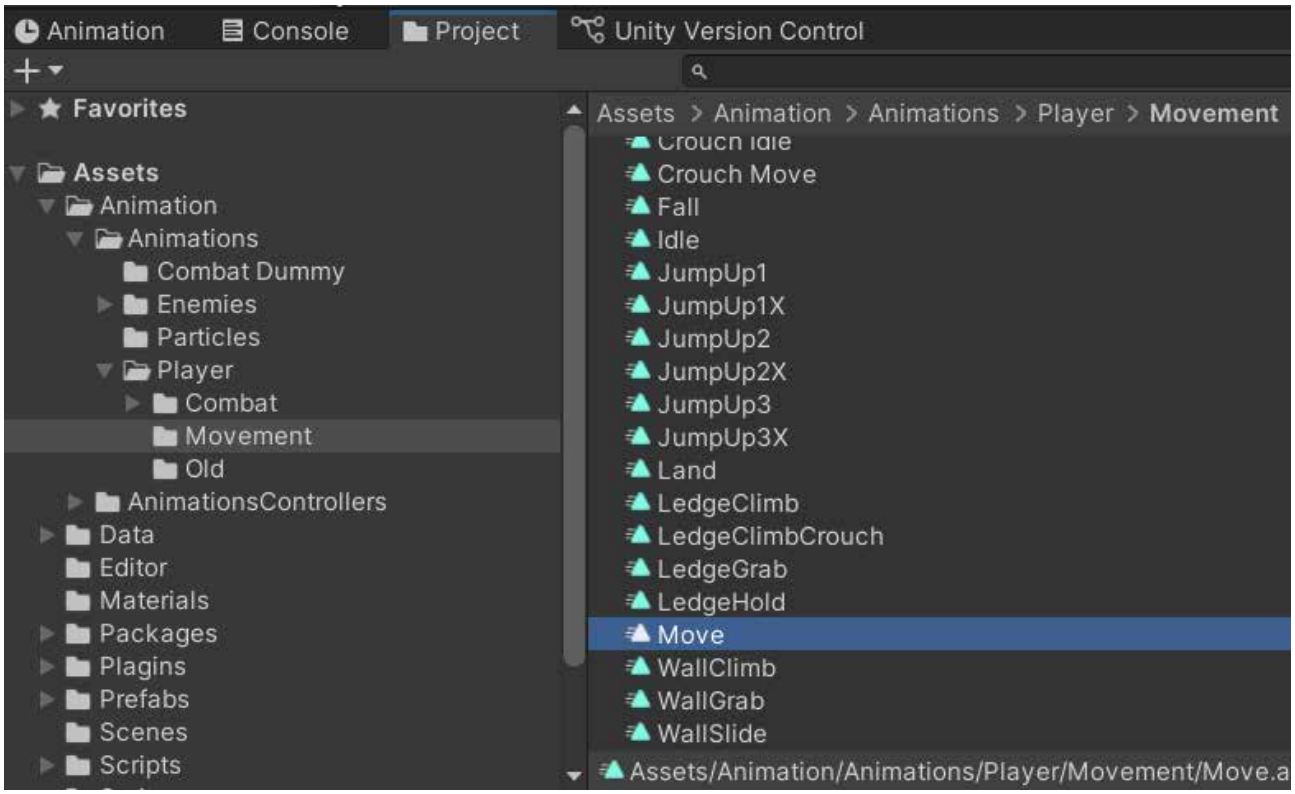


Рис.7 Шлях до анімації в ієрархії

Рис.8 Властивості та кількість кадрів

Рис.9 Половина аніміції руху гравця (спрайти 1-6 з 12)

Контролери(Animator Controller) – спеціальний об’єкт що керує набором анімацій переключаючи їх на основі певних умов описаних розробником. Зазвичай такою умовою є значення відправлене скриптом внаслідок свого

відпрацювання. За ці параметри приймають цілі та дробові числа, булеве значення та тригер. Кожен контролер має три базових стани. «Вхід(Entry)», що при відсутності додаткових умов перегодить до встановленої за замовченням анімації. «Вихід(Exit)» використовується як заключна точка при попаданні в неї відбувається перехід до стану «Вхід». «Будь-який(Any state)» перехоплює стан та передає далі за умови виконання всіх визначених умов незалежно від поточного позиції в дереві конролера. На рис. 10 зображено контролер анімацій гравця. Основна проблема класичного підходу до реалізації дерева аніматора полягає в його заплутаності та складності масштабації, адже при доданні нової анімації потрібно також зробити всі пов'язані з нею переходи. Цей підхід було модернезовано. Справа у тому що аніматор є наочним прикладом реалізації рушієм Unity основних принципів патерну машини кінцевих автоматів, а так як цей же шаблон проеткування буде використано при розробці контролерів персонажів що мають відповідні аніматори, то логіку переходів буде залишено з ним, тим самим можна спростити дерево анімацій, та уникнути дублювання станів між контролером персонажа та його аніматором. Фінальна структура контролеру анімацій має вигляд:

- Перехід між анімаціями не описується
- Ітерація починається зі стану «Вхід»
- Далі на основі наявних параметрів переходить до анімації, що їм відповідає, якщо така відсутня, то виконується обрана по замовченню.
- Після відпрацювання відбувається перехід до стану «Вихід».
- Ітерація повторюється.

На практиці такий підхід є дуже подібним до використання базового стану «Будь-який». Патерн машини кінцевих автоматів буде розглянуто більш детально в наступних підрозділах.

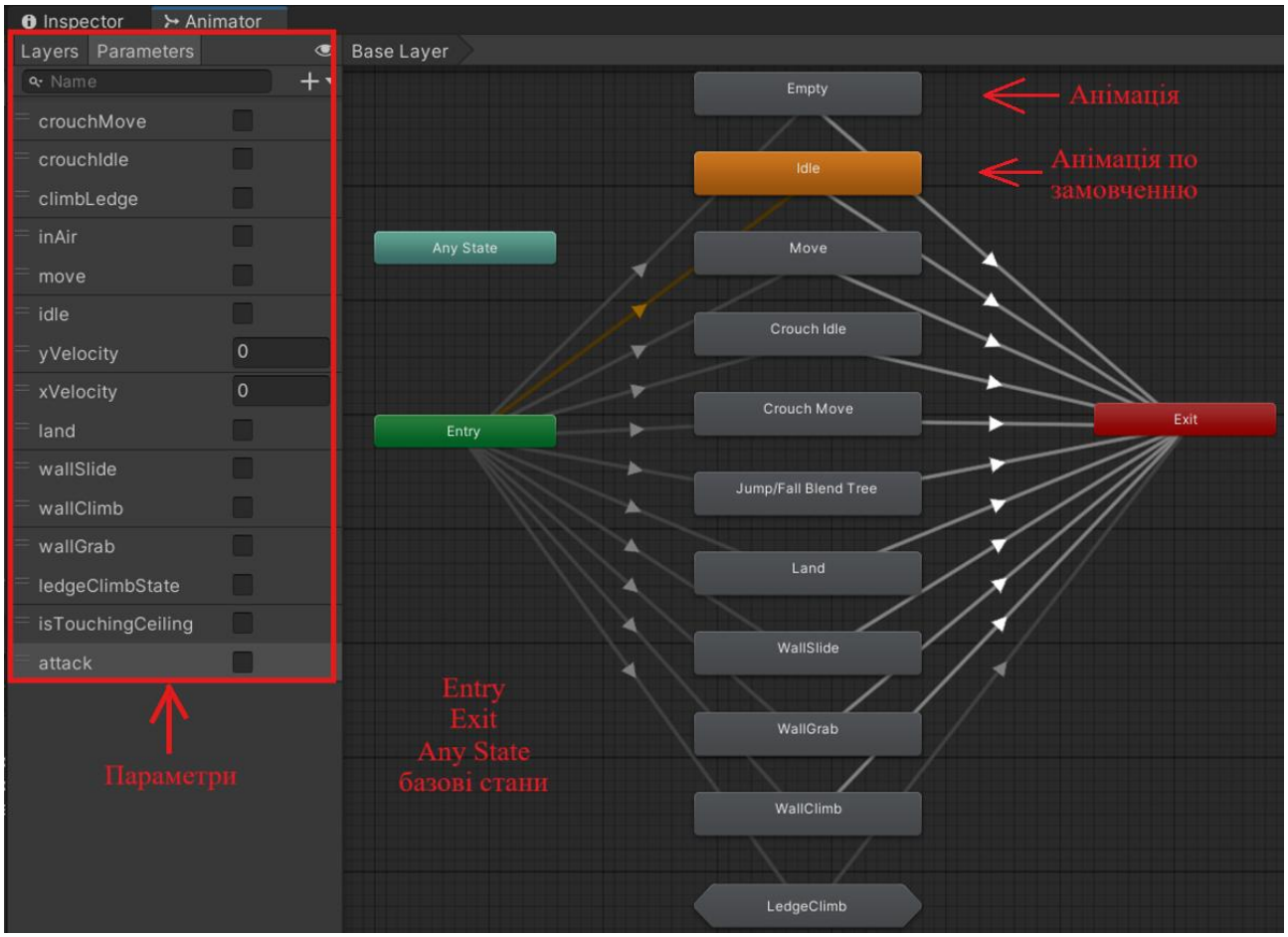


Рис.10 Контролер анімацій гравця

2.2.2 Дані(Data). Ця папка містить об’єкти сховища даних, що також відомі як скриптовані об’єкти(scriptable object). Створити їх можна в редакторі Unity, проте конкретні екземпляри не визначені ігровому рушії по замовченню. Вони є повністю кастомними, тому для кожного типу сховищ потрібно розробляти свою власну структуру. В коді що відповідає за перелік даний скриптованого об’єкту можна визначити значення змінних по замовченню що будуть підставлені при створенні екземпляру. В папці зберігаються у вигляді файлу з розширенням .asset.

Для створення потрібно виконати наступні кроки:

1. Натиснути на праву кнопку миші у будь якому місці вікна папки проекту.
2. Обрати «Створити(Create)»
3. Обрати «Дані(Data)»

4. З отриманого переліку обрати для чого будуть використовуватися дані.
5. Обрати тип даних.

На рис. 11-13 зображено приклад створення та заповнення даними скриптованого об’єкту з редактору.

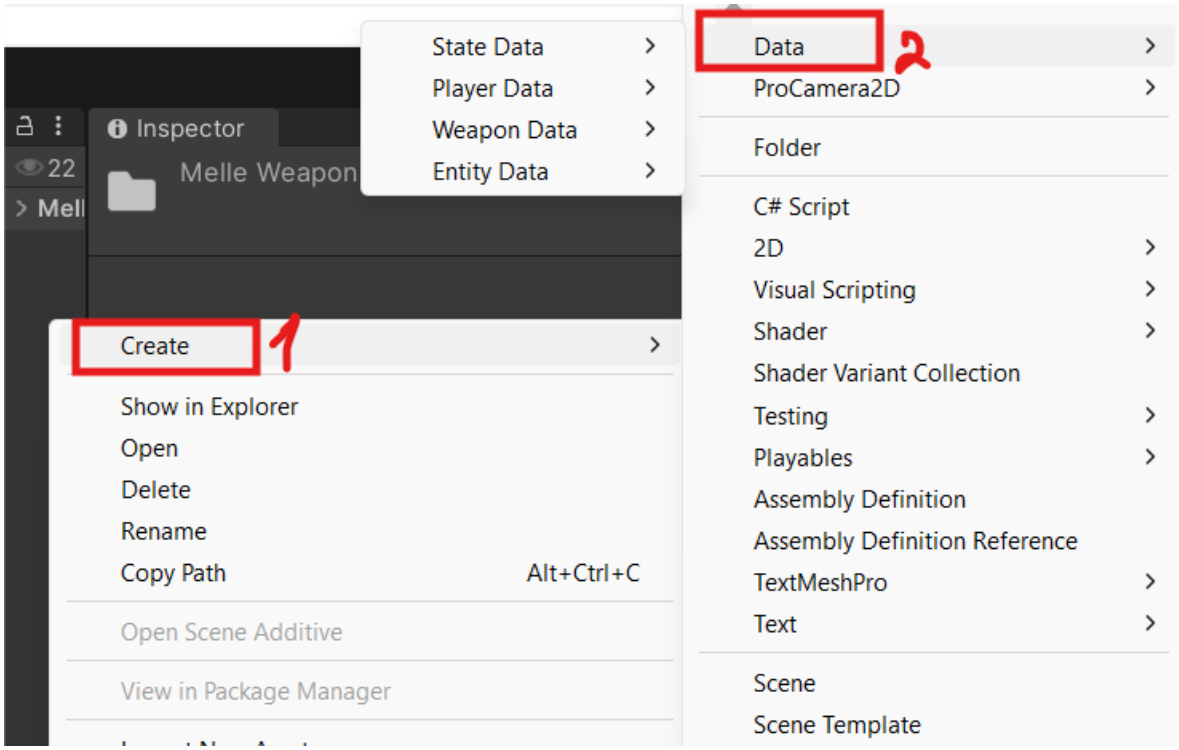


Рис.11 Вибір кастомного меню в редакторі

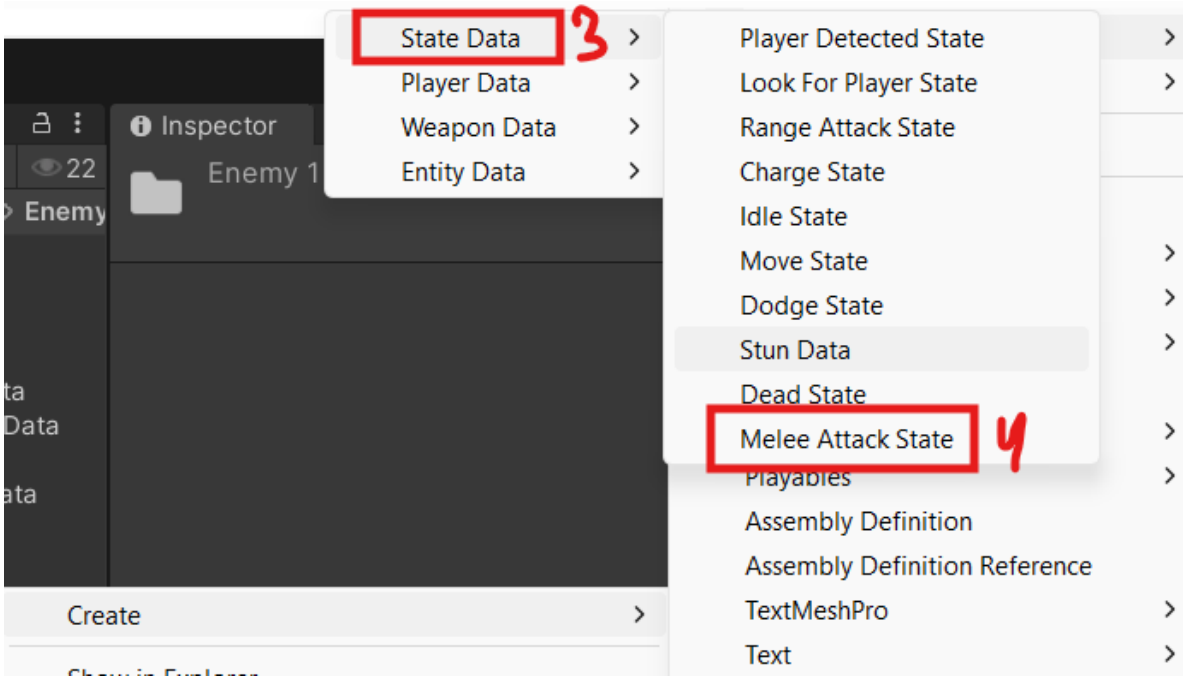


Рис.12 Створення об’єкту даних

Рис.13 Заповнене сховище даних

2.2.3 Матеріали(Materials). Папка містить екземпляри, що визначають, як поверхня двовимірного об’єкту буде виглядати при рендерингу. Складаються з шейдеру та певних параметрів таких як колір, текстура, прозорість, чи навіть текстура.

В межах цього проекту матеріали вирішують дві проблеми. Перша полягає в невірному відображенні тайлів, а саме без їх використання наявні щілини між квадратами тайлів, що робить картинку не цільною та дискомфортною. Поняття тайлів буде розглянуто в пункті 2.2.9.

Для розуміння другої проблеми потрібно також зазначити, що існують фізичні матеріали, що поділяються на двовимірні та тривимірні. Вони не впливають на рендеринг, проте застосовуються при обчисленні фізики. Рушій Unity симулює тертя при русі об’єктів. Таким чином виникає проблема, а саме коли гравець знаходиться в повітрі впритул до стіни та продовжує рух в її бік то виникає тертя, що сповільнює спускання персонажа на землю. Це виглядає неестетично та є непрактичним. Тому в параметрі двовимірного фізичного матеріалу показник тертя повинно бути встановлено на нулі. Цей же принцип можна використати і для персонажів противників, щоб уникнути непередбаченого відпрацювання логіки.

2.2.4 Пакети(Packages). Містить в собі плагіни та розширення, що використовує проект. Для цього проекту найбільш цікавими є ProCamera2D, ClosedXml, TextMeshPro.

ProCamera2D – потужний плагін керування камерою в двовимірних світах. Основне призначення полягає в переміщенні камери за гравцем. Має цікаву

функцію «Фокус попереду». Центрує камеру не на цільовому персонажі, певній точці спереду. Важливим є те що на відміну від тривимірних проектів поняття «попереду» є неоднозначним. Можна провести аналогію, європейська людина може подумати про те що всі об’єкти які знаходяться справа від гравця є попереду, ототожнюючи це з правилами письма, де слова розміщуються зліва – направо, проте арабські народи мають прямо протилежні звичаї. А традиційна китайська та японська писемність взагалі зверху – донизу. Тому важливим є визначити, що мається на увазі під цим терміном. Для цього плагіну точка «попереду» визначається динамічно, згідно з останнім вектором руху цільового об’єкту на заздалегідь визначеній відстані. Після припинення руху камера плавно переміщується від гравця до точки, впродовж певного відрізку часу.

ClosedXml – Інструмент для обробки та створення таблиць в форматі .xml та .xmls. В контексті проекту потрібен для формування звітів на основі збережень. Дозволяє програмно редагувати файл після внесення в нього даних. Наприклад перемасштабовувати розмір рядків та колонок так, щоб весь вміст в них був видимий.

TextMeshPro – набір інструментів для роботи з текстом в Unity. Використовується як в двовимірних так і у тривимірних проектах. Ігровий рушій по замовченню має текстовий компонент UI.Text, проте в ньому відсутні базові рішення такі як шрифти з редакторів таких як Word, для їх використання потрібно спочатку їх завантажити з простору інтернету та вручну додати. В TextMeshPro ця робота уже виконана.

2.2.5 Шаблони(Prefabs). Ця папка містить готові зразки об’єктів, що можуть бути використані на ігровій сцені. Шаблони є потужним інструментом, що має ключовий вплив на формування ігрових світів.

Для початку потрібно зрозуміти, що взагалі таке ігровий об’єкт. Ігровий об’єкт в Unity – це основа яка сама по собі не виконує ніяких дій та ні з чим не взаємодіє. Справа в тому, що Unity використовує компонентну систему щодо створення об’єктів, де його екземпляр це просто контейнер для компонентів, а ось уже вони описують логіку його поведінки та взаємодії. Існують різноманітні

компоненти від кастомних, будь який розроблений скрипт можна вважати за такий, до більш класичних таких як фізичне тіло, колайдер, параметри розташування. Цікавим є те що розглянуті вище контролери анімацій та матеріали також є їх представниками. Всередині можуть використовуватися будь які типи даних в тому числі посилання на скриптовані об’єкти розглянуті раніше та навіть представники інших компонентів.

Шаблоном може бути будь який ігровий об’єкт переміщений зі сцени в будь-яку папку всередині «Assets». Зазвичай розробники називають цю папку «Prefabs». Після переміщення об’єкт залишається на сцені, проте набуває блакитного кольору. В такий спосіб Unity інформує розробника, що перед ним не об’єкт, екземпляр шаблону, зміна якого ніяк не повпливає на оригінал. Звісно передбачена можливість застосування змін до оригіналу через його зразок, проте це відбувається в ручному режимі. На сцені можна розмістити безліч екземплярів, кожен з яких є самостійною, незалежною одиницею. Важливим є те, що будь які зміни в шаблоні приводять до аналогічних дій в його екземплярах. Як видно на рис.14 в папці противників знаходиться шаблон лише одного противника «Monster». На сцені серед об’єктів персонажів (Combat Test Dummy, Monster 1, Archer 1, Player) блакитний колір має лише «Monster 1», саме він і є екземпляром шаблону.

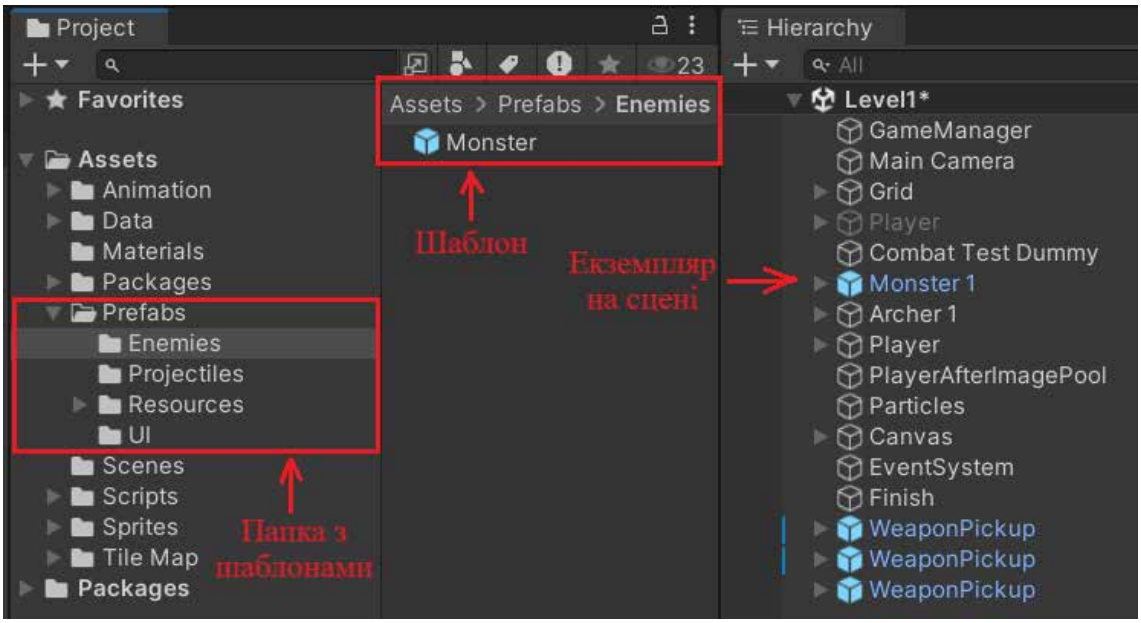


Рис.14 Використання шаблону в проекті

2.2.6 Сцени(Scenes). У рушії Unity для розташування усіх об’єктів у певному просторі використовуються сцени. Їх екземпляри ізольовані між собою. Одночасно функціонує лише одна, яка є активною. Кожна має свій порядковий номер. Нумерація починається з нуля подібно до масивів, цей порядок можна налаштувати. При запуску програми позамовченню відкриється сцена з нульовим індексом. Виходячи з цієї інформації на даному етапі можна визначити правила нумерації сцен. Нульовою є головне меню, всі інші це рівні з відповідними номерами. При перезапуску сцени, дані сесії в ній не зберігаються. Наприклад, якщо з першого рівня вийти в меню після чого знову відкрити рівень то прогрес буде анульовано. Це не є проблемою, адже зазвичай подімні ігри використовують таку ж систему. Тим більше в рамках проекту буде розроблено механіку збережень, що невілює цю особливість за потреби.

2.2.7 Скрипти(Scripts). Папка, що містить весь програмний код проекту. Для Unity скриптом є файл з кодом написаний на мові програмування C#. Він описує логіку та поведінку об’єктів гри.

Основні типи скриптів:

1. MonoBehaviour. Базовий клас, що дозволяє взаємодіяти з життєвим циклом Unity. Має реалізацію таких ключових методів як:
    - Awake(). Викликається при створенні об’єкта.
    - Start(). Викликається в першому кадрі, якщо його компонент активний.
    - OnEnable() / OnDisable(). При активації об’єкту чи завантаженні сцени для першого і прямо протилежна дія для другого.
    - Update(). Виконується в кожному кадрі. В ньому обраховується логіка, що потребує постійних змін.
    - FixedUpdate(). Виконується фіксовану кількість раз за секунду. За замовченням це значення дорівнює 50. Основним призначенням є розрахування фізики та подібних задач які

потребують змін на прогнозованих відрізках часу. Справа в тому, що по замовченню Update() обмежується лише технічними характеристиками пристрою. Тому наприклад, якщо виконувати в ньому оновлення переміщення об’єкта зі статичною швидкістю, то на практиці кінцева відстань що пройде об’єкт за однаковий проміжок часу на різних машинах буде кратно відрізнятися. Навіть в межах однієї машини досягти одинакового результату неможливо. Тому для таких ситуацій ідеальним рішеннням є FixedUpdate().

- Група OnCollision2D. Складається з методів Enter, Stay, Exit. Викликається для об’єкта з яким відбулося зіткнення колайдерів, продовження зіткнення та кінець зіткнення відповідно. Колайдер(Collider) – це компонент Unity, що дозволяє об’єктам взаємодіяти. Collision, тобто колізія, власне і означає зіткнення колайдерів.
- Група OnTrigger2D. Має аналогічні до OnCollision2D методи з однією відмінністю, а саме в тому вони викликаються коли тригерний колайдер взаємодіє з фізичним. Тригер – надбудова над колайдером, при його однойменний активації в редакторі компоненту колайдера він перестає бути фізично осяжним проте входження в його зону фіксуються.

2. ScriptableObject**.** це клас, що дозволяє нащадкам зберігати дані як об'єкти, що доступні для редагування в інспекторі, хоча при цьому не є частиною звичайних ігрових об'єктів чи компонентів. Корисні у випадках збереження даних, що повинні бути спільними для кількох об'єктів, або при створенні налаштувань, які не повинні залежати від конкретного ігрового об'єкта. В пункті 2.2.2 уже було частково піднято тему створення та використання скриптованих об’єктів.

3. StateMachineBehaviour. Завдяки ньому у нащадків з’являється можливість керувати логікою окремих станів контролеру анімацій. Для використання слід додавати не напряму до ігрового об’єкту, а до його аніматору.
4. PlayModeTest. Спеціальний тип скрипта, з можливістю автоматичного тестування поведінки гри під час виконання. Тобто безпосередньо відповідає за юніт-тестування.

За допомогою скриптів можна реалізувати безмежну кількість ідей та механік. До прикладу в рушії передбачено можливість змінювати редактор під свої задачі програмним шляхом. В пункі 3.3.1 буде піднято тему розробки кастомного редактору для формування зброї гравця.

2.2.8 Спрайти(Sprites). Папка з графічними ресурсами проекту. На рис.15 перелічено всі типи зображень, що використовуються.

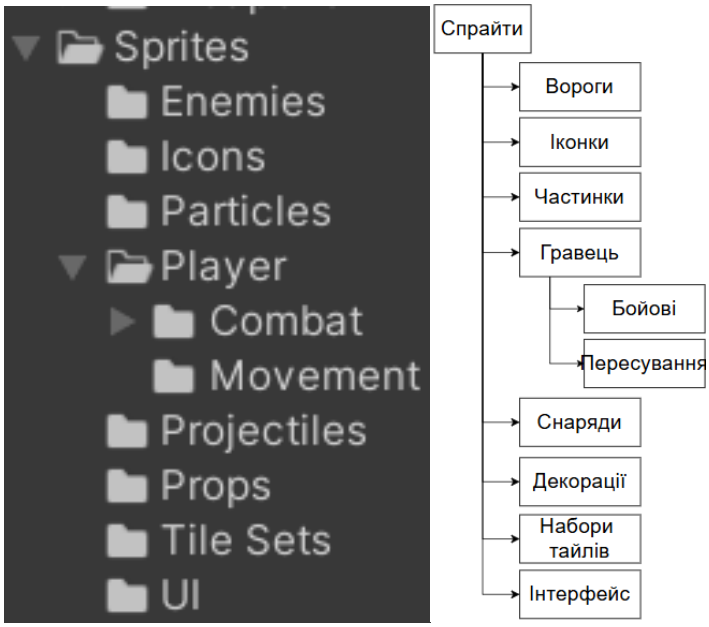


Рис.15 Перелік графічних об’єктів

2.2.9 Карти тайлів(Tile Map). Unity має зручне рішення для створення двовимірних світів на основі плиток, які можуть бути прямокотними, гексагональними та ізометричними. Для платформеру добре підходить перший варіант. Підхід базується на сітці, яка складається з клітинок. Кожна з них може містити одну плитку яку називають тайлом(tile). Її зображення є спрайтом. Карта

тайлів це шар, створений на основі сітки. Для кожного шару можна застосовуванти різні параметри [8].

Основні способи застосування шарів:

- Фізичний. На ньому розміщується поверхня з якою можуть взаємодіяти всі об’єкти з колайдерами.
- Фоновий. Містить неінтерактивні декораці та елементи заднього фону.
- Односторонні платформи. Комбінація з фізичного та фонового. Суть полягає в тому що з одного боку, взаємодія з об’єктами не відбувається, тому вони можуть легко його перетинати. Проте з в зворотньому напрямку шлях зачинено. В платформерах таке явище є дуже поширеним.

Тайли зберігаються в наборі приток(Tileset), що є файлом, зазвичай у вигляді сітки. Зображення для його створення імпортуються відповідного спрайту. На основі набору фомується палітра тайлів. Палітра використовується як інструмент для заповнення тайлмапів. На рис.16 справа зображено палітру, а зліва сцену намальовану з її допомогою.

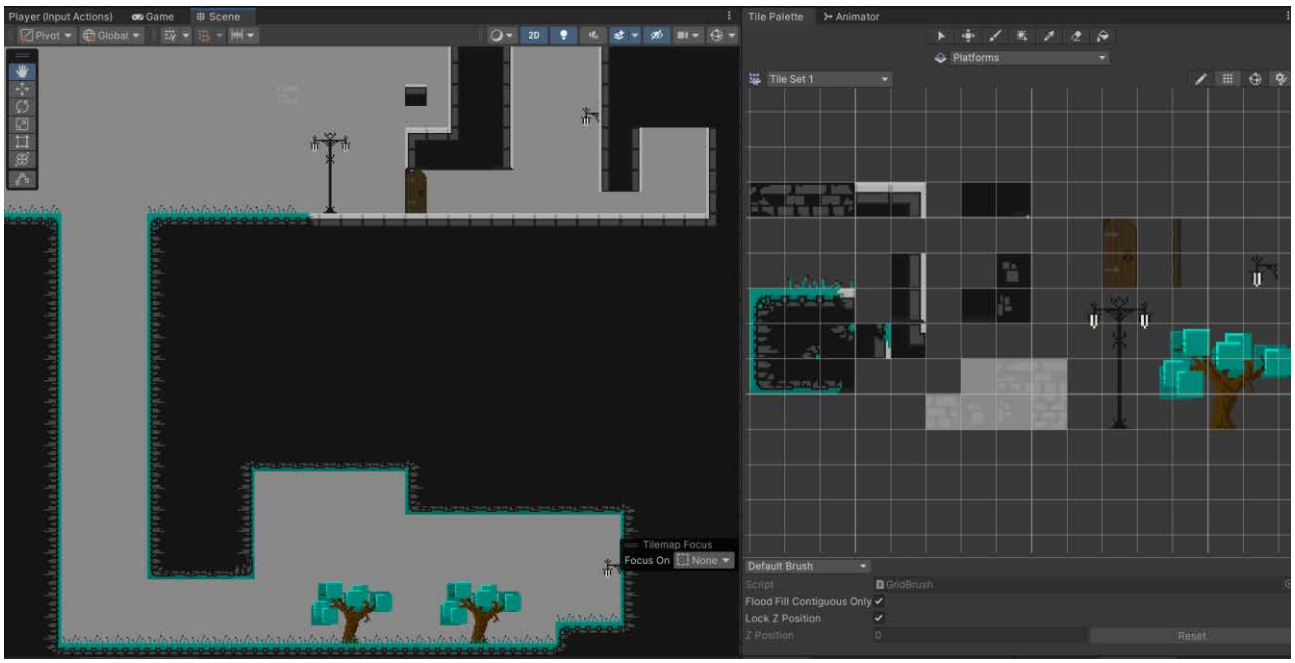


Рис.16 Сцена та її палітра

2.3 Висновки до другого розділу

В межах описаного розділу було піднято тему моделювання інформаційної системи. Вирішено, що користувач буде взаємодіяти зі застосунком через контролер гравця та інтерактивний інтерфейс. Основним пристроєм введення є клавіатура та миша, хоча крім них повинна існувати ще й альтернативна схема прив’язки клавіш для геймпаду. Щодо організації даних система не використовує бази даних, ресурси проекту зберігаються в однойменній папці Assets та розділенні на підпапки, кожна з яких відіграє власну роль описану в другому розділі. Дані зберігаються в різноманітних проявах. Від класичних по типу зображень в форматах .png або .jpg до більш специфічних таких як анімації, шаблони, або ж навіть створених власноруч сховищ даних, що зберігаються у форматі .asset. Додаток А містить діаграму, що графічно зображує використання даних папками структури.

3. РОЗРОБКА СИСТЕМИ

3.1 Машина кінцевих автоматів

3.1.1 Поняття машини кінцевих автоматів. Машина кінцевих автоматів – це шаблон проектування, який приймає на себе роль керування поведінкою об’єкта залежно від його поточного стану [9].

Принципи роботи цього шаблону проектування:

- Об’єкт може перебувати лише в одному стані в конкретний момент часу.
- Кожен стан описує правила роботи окремої специфічної логіки поведінки та взаємодії об’єкту. Так наприклад, до станів можна віднести ходьбу, атаку або стрибок.
- Перехід між станами відбувається внаслідок виконання певних умов.

Патерн вирішує одразу декілька проблем. Основні з них масштабованості та структуризації. Вони полягають у тому, що якщо спробувати реалізувати контролер поведінки персонажа не використовуючи принципи описаного шаблону проектування, то складність розробки відчутно збільшується з кожною новою механікою, адже для коректної роботи потрібно узгодити взаємодію всіх механік з новою, шляхом додання в кожну з них додаткових умов. Завдяки розділенню контролеру на окремі стани досягається чітка структура, що допомагає при отриманні непередбачуваних результатів швидко локалізувати та усунути причину.

3.1.2 Реалізація структурної частини. На основі перелічений понять було реалізовано патерн машини кінцевих автоматів. На рис.17 зображено варіант діаграми класів, що реалізує описаний патерн, який було спрощено для наочності. В додатку Б знаходиться більш розгорнутий варіант.

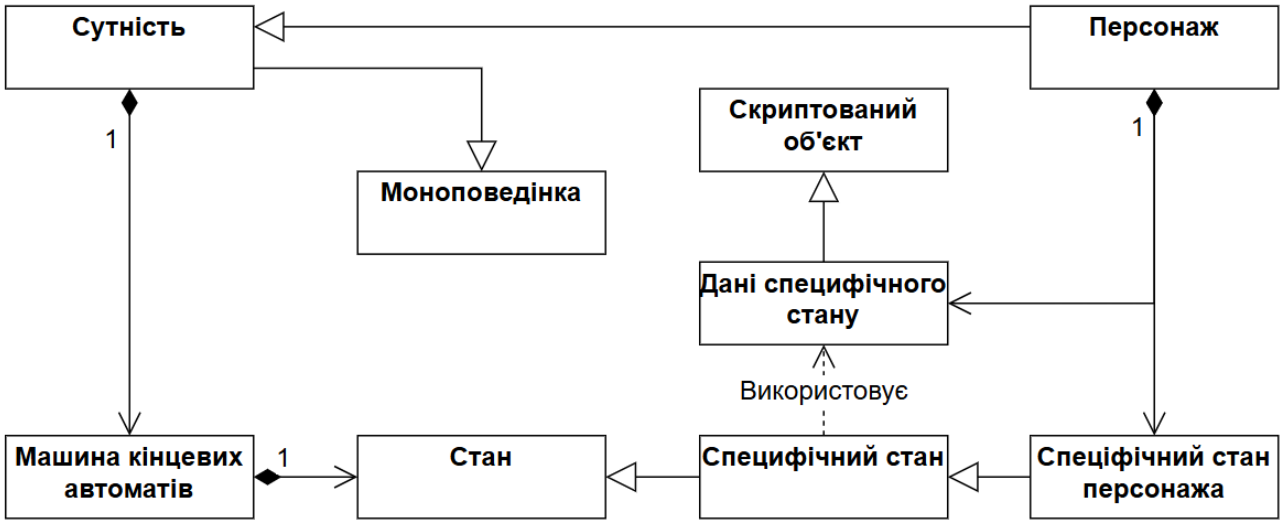


Рис.17 Спрощена діаграма класів патерну машини кінцевих автоматів

Класи «Моноповедінка» та «скриптований об’єкт» уже розглянуто в межах пункту 2.2.7.

Сутність – є базовим класом для всіх персонажів, наслідує моноповедінку та ініціалізує компоненти для функціонування об’єкту, такі як: Rigidbody2D(фізичне тіло об’єкту), Animator (контролер анімацій), початкові параметри, статуси розташування.

Персонаж – нащадок від класу «Сутність». Формує в собі образ певного персона, наприклад: лучник, боєць, монстр. Перелічує усі стани та дані для них, що будуть використовуватись в його межах.

Машина кінцевих автоматі - клас, що описує логіку ініціалізації та зміни станів.

Стан – описує що таке стан, та які основні функції він виконує, тому є базовим класом всіх нащадків. Хоч і не наслідується від «Моноповедінки» проте користується частиною його функцій через віртуальні відповідники, викликом яких займається клас «Сутність». Це є можливим через те що «Сутність» є нащадком «Моноповедінки». З цього випливає, що всі подальші нащадки базового класу «Стан» можуть використовувати деякі функції «Моноповедінки», без наслідування від нього.

Специфічний стан – нащадок класу «Стан». Є самостійною одиницею в межах якої визначається базову логіку поведінки для певного стану. Наприклад: атака, пошук гравця, переслідування, патрулювання.

Специфічний стан персонажа – нащадок класу «Специфічний стан». Корегує логіку відповідного специфічного стану під конкретного персонажа. Наприклад: атака лучника на відстані, переслідування гравця монстром. З першого погляду здається що цей клас просто дублює батьківський, проте важливість коригування логіки поведінки та переходів між станами полягає у тому, що розроблена система передбачає те, що різні персонажі мають різні стани, тому саме в цьому класі знаходяться умови переходу в інший стан.

Дані специфічного стану – наслідник «Скриптованого об’єкту», містить весь перелік унікальних характеристик, що використовуються при роботі стану. Також в ньому можна вказати значення даних по замовченню.

3.1.3 Реалізація поведінки персонажів. Для початку можна описати логіку переходів між станами для нескладного персонажа.

Стани та умови їх переходів для персонажа монстра:

1. Стан руху. При створенні об’єкту є початковим. Виконує рух у бік в який повернутий спрайт монстра. Рух виконується доти, поки не буде виконано одну з двох умов:
    - Перша – виявлено персонажа, як наслідок виконується перехід до стану 3.
    - Друга - виявлено стіну чи обрив, після чого перехід до стану 2.
2. Стан спокою. Виконується зупинка. Після визначеної затримки персонаж оглядається, якщо в цей момент було помічено гравця, то монстр переходить до стану 3, інакше розпочинає рух у зворотному напрямку з переходом до стану 1.
3. Стан виявлення гравця. Виконує роль проміжного стану, під час його відпрацювання виконуються наступні перевірки:

- Якщо гравець в полі зору, але не в радіусі атаки, то противник переходить до стану 4.
- У разі знаходження в зоні атаки, то активується стан 5.
- Якщо за час перебування в цьому стані та виконання перевірок гравець встигнув вийти з поля зору, то наступним станом буде 6.

4. Стан ривку до гравця. Виконується пересування в бік перед монстром з визначеною швидкістю протягом певного часу. Якщо під час виконання ривка досягнуто зони атаки, то виконується перехід до стану 5.
5. Стан атаки. Монстр виконує атаки після чого перевіряються такі умови:
    - Якщо гравець на відстані ураження, то після певної затримки атака повторюється.
    - Якщо в полі зору, проте поза зоною ураження, то стан переходить до 3.
    - Якщо гравця втрачено, то стан переходить до 6.
6. Стан пошуку гравця. Персонаж розвертається для того щоб оглянути територію позаду себе, після чого знову повертається в початковий бік. Якщо на якомусь з етапів було помічено гравця, то виконується перехід до стану 3, інакше 1.
7. Стан приголомшення. При реєстрації влучання гравцем по монстру крім отримання шкоди для здоров’я також вичерпаються показник захисту від приголомшення, коли він досягне нуля, то персонаж отримає статус «приголомшено», що супроводжується відповідною анімацією, та недієздатністю протягом певного проміжку часу. Після чого у разі виявлення гравця відбувається перехід до стану 4, інакше 6.

На рис.18 зображено графічне представлення цієї логіки.

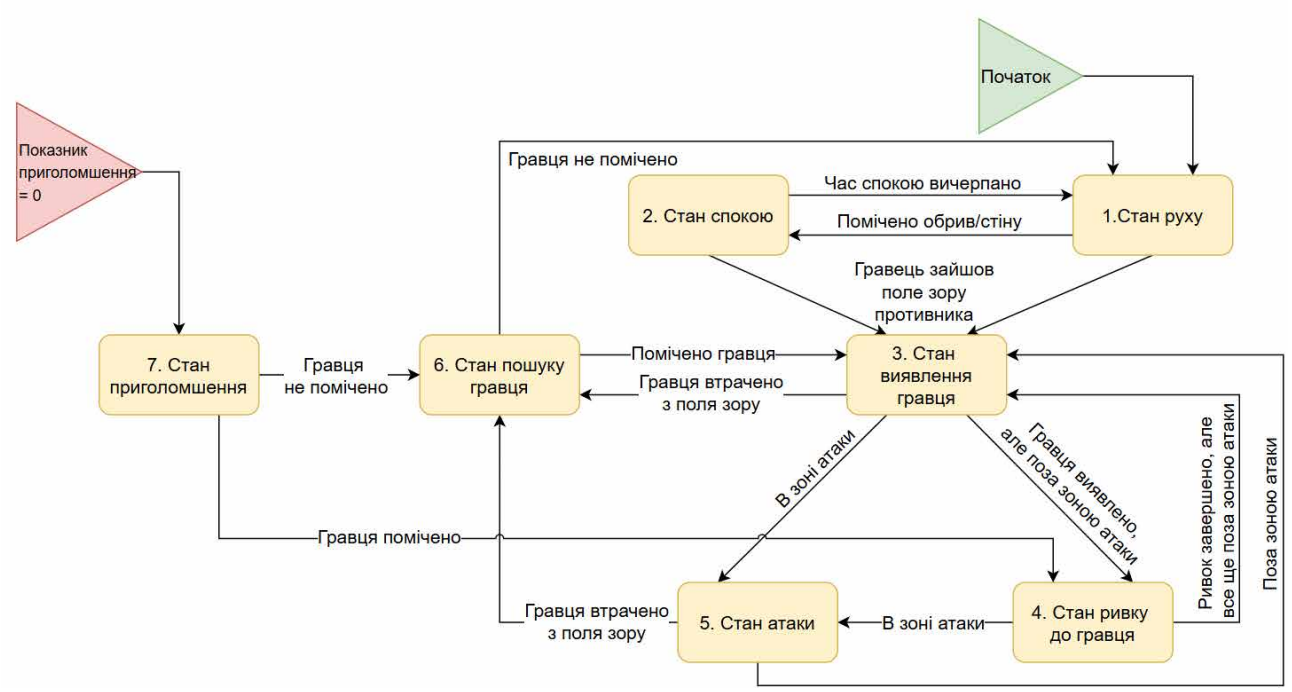


Рис. 18 Діаграма станів для монстра.

Важливим доповненням є те яким саме чином противник «помічає» гравця. У цій механіці лежить простий принцип променевого трасування. Цей метод дозволяє випустити уявний промінь з певної точки і напрямку, та виконати перевірку на зіткнення з чимось. По замовченню відстань обмежена лише першим колайдером з яким відбулася взаємодія, проте також можна вказати конкретні значення, що робить з нього не промінь, а радше відрізок. Також можна уточнити тег колайдеру з яким буде відбуватися реакція. В даному випадку гравець має однойменний тег, про який знає противник.

Наступною на черзі є реалізація машини кінцевих автоматів для гравця. Вона є складнішою ніж для ворогів, адже потребує впровадження більшої кількості механік. Для кращої структуризації, стани поділено на два типи: суперстани та субстани. Перший тип може мати нащадків вигляді представників другого. Цей тип використовується для швидкого переривання поточного стану, з цілю переходу в інший стан будь якого типу. Важливо уточнити, що суперстан може перервати роботу лише у субстанів нащадків. Другий тип це звичайний стан, який необов’язково має бути нащадком суперстану. На рис.19 зображено локалізовану діаграму, що активно використовувалася для розробки станів

гравця. У зв’язку з тим, що українська мова не підтримує нотації скорочень, що зазвичай зустрічаються в мовах програмування було додано блок термінів та скорочень.

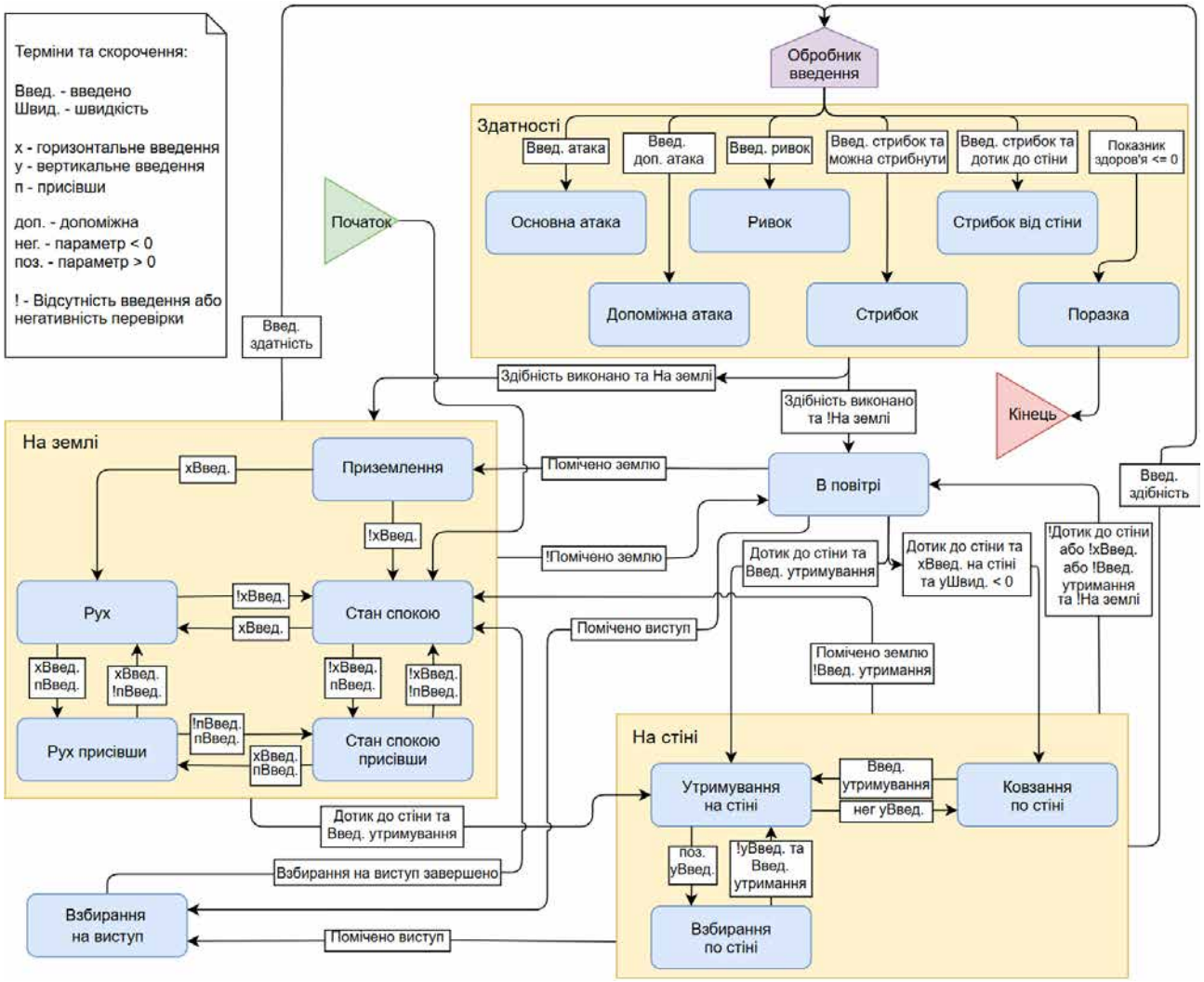


Рис.19 Діаграма станів гравця

3.2 Підхід до формування персонажів

При розробці оптимального методу для створення персонажів було вирішено скористатися базовим та основоположним принципом щодо створення об’єктів в Unity, а саме модульності, що досягається за допомогою компонентів. Ідея полягає у формування так званого «Ядра» для персонажа, що описує доступний функціонал. Ядро є дочірнім об’єктом для основного. В свою чергу його нащадки це компоненти ядра. Кожен з них відповідає за певну механіку.

Складаються здебільшого з об’єкта та однойменного скрипта. Пропоную розглянути більш детально принцип роботи на прикладі реалізованого контролеру користувача.

Ієрархія об’єктів гравця:

1. Гравець. Основний об’єкт. Містить в собі наступні компоненти:
    - Розташування. Має інформацію про масштаб, кут нахилу та місцезнаходження відносно початку координат.
    - Введення гравця. Містить карту обробки значень для пристрою введення (рис.5).
    - Однойменний скрипт. Ініціалізує всі стани патерну машини кінцевих автоматів та приймає сховище даних для визначення параметрів при їх відпрацюванні.
    - Рендер спрайту. Зображує гравця на сцені та визначає шар на якому він буде знаходитись.
    - Фізичне тіло. Прораховує фізику взаємодії з іншими тілами та силу гравітації.
    - Колайдер. Зазначає область, що доступна до взаємодії.
    - Менеджер збережень.
    - Перемикач карт введення. Основне призначення в блокуванні однієї з карт введення при переході між введенням для контролеру гравця та UI. для уникнення непередбачуваних результатів.

1.1. Ядро. Базовий для всіх компонентів:

- Однойменний скрипт. Описує логіку додання та отримання будь якого типу компонентів з допомогою дженериків за умови його наявності на об’єкті.

1.1.1. Пересування. Містить усі методи, що пов’язані з переміщенням.

1.1.2. Відчуття зіткнення. Містить дані про оброку зіткнень зі землею, стіною, урвищем, та стелею. Також визначає шар що буде сприйматися як земля.

1.1.3. Бойовий сегмент. Відповідає за визначення зони в якій атаки по гравцю будуть вважатися успішними. Також визначає правила для отримання шкоди та відкидування.

1.1.4. Статистика показників. Обраховує дані пов’язані зі здоров’ям, приголомшенням, балам за знешкодження та штраф отримання шкоди. Важливо уточнити, що в контексті гравця релевантною є лише перша характеристика, іншими він просто не користується, проте для противника важливими є усі.

1.1.5. Менеджер частинок.

1.1.6. Поразка. Виконує дії після нейтралізації персонажа, такі як оновлення балів оцінювання та генерація частинок.

1.1.7. Шкала здоров’я (див. підрозділ 3.5).

1.1.8. Збереження(див. підрозділ 3.6).

1.1.9. Інвентар (див. підрозділ 3.5).

1.1.10. Взаємодія з предметами (див. підрозділ 3.5).

1.1.11. Зміна озброєння (див. підрозділ 3.5).

1.1.12. Викладання озброєння (див. підрозділ 3.5).

2. Основне озброєння (див. підрозділ 3.3).
3. Допоміжне озброєння (див. підрозділ 3.3).
4. Індикаторна стрілка. З’являється лише під час сповільнення часу спричиненого ривком, та вказує напрямок руху.

3.3 Формування озброєння

3.3.1 Підхід до розробки екіпіровки. Для реалізації завдання було розроблено комплексний підхід, що охоплює вирішення наступних питань:

1. Що таке зброя та які її типи реалізовано?
2. Як екіпірована зброя узгоджується з гравцем?

Розглядаючи перше питання потрібно зазначити що сама по собі одиниця зброї це об’єкт сформований на основі екземпляру сховища даних. Проте на відміну від персонажів з їх даними станів кожен з яких має власний тип, вся зброя використовує єдиний спільний тип. На даний момент реалізовано скриптовані об’єкти для меча, що взаємодіє з противниками на короткій дистанції та лук, що випускає стріли для нанесення шкоди на дистанції. На перший погляд це два несумісних між собою концепта що не можуть бути реалізованими в межах об’єкту одного класу. Насправді так і є, якщо користуватися стандартними принципами. Проте нічого не заважає звернутися до уже знайомого компонентно-орієнтованого підходу Unity та реалізувати його в середині скриптованого об’єкту. Все що для цього потрібно це використати дженерики та наслідування. Ігровий рушій передбачає зміну інтерфейсу редактору під свої потреби через код скриптів. Так було досягнуто послідовності за якої при написанні нового компоненту зброї шляхом наслідування від базового класу, що відповідає за логіку роботи будь якого компоненту, автоматично з’являється відповідна кнопка в редакторі для його додання до сховища даних екіпіровки. Хоч всі потрібні підписи підтягуються автоматично, проте були також реалізовані кнопки для їх примусового оновлення у разі виникнення помилок пов’язаних з невідповідностями. Таким чином було створено своєрідний імпровізований інтерфейс для роботи з екземплярами сховища даних. На рис.20 зображено його вигляд у редакторі Unity.

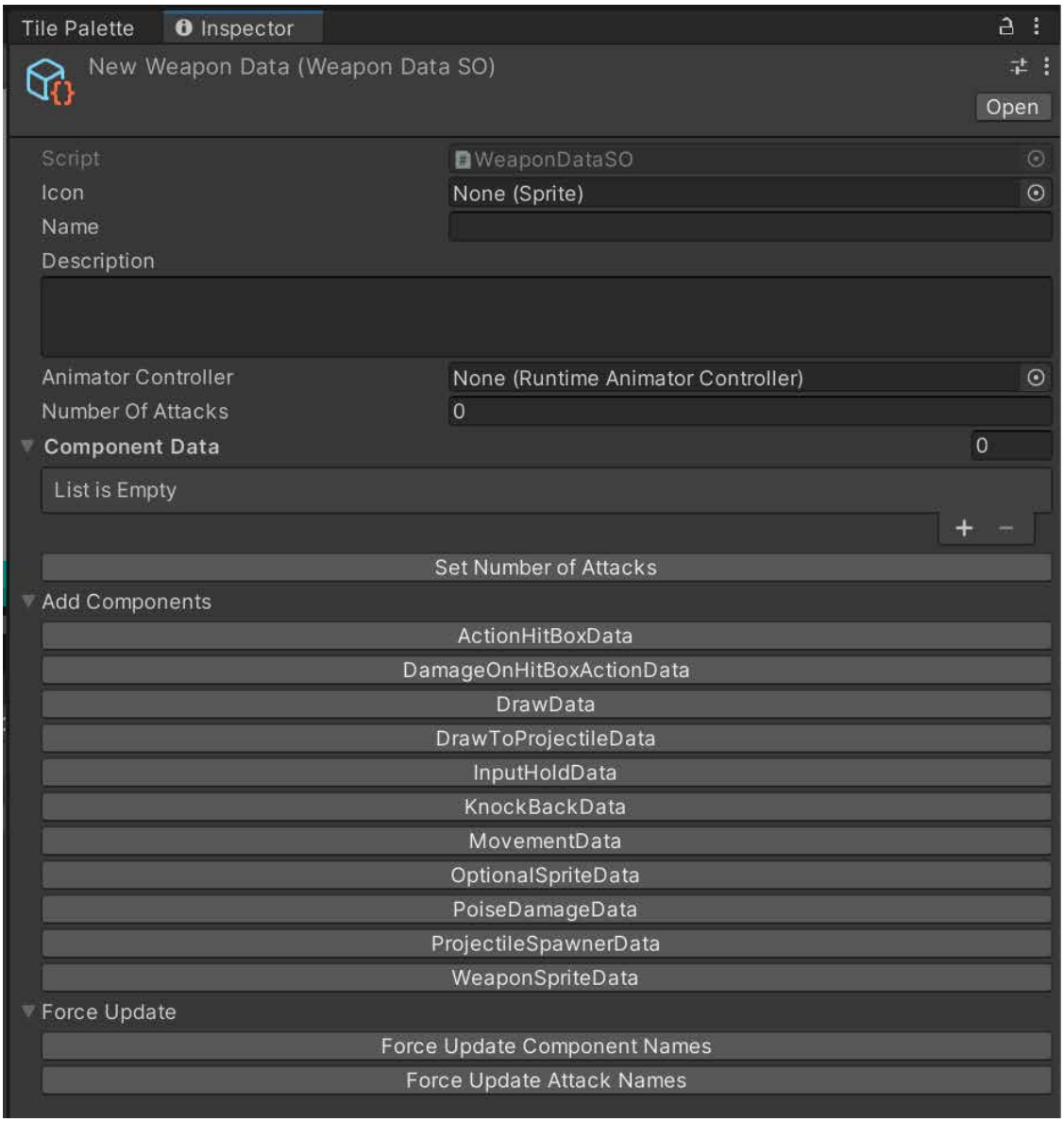


Рис.20 Редактор створення екземплярів озброєння

Вирішуючи друге питання було прийнято рішення розділити спрайти атаки гравця та зброї. Сховище екіпіровки має обов'язкове поле в якому вказується аніматор гравця в якому визначені всі анімації персонажа без відповідного типу зброї. В противагу цього існує в цьому ж сховищі створюється компонент пов'язаний зі спрайтами для анімацій та фаза атак на яких розташовано лише предмет без гравця. Важливо також зазначити що аніматор має спрайти персонажа саме за типом зброї, а не її конкретним екземпляром. Тобто при поєднанні цих двох груп зображень можна отримати результат за якого однакові анімації гравця використовуються для різних мечів або луків. Сформований скриптований об'єкт розміщується при підборі озброєння в

інвентарі, що є компонентом ядра гравця. Для його використання використовуються об’єкти основного та допоміжного озброєння. Вони мають однакову структуру та призначення. Як було визначено перший об’єкт відповідає за першу комірку інвентаря та активується при натисненні на ліву кнопку миші, інший за другу комірку та відповідно пов’язана з правою кнопкою миші. Ці значення є релевантними лише для схеми введення «клавіатура-миша».

Структура об’єкту озброєння:

1. Базовий об’єкт.
    - Відповідний скрипт, що визначає основні положення того чим є зброя та які методи вона використовує. Також тут вказується час до скидання лічильника порядку атак. Наприклад якщо це значення дорівнює п’яти, а комбо містить три атаки, то після запуску першої є вікно в розмірі п’яти секунд під час якого у разі повторної активації відпрацьовується друга атака в серії. При закінченні таймеру, або виконання всього комбо наступна атака почнеться з першої.
    - Скрипт генерації озброєння. Автоматично додає до об’єкту всі компоненти озброєння, що містить сховище даних на відповідній комірці інвентаря при його екіпіруванні.

    1.1. Основа.

    - Містить аніматор гравця, отриманий від скриптованого об’єкту озброєння.
    - Компонент рендерингу спрайтів, який зображує персонажа на сцені.
    - Скрипт з тригерами подій, що активуються під час відпрацювання анімацій.

    1.1.1. Додаткові спрайти. Використовується для генерації додаткових зображень, що потрібні для таких типів

озброєнь як луки. Допоміжними в цьому випадку є зображення стріл.

1.2. Зображення зброї. Спрайти отримуються з сховища даних та заміщуються одночасно з анімаціями основи персонажа через активацію тригерів.

Додаток В містить зображення налаштованих скриптованих об’єктів меча та лука, а також шаблону для стріли.

3.3.2 Реалізація меча. Всі екземпляри цього типу мають короткий радіус атаки та швидкі анімації, що робить його сконцентрованим на сутичках на близькій дистанції. В базовій варіації складається зі шести компонентів.

Поля по замовченню:

- Іконка. Визначає вигляд предмета в інвентарі, меню вибору та на землі.
- Ім’я. Потрібне для підпису в меню вибору екіпіровки.
- Опис. Аналогічно розміщується в меню.
- Аніматор контролеру основи гравця.
- Кількість атак.

Компоненти:

- Спрайти атак. Вказується кількість атак після чого обирається фаза та призначаються спрайти. Значення обов’язкового поля кількості атак має вищий пріоритет якщо в ньому вказано менше значення, то використовуються лише та кількість атак. Для цього озброєння використовується лише фаза дії.
- Бойове переміщення. Визначає напрямок та швидкість руху при активації кожної з атак.
- Хітбокс дії. Визначає шари та межі взаємодія з якими вважається за влучання в ціль.
- Шкода за влучання. Призначається для кожної атаки окремо.
- Відштовхування. Визначає кут та силу відштовхування.

- Шкода стійкості. Означає скільки шкоди нанесе кожна успішна атака окремо по шкалі приголомшення противника.

3.3.3 Реалізація лука. Складається зі снаряду стріли та безпосередньо лука.

Компоненти лука:

- Спрайти атак. В ситуації з луком використовуються три різні фази. Перша відповідає за натягування тетиви. Друга за утримання в статичній позиції. Третя за відпускання стріли.
- Утримання. Не містить даних проте вказує, що запуск стріли відбувається після відпускання кнопки, не по її натисненню.
- Додатковий спрайт. Передає своє значення у однойменний об’єкт. У даному випадку це зображення стріли.
- Бойове переміщення. Для луків переміщення дорівнює нулю. Наявність зумовлена виправленням помилки при якій якщо під час руху виконати постріл, то персонаж сам почне пересуватися в останньому напрямку, аж поки викликана дія не завершиться.
- Породжувач снарядів. Визначає з якої точки та напрямку відносно гравця будуть з’являтися стріли, їх шаблон, шкоду здоров’ю та стійкості, значення відштовхування, спрайт.
- Дані кривої. Відповідає за оцінку кривої за якою повинен летіти снаряд, після припинення введення транслює це значення.
- Використання даних кривої снарядом. Працює як з компонентами «Дані кривої», так і «Породжувач снарядів». Прослуховує подію оцінки і створення снаряду. Під час оцінки це значення зберігається, коли снаряд породжується, то упаковується та надсилається, щоб будь-який компонент міг його використовувати.

Сховище даних для снарядів реалізовано подібно до зброї, з використанням власних компонентів.

Компоненти стріли:

- Однойменний скрипт.
- Пересування. Компонент руху снаряда відповідає за застосування швидкості до снаряда. Швидкість можна застосувати до снаряда лише один раз.
- Затримка гравітації. Встановлює початкову гравітацію снарядів до нуля. Після того, як певну відстань пройдено значення змінюється. Це надає снарядам ефект, подібний до стріл у DeadCells, які рухаються прямо на деяку відстань, а потім починають падати.
- Обертання снаряду. Обертає поточний об’єкт таким чином, щоб кінець стріли вказував у тому самому напрямку, що й вектор швидкості.
- Хітбок. Описує область та шари доступні до взаємодії.
- Застрягання. Цей компонент відповідає за те, щоб снаряд застряг у певному шарі на основі того, що виявляє хітбокс.
- Пул об’єктів снарядів (див. підрозділ 3.4).
- Компоненти «Шкода», «Відкидування», «Шкода стійкості». Відповідають за використання інформації, наданої компонентом «хітбокс», для пошкодження будь-яких сутностей, які знаходяться на відповідному шарі.

3.4 Пул об’єктів

Objective Pool – також відомий як пул об’єктів, представляє з себе шаблон проектування, який активно використовується для ефективного повторного використання існуючих об’єктів [10]. Основною проблемою яку вирішує описаний патерн є те, що для створення нових об’єктів використовується велика кількість ресурсів, що негативно впливає на продуктивність програмного забезпечення. Виходячи з цього потрібно максимально уникати постійної ініціалізації нових об’єктів, як наслідок знищення також є небажаним.

Закономірно постає питання: «То що ж робити, якщо все під забороною, а використовувати об’єкти все ще потрібно?». Шаблон проектування надає рішення цієї дилеми. Для цього потрібно лише більш уважно поглянути на саме питання. Ключовим словом є «використовувати», а не «створювати», тому і відповідь є доволі простою – навіщо створювати нові об’єкти, якщо можна просто змінити параметри уже існуючих під актуальну ситуацію. Щоб реалізувати цю ідею використовується заздалегідь створена колекція об’єктів, яку називають пулом, елементи якої за потреби тимчасово активуються або деактивуються.

Принцип можна узагальнити до наступних пунктів:

1. Ініціалізація пула – створюється фіксована кількість об’єктів, які зберігають у черзі або списку.
2. Отримання об’єкта – коли виникає потреба у новому об’єкті, пул бере перший з неактивних та виконує його активацію, тим самим підготовивши його до роботи, без виконання при цьому повторного створення.
3. Повернення об’єкта – після відпрацювання поставленої на об’єкт задачі, об’єкт деактивується та повертається до пулу, що з одного боку імітує його знищення, проте залишає можливість до повторного використання.

Описаний патерн в межах проекту було використано для виконання наступних задач:

- Залишковий слід персонажа. У рамках реалізації патерну машини кінцевих автоматів для гравця було розроблено стан ривка, під час якого швидкість гравця кратно зростає на короткий проміжок часу. Для кращої передачі швидкості зберігаючи при цьому цільність зображення було прийнято рішення у створенні залишкового сліду. Цей слід складається зі зафіксованих спрайтів гравця у пройдених під час виконання здібності частинах простору. Перший спрайт береться з пулу в момент активації ривку. Всі наступні через

однакові проміжки часу. Коли час виконання здібності вичерпується всі об’єкти по черзі повертаються назад до колекції починаючи з першого. На рис.21 зображено виконання ривка з відпрацюванням залишкового сліду.

- Снаряди. Одним з об’єктів, що часто може генерувати як гравець так і певні види ворогів є снаряди, а саме в цій реалізації їх роль виконують стріли.

Рис.21 Залишковий слід

3.5 Інтерфейс користувача

Важливою складовою успішної гри є зручний інтерфейс. Він забезпечує взаємодію між користувачем та грою. В Unity за його створення відповідає спеціальна UI-система. Вона дозволяє формувати меню, його кнопки, текстові поля, індикатори здоров’я, та будь-які інші типові елементи, що сприяють ефективному залученні клієнта в ігровий процес.

Основні елементи інтерфейсу:

- Полотно. Також відоме як Canvas та є базовим контейнером в якому знаходяться всі UI-елементи.
- Текст. Відповідає за відображення текстової інформації. Замість встановленого по замовченню UI.Text використовується TextMeshPro, описаний в пункті 2.2.4.
- Зображення. Використовується для показу графічних елементів.
- Кнопка. Інтерактивний об’єкт з можливістю виклику функцій при натичненні.
- Слайдер. Повзунок для вибору значень.
- Чекбокс. Перемикач, що приймає два значення, а саме обрано та необрано.
- Поле введення. Представляє з себе область в межах якої можна вводити текстові значення.
- Панель. Контейнер для групування UI-елементів.

Через компонентно-орієнтованість рушія щодо створення об’єктів, всі елементи в першу чергу це ігрові об’єкти що мають відповідні компоненти. Тобто і формування будь-якого інтерфейсу є аналогічним до створення персонажів. Тому для нього доступні такі ж особливості, як от, наприклад, створення шаблонів або використання одразу декількох компонентів. Так розташування UI-елементів задається за допомогою компоненту RectTransform, який відрізняється від звичайного Transform у тому, що дозволяє налаштовувати позицію і розмір прив’язуючись до меж екрану.

Головне меню має 3 кнопки: «Грати», «Налаштування», «Вихід», як і зображено на рис.22. При натисненні на кнопку «Грати» запускається рівень. Кнопка «Налаштування» відкриває вікно, у якому можна змінити гучність та повернутися до меню, рис.23. «Вихід» завершує роботу програми.

Рис.22 Головне меню

Рис.23 Меню налаштувань

Інтерфейс гравця складається з двох частин. Перша - це показник здоров’я, що розміщується у лівому верхньому куті. Має вигляд заповненої шкали, при цьому колір вмісту варіюється відповідно до відсотку її заповнення. Всього має три стадії: від нуля до 33-ох відсотків червоний колір, далі до 66-и відсотків жовтий, і закономірно зелений у всіх інших випадках. Цю закономірність продемонстровано на рис.24.

Рис.24 Шкала здоров’я з усіма стадіями

Друга частина - це інвентар гравця. Розташовується в нижньому лівому куті. Містить в собі екземпляри зрбої, що підібрав користувач під час гри рис.26. По замовченню на початку гри комірки пусті рис.25.

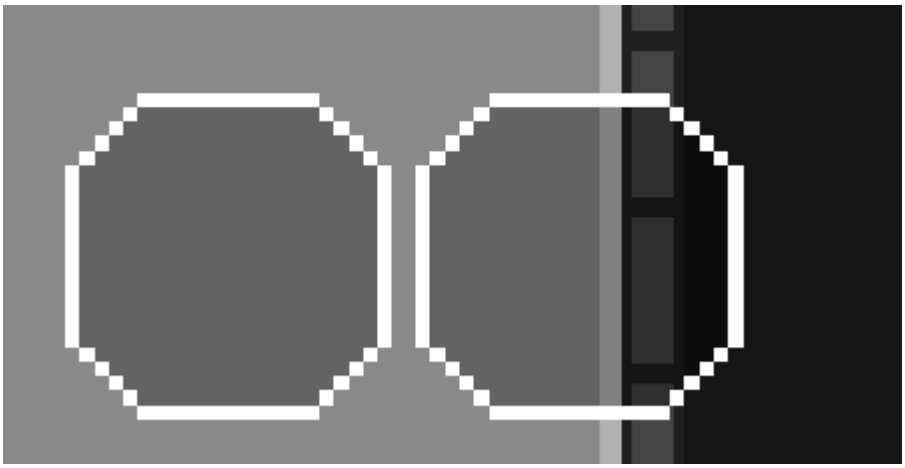


Рис.25 Пусті комірки інвентаря

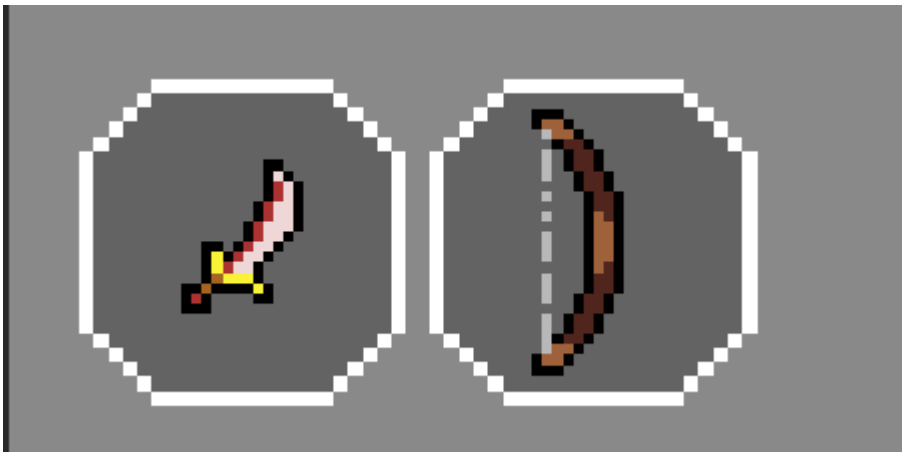


Рис.26 Комірки з екіпійованою зброєю

Крім перелічених існують ще інтерфейси вспливаючих меню. Так якщо спробувати екпіювати зброю при переповненому інвентарі, то з’являється меню заміни зброї. Воно надає короткий зміст, щодо особливостей екіперовки. Зверху показує предмет що буде додано до інвентаря, знизу перелічуються всі наявні на даний момент у гравця одиниці зброї, з яких потрібно обрати той що буде викинуто. На рис. 27 зображено цей процес. Блок «меча» є текстовим, а ось «скімітар» та «лук» представляють із себе кнопки. Також у лівих верхніх кутах кожного блока містяться відповідні піктограми.

Рис.27 Меню заміни озброєння

Наступні вспливаючі меню це вікна паузи та поразки. Мають схожий функціонал. При їх виклику гра зупиняється, у тому числі всі вороги та таймер проходження. В своєму складі мають спільні кнопки переходу в головне меню та виходу зі застосунку.

Відмінним є те, що пауза викликається по натисненню на клавішу «Esc» продовжити гру можна натиснувши на відповідну кнопку, або повторно натиснувши на «Esc» рис.28.

Інше меню активується при поразці. Тобто коли здоров’я гравця вичерпається. В грі передбачена ситуація коли гравець провалюється за межі ігрового рівня. У цьому разі при досягненні певної висоти програма сама викличе вікно поразки, за допомогою якого можна перезавантажити поточний рівень по натисненню на кнопку «Повторити» рис.29.

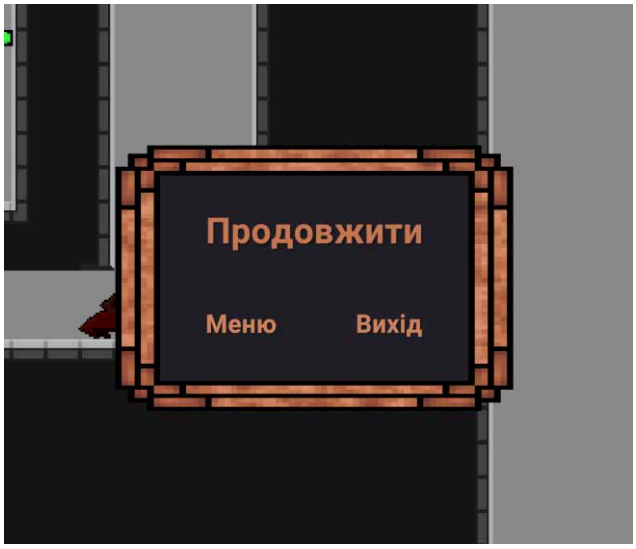


Рис.28 Меню паузи

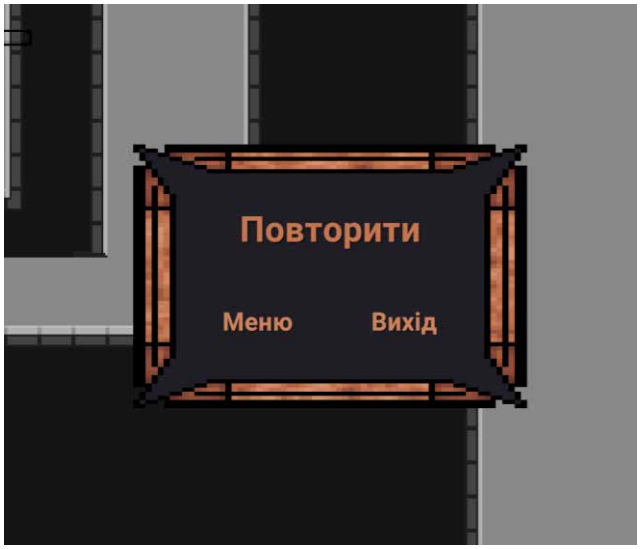


Рис.29 Меню поразки

По завершенню проходження рівня гра покаже результати у відповідному меню (рис.30).

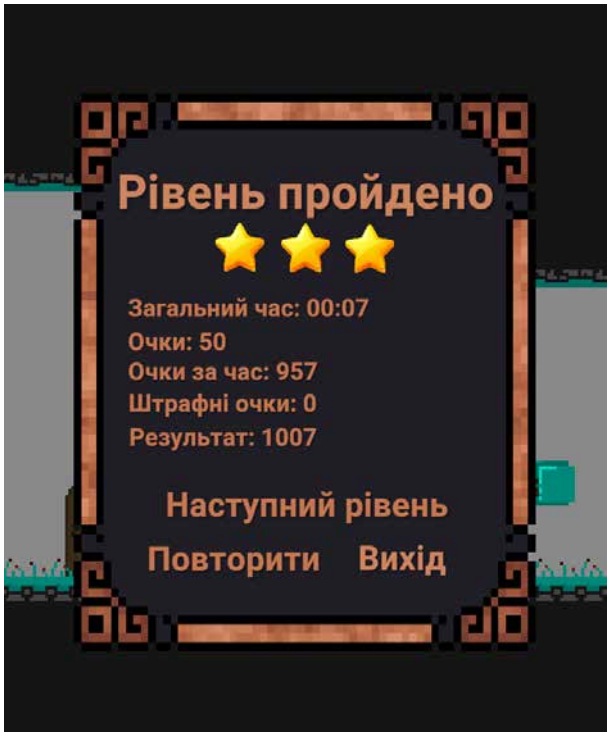


Рис.30 Меню успішного завершення рівня

3.6 Збереження

Так як представлена система не передбачає наявність таблиць на основі яких можна формувати звітно-статистичну інформацію то для цього завдання було створено механіку збереження гри.

Збереження поділяються на 2 типи:

- Сесії поточного рівня.
- Результатів проходження рівня.

Збереження рівня виконується в ручному режимі по натисненню на кнопку «F5» (рис.30). Це збереження можна завантажити запустивши відповідний рівень після чого натиснувши на «F9». Всі дані з фалу будуть імпортовані до гри. На рис.32 видно що в порівнянні з рис.31 гравець виконав переміщення та отримав шкоду, після завантаження на рис.33 його стан став аналогічним до збереженого. Для кожного рівня доступним є лише одне збереження. Спроба сформувати нове призведе до його перезапису.

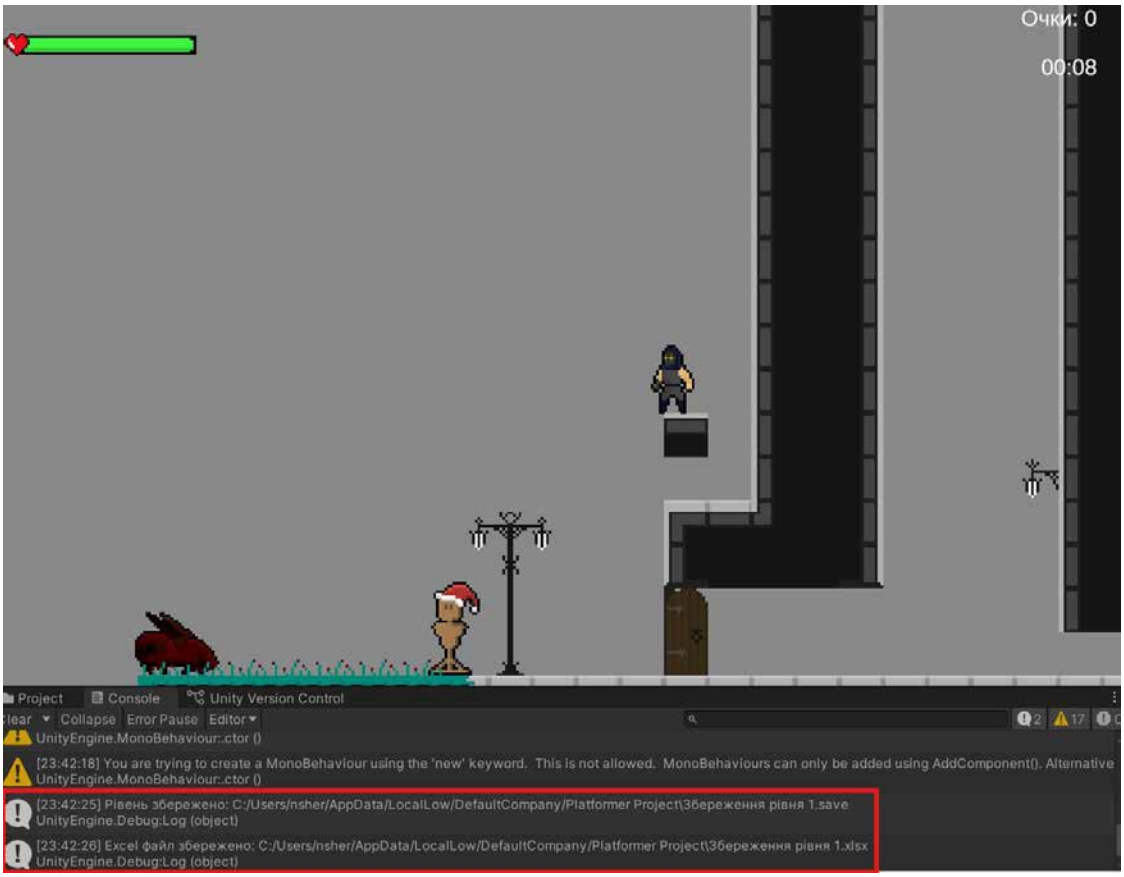


Рис.31 Збереження гри

Рис.32 Зміна параметрів після збереження

Рис.33 Завантаження

Збереження результатів виконується автоматично по завершенню рівня. Таке збереження не можна завантажити. Для нього розроблено підхід до оцінювання досягнень. Гравець отримує очки за знешкодження противників, та втрачає при отриманні шкоди. Винагороду та штрафи можна назначати для кожного типу персонажа окремо за допомогою відповідних налаштувань в скриптованому об’єкті. Бонусні очки нараховуються за швидкість проходження рівня. Вони розраховуються за принципом постійного зменшення, поки їх кількість через певний час не стане дорівнювати нулю. Підсумувавши всі джерела отримання очків програма призначає проходженню рівня ранг. Для гравця представлений у вигляді трьох зірок, що мають заповнену та незаповнену варіацію. Програмно ж оцінювання відбувається у діапазоні від нуля до трьох, де вище чим вище значення тим краще результат.

Збереження представлене у форматі json файлу з розширенням «.save» (рис.34, 36).

Звіт створюється автоматично при створенні збереження та має формат Exel таблиці з розширенням «.xlsx» (рис.35, 37). За допомогою словника дані збережень локалізуються українською мовою та переносяться в таблиці.

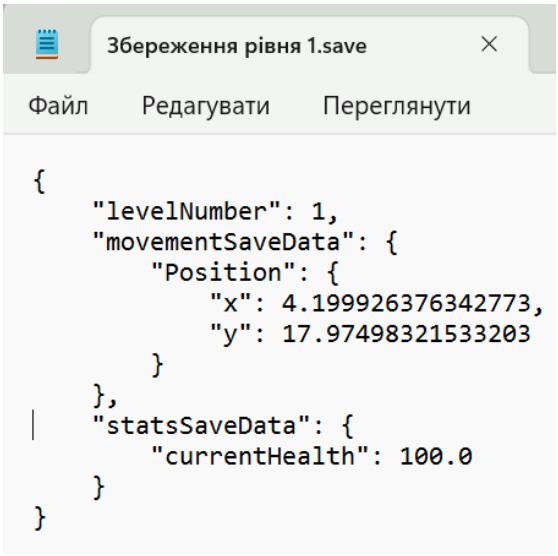


Рис.34 Ручне збереження .save

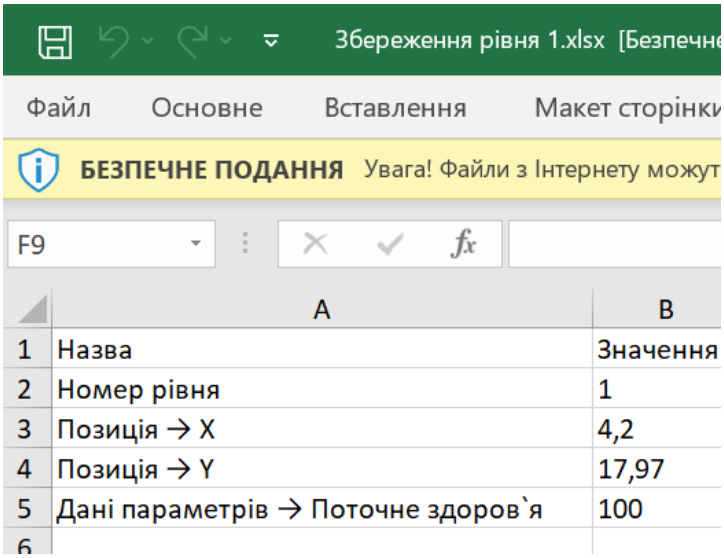


Рис.35 Звіт таблиця .xlsx

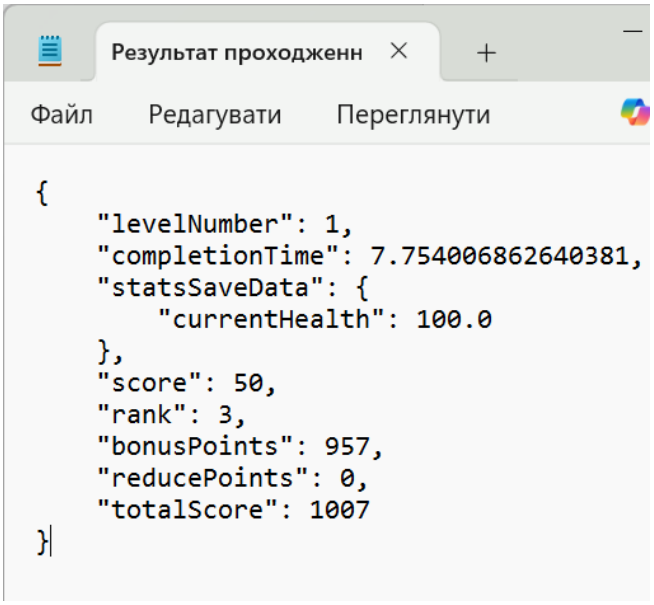


Рис.36 Збереження проходження .save

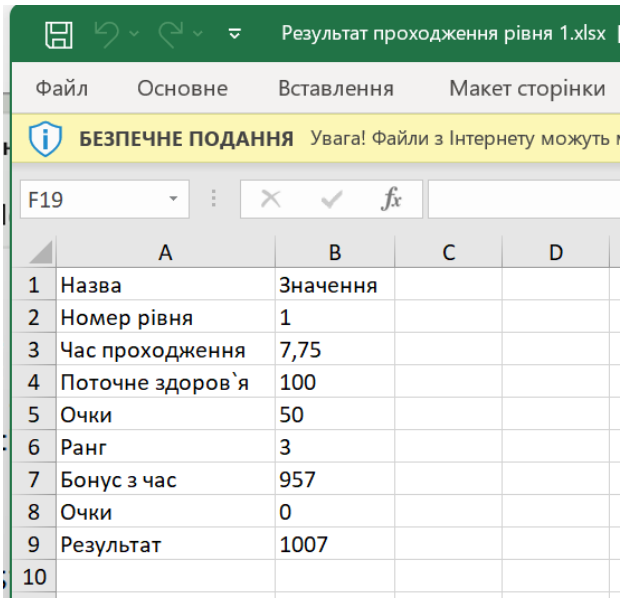


Рис.37 Звіт таблиця .xlsx

Файли зберігаються на персональному комп’ютері користувача за наступним шляхом: C:\Users\<user>\AppData\LocalLow\<company name> (рис.38)

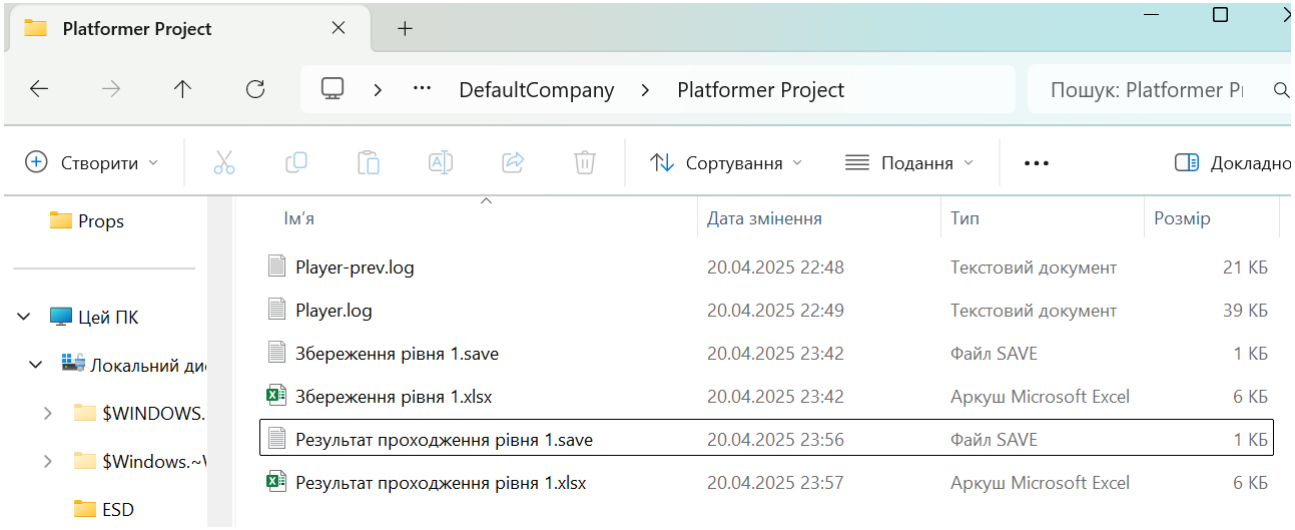


Рис.38 Вміст папки зі збереженнями

3.7 Висновки до третього розділу

Метою третього розділу стала розробка системи. Було розглянуто та реалізовано основні алгоритми, що знадобилися для розробки двовимірної гри в жанрі платформер.

Ключовим патерном з позиції поведінки та взаємодії персонажів є машина кінцевих автоматів. Вона дозволяє розділити логіку та обмежити функціонал конкретного персонажа в межах окремих станів. Умови переходу між якими описані всередині поточного стану. Це робить поведінку прогнозованою та дозволяє будувати її схеми. Найкращим прикладом реалізації є базовий компонент рушія Unity аніматор. Його призначення керувати анімаціями на основі описаних розробником правил.

Оптимізаційним шаблоном проектування використаним в проекті є пул об’єктів. Його призначення зберігати в собі неактивні об’єкти та надавати доступ до них за потреби, замість створення нового. Це значно зменшує навантаження на систему, адже активація та зміна параметрів уже існуючого об’єкту набагато дешевша в плані ресурсів за ініціалізацію нового. Пул використано для залишкового зображення гравця під час ривку та для генерації стріл луком.

Для взаємодії з грою та надання гравцю додаткової інформації було сформовано декілька типів інтерфейсів. Перший – інформаційний. До його представників входить інвентар та шкала здоров’я. Другий – інтерактивний. Представляє з себе вспливаючі меню які виконують різні ролі: пауза, перезапуск при поразці, навігація після проходження рівня, заміна озброєння за умови переповнення інвентаря.

В умовах відсутності бази даних було реалізовано механіку збережень для отримання звітності по поточній сесії та результатам проходження конкретного рівня. Збереження створюються локально на пристрої.

РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

У процесі розробки двовимірної гри було проведено комплексне тестування для перевірки коректності роботи функціоналу, зручності використання та відповідності поставленим вимогам. Тестування проводилося на всіх етапах створення гри, що дозволило вчасно виявляти та усувати помилки.

Сценарії функціонального тестування:

1. Підбір зброї.
2. Виконання анімації атаки та коректна зміна характеристик у противника.
3. Отримання шкоди та вірне узгодження з UI.
4. Збереження параметрів персонажа.

Перший сценарій

Вхідні дані:

- Пустий інвентар.
- Об’єкт меча лежить на підлозі рівня.

Алгоритм тестування:

1. Увійти в радіус підбору. Маркером для гравця є погойдування зброї в повітрі.
2. Натиснути на кнопку взаємодії «E».

Очікуваний результат:

- Зникнення зброї зі землі.
- Поява відповідного екіпірування в інтерфейсі інвентаря в лівому нижньому куті.
- Перша комірка компоненту інвентаря гравця повинна отримати посилання на об’єкт сховища даних підібраного меча.

- На основі даних зі сховища об’єкт основної екіпіровки самостійно додає потрібні компоненти для формування зброї.

Результат тестування. Успішно.

Другий сценарій

Вхідні дані:

- Екіпіровано меч як основну зброю.
- Показник шкоди на першій та другій атаці дорівнює 10, на третій 20.
- Максимальне здоров’я противника «Монстр» дорівнює 50

Алгоритм тестування:

1. Підійти до монстра на дистанцію ураження меча.
2. Виконати три атаки. Після кожної дії фіксувати зміну показника здоров’я противника.
3. Візуально перевірити коректність виведення анімацій.

Очікуваний результат:

- Після першого попадання здоров’я монстра дорівнює 40.
- Після другого 30.
- Після третього 10, ворог підкидається в повітря.

Результат тестування. Успішно.

Третій сценарій

Вхідні дані:

- Максимальне здоров’я гравця дорівнює 100.
- Шкала здоров’я в інтерфейсі повністю заповнена та окрашена в зелений колір.
- Шкода від атаки монстра дорівнює 40.

Алгоритм тестування:

1. Підійти до монстра на дистанцію його ураження.
2. Отримати два влучання.

Очікуваний результат:

- Після першого атаки здоров’я гравця дорівнює 60. Шкала заповнена трохи більше за половину, її колір змінився на жовтий.
- Після другого показник дорівнює 20. Шкала заповнена на одну п’яту, її колір червоний.

Результат тестування. Успішно.

Четвертий сценарій

Вхідні дані:

- Здоров’я гравця дорівнює 20.
- Гравець знаходитися в точці x =2.6; y = 19.975

Алгоритм тестування:

1. Створити збереження натиснувши на кнопку «F5»
2. Порівняти отримані результати

Очікуваний результат:

- У створеному звіті в формтаті Exel таблиці повинні бути наявні чотипри значення. Перше «Номер рівня = 1». Друге «Позиція x –> 2.6. Третє «Позиція y – > = 19.98». Четверте «Дані параметрів –> поточне здроров’я = 20». Точність значень при цьому повинна бути до однієї сотої.

Результат тестування. Успішно.

4.2 Вимоги до апаратного та програмного забезпечення

Мінімальні вимоги:

- Операційна система: Windows 7 SP1+
- Процесор: x86_64 або ARM64, 2.6 ГГц двоядерний процесор
- Оперативна пам'ять: 4 ГБ RAM
- Графіка: GPU, що підтримує OpenGL 3.2 або DirectX 11
- Місце на диску: 1 ГБ
- Роздільна здатність екрана: 1280 x 800

Рекомендовані вимоги:

- Операційна система: Windows 10 64-bit
- Процесор: Quad-core Intel або AMD, 3.0 ГГц
- Оперативна пам'ять: 8 ГБ RAM
- Графіка: GPU, що підтримує DirectX 12 / Vulkan
- Місце на диску: 5 ГБ
- Роздільна здатність екрана: 1920 x 1080 або більше

Для запуску ігор, створених за допомогою рушія Unity не потрібні додаткові утиліти, вони працюють як звичайні самостійні програми. На операційній системі Windows для відпрацювання C# коду використовується .NET Framework, проте починаючи з версії Windows 10 він є вбудованим по замовченню.

4.3 Склад інсталяційного пакету

Інсталяційний пакет також не має унікальних компонентів. Рушій після виконання операції білду сам запаковує всі потрібні файли в обрану папку(рис. 39), яку можна заархівувати для передачі між пристроями. В контексті монетизації програмного продукту існують спеціалізовані магазини, які беруть на себе роль поширення та інсталяції застосунку.

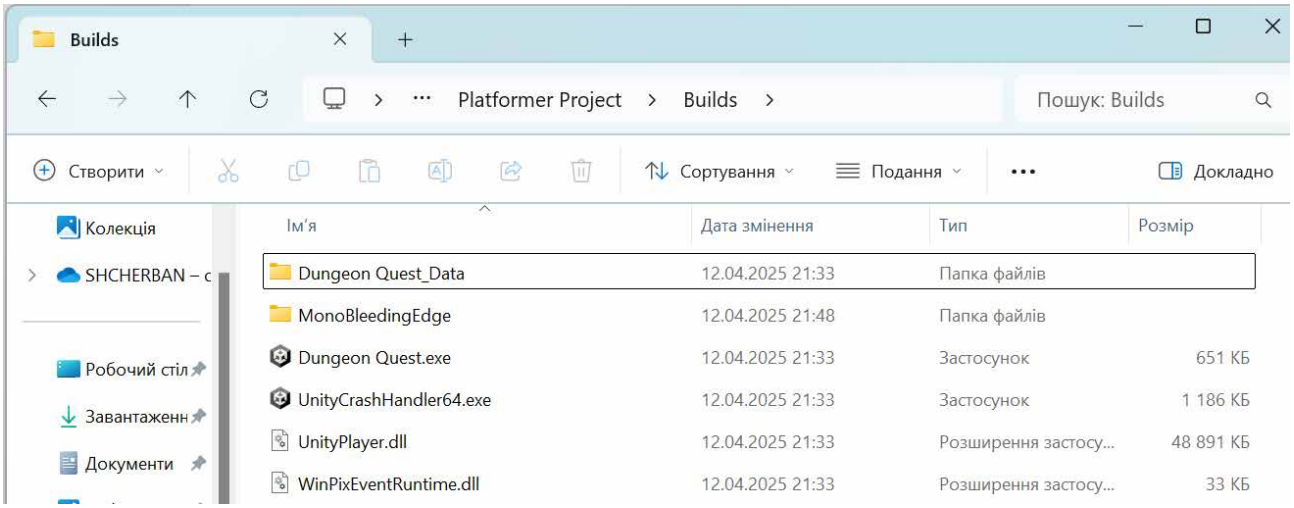


Рис. 39 Вміст папки інстальованого програмного продукту

4.4 Висновок по четвертому розділу

В межах цього розділу було проведено тестування та впровадження розробленої інформаційної системи. У ході тестування результат описаних сценаріїв виявився позитивним. Додаток Д містить всі необхідні перевірочні скріншоти, що було зроблено у ході відпрацювання сценаріїв. Розроблений застосунок має низькі вимоги до апаратного забезпечення, що забезпечує його доступність для широких мас. Програмна сторона також є дружньою для користувача, адже всі обов’язкові утиліти для запуску гри вже передвстановлені операційною системою Windows.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розглянуто ігрову індустрію, визначено її взаємодію з іншими сферами. На основі дослідження існуючих рішень було розроблено підхід для створення двовимірних ігор в жанрі платформер за допомогою рушія Unity. Він дозволяє розробникам уникати проблем пов’язаних з масштабованістю та структурованістю проекту. Головними натхненниками стали відеоігри під назвами Hollow Knight та Dead cells. Від першої взято принципи роботи контролеру гравця і поведінки ворогів та реалізовано їх за допомогою патерну машини кінцевих автоматів. Другий в свою чергу показав приклад простих та логічних бойових механік в контексті двовимірних світів. Це стало базисом для формування підсистеми модульного конструювання озброєння для персонажа гравця.

Описані підходи автоматизують процес розробки основних механік гри та забезпечують достатньо великий рівень варіативності щодо написання різноманітних типів озброєння та ворогів що використовують унікальну поведінку.

Частково піднята тема мультиплатформенності, а саме введення користувача представлено у двох варіаціях: класичне з використанням клавіатури та миші і консольне через геймпад. Проте варто зазначити, що програмний продукт не портовано на консолі. Автоматичне визначення та застосування схеми введення це лише перший крок на шляху до цього.

Всебічно досліджено та опановано інструментарій який використовують розробники відеоігор при роботі з рушієм Unity. До головних представників можна віднести:

- Анімації, що складаються зі групи логічно пов’язаних спрайтів, які замінюють один одного з певною частотою.

- Матеріали, що використовуються як для модернизації графічного відображення, так і при розрахунку впливу фізики на об’єкт до якого вони застосовані
- Шаблони об’єктів. Ключовий інструмент при використанні об’єктів, що повинні повторюватись на сцені.
- Карти тайлів в свою чергу надають просте та потужне рішення щодо формування ігрових світів.
- Ігрові сцени, що використовуються для зберігання готових рівнів з наявними у них всіх необхідних об’єктів.

Для досягнення задовільного рівня користувацького досвіду було розроблено багатофункціональний інтерфейс, що виконує як інформативні так і інтерактивні задачі. Ще одним клієнтоорієнтованим рішенням є механіки збереження, що також є засобом отримання звітної інформації для розробника.

Виконано збірку програмного продукту та пройдено всі поставлені перед ним тести. У зв’язку з низькими програмними та апаратними вимогами застосунок може бути поширений для використання на будь якому пристрої, що керується операційною системою Windows.

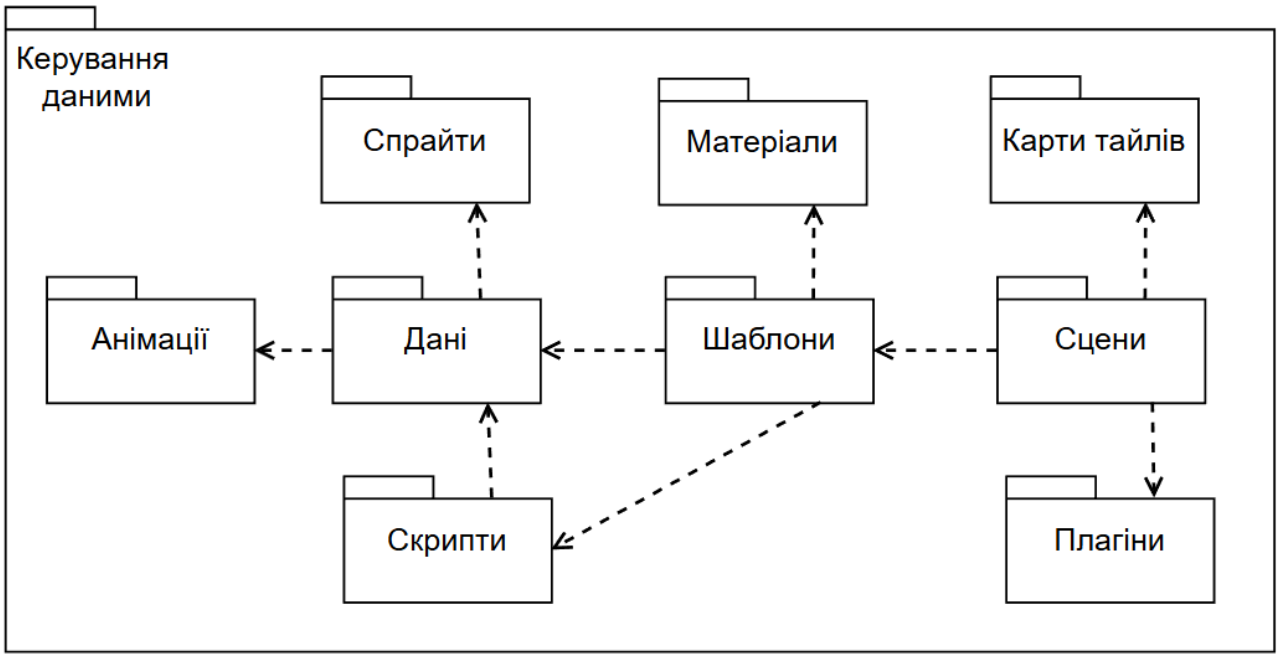
ПЕРЕЛІК ЛІТЕРАТУРИ

1. Geniuscrate. Gaming in the Classroom: How Educators Use Video Games to Teach History URL: https://www.geniuscrate.com/gaming-in-the-classroom-how-educators-use-video-games-to-teach-history?utm_source=chatgpt.com (дата звернення: 14.02.2025).
2. Relations between Video Game Engagement and Social Development URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC10604845/?utm_source=chatgpt.com (дата звернення: 05.02.2025).
3. Майклс Д. A Million People Play This Video Wargame. So Does the Pentagon URL: https://www.wsj.com/politics/national-security/a-million-people-play-this-video-wargame-so-does-the-pentagon-e6388f50?utm_source=chatgpt.com (дата звернення: 12.02.2025).
4. Games Market Reports and Forecasts URL: https://newzoo.com/games-market-reports-and-forecasts (дата звернення: 16.02.2025).
5. Team Cherry Hollow Knight URL:https://www.hollowknight.com/ (дата звернення: 24.02.2025).
6. Motion Twin Dead Cells URL: https://dead-cells.com/ (дата звернення: 27.02.2025).
7. Quick start guide | Input System | 1.0.2 URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/QuickStartGuide.html (дата звернення: 02.03.2025).
8. Unity Technologies. Tilemap URL: https://docs.unity3d.com/6000.1/Documentation/ScriptReference/Tilemaps.Tilemap.html (дата звернення: 05.03.2025).
9. Батфілд-Еддісон П., Меннінг Д., Ньюджент Т. Unity Game Development Cookbook. – Sebastopol, CA: O'Reilly Media, 2021. – 354 с.
10. Нюстрем Р. Game Programming Patterns. – USA: Genever Benning, 2014. – 354 с.

11. Хромов І. В. Графіка та анімація в Unity: навч. посіб. / І. В. Хромов. — Київ: Наука і освіта, 2021. — 240 с..
12. Марченко А. І. Unity для початківців: розробка 2D ігор: навч. посіб. / А. І. Марченко. — Київ: Діалектика, 2020. — 256 с.
13. Bardent. The Bardent Asset Pack! URL: https://bardent.itch.io/the-bardent-asset-pack (дата звернення: 25.03.2025).

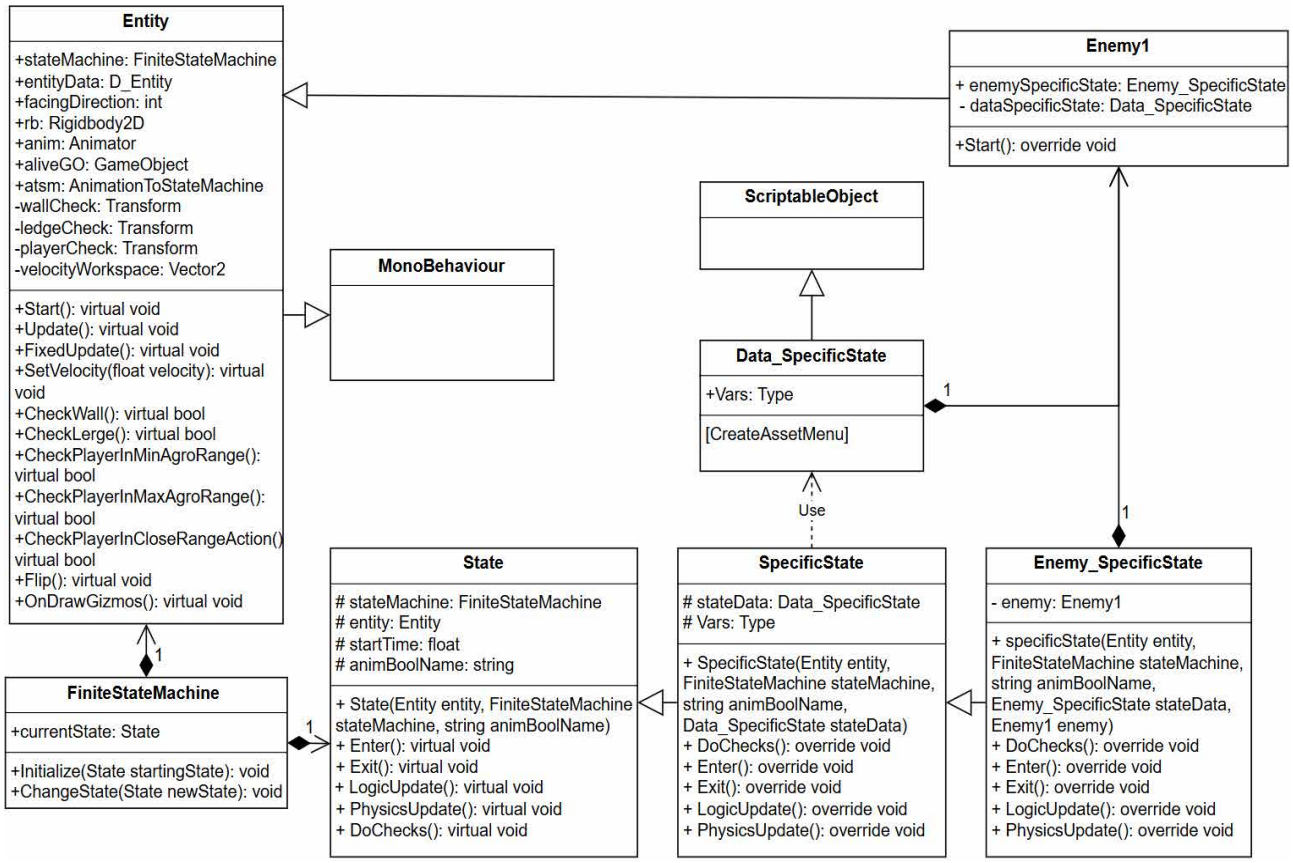
Додаток А

Організаційна структура програмного забезпечення



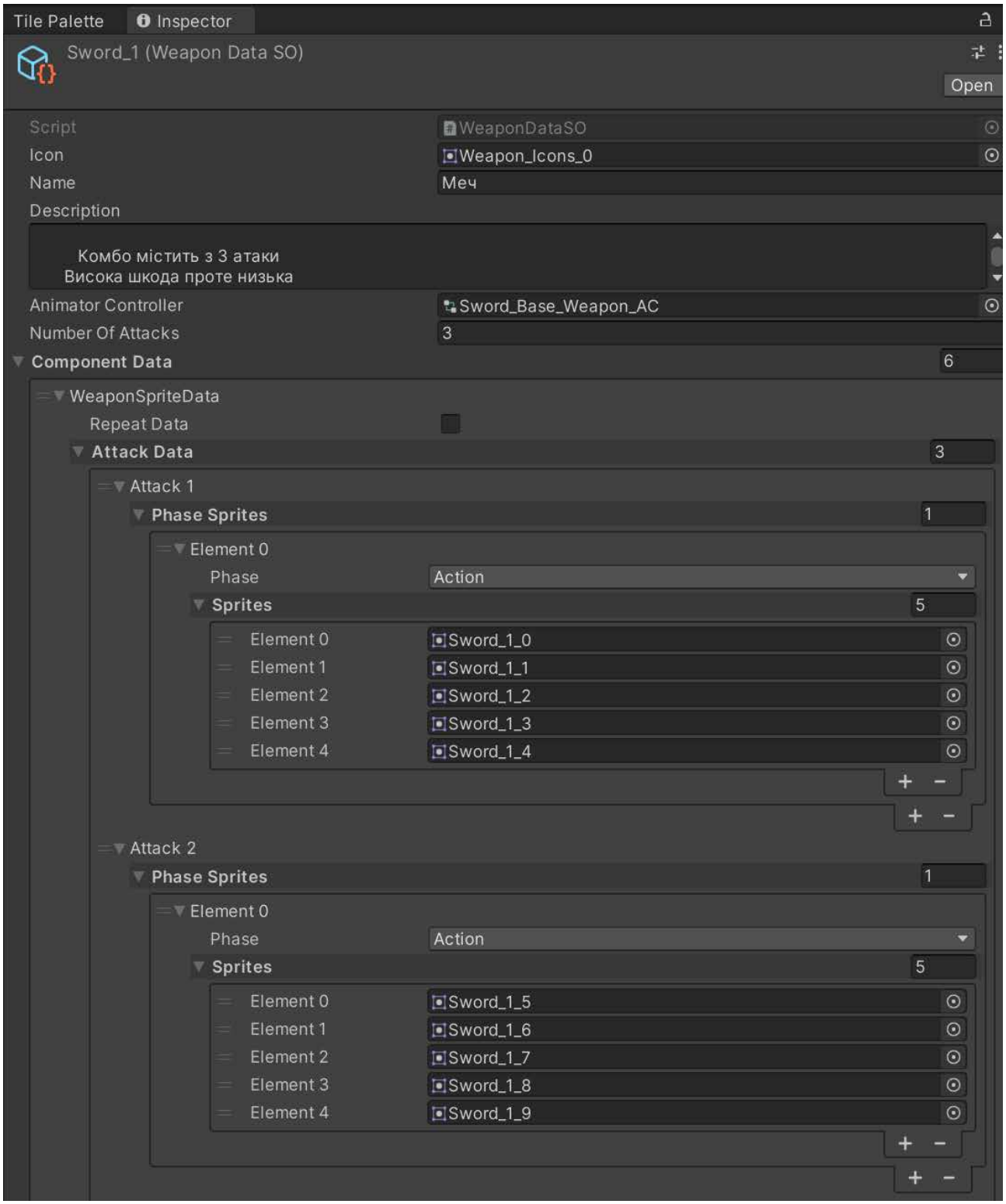
Додаток Б

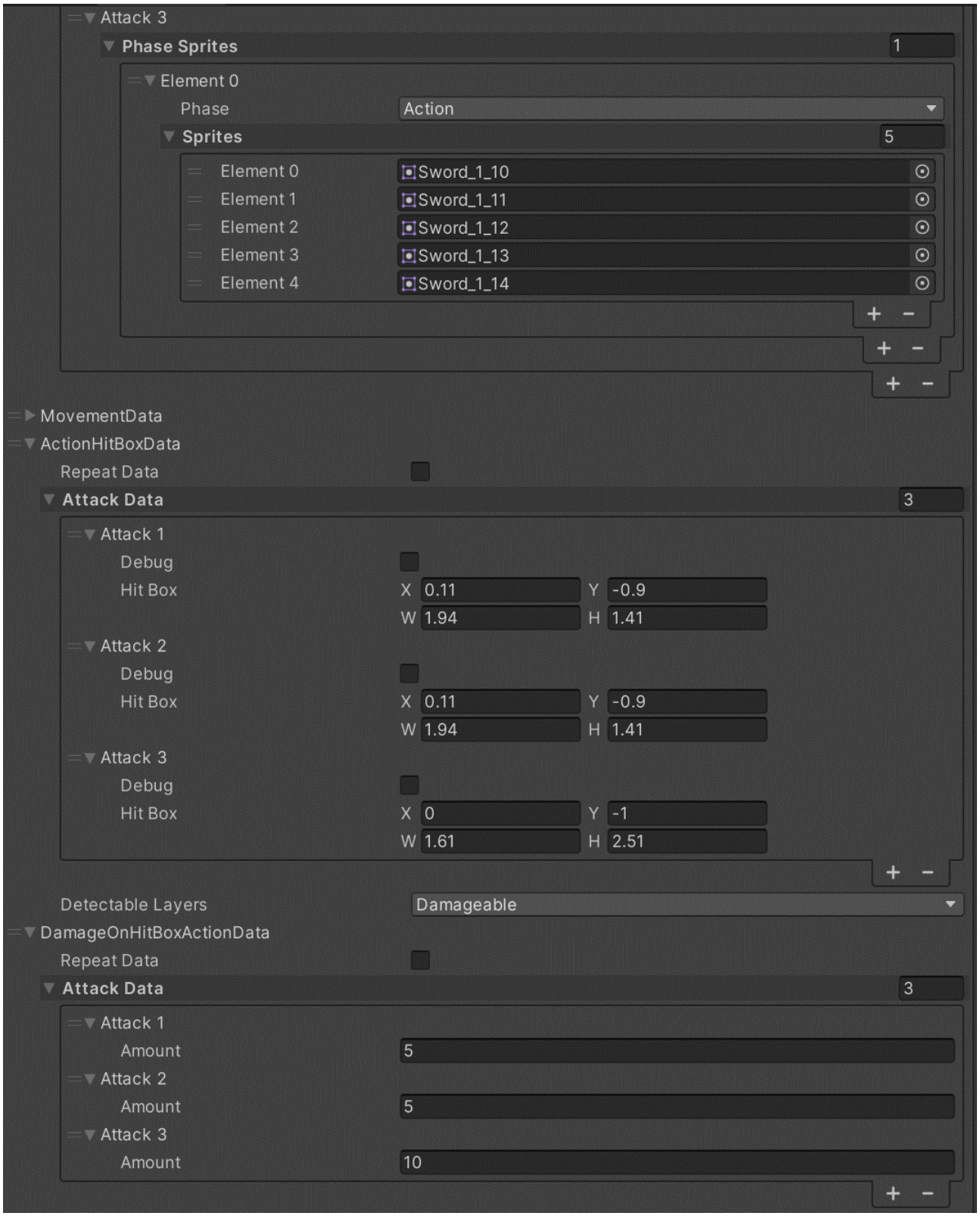
Діаграма класів

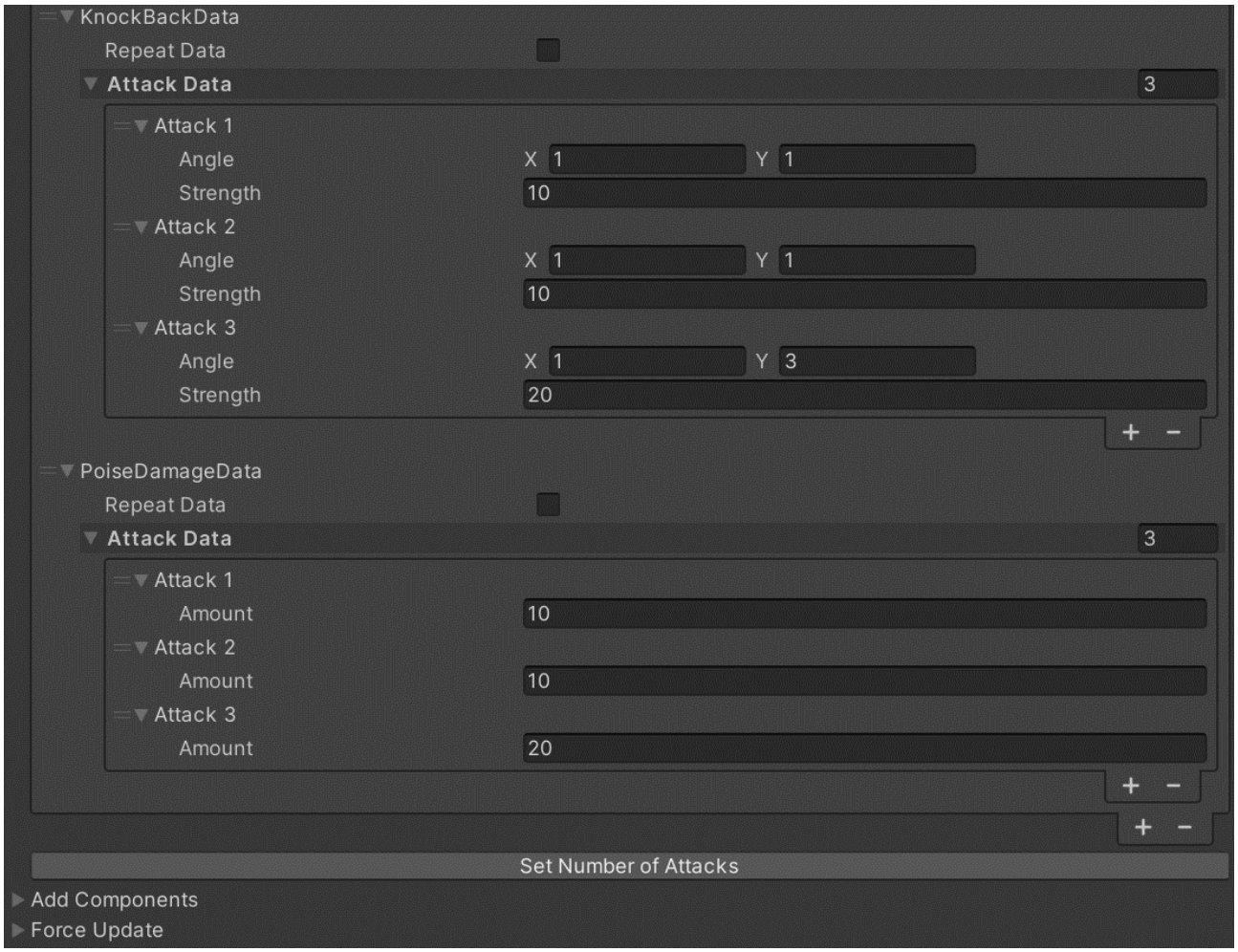


Додаток В

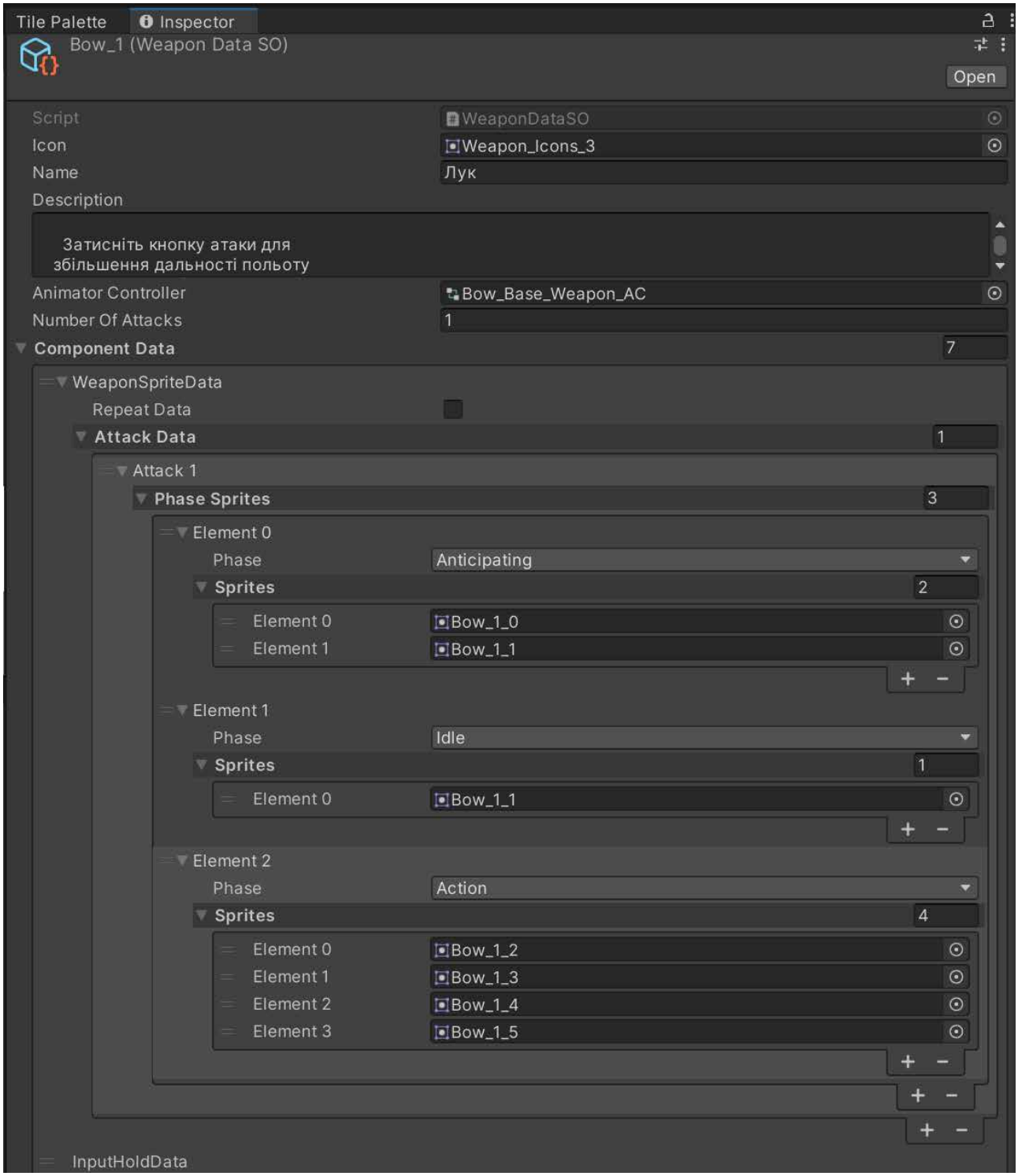
Скриптований об’єкт меча

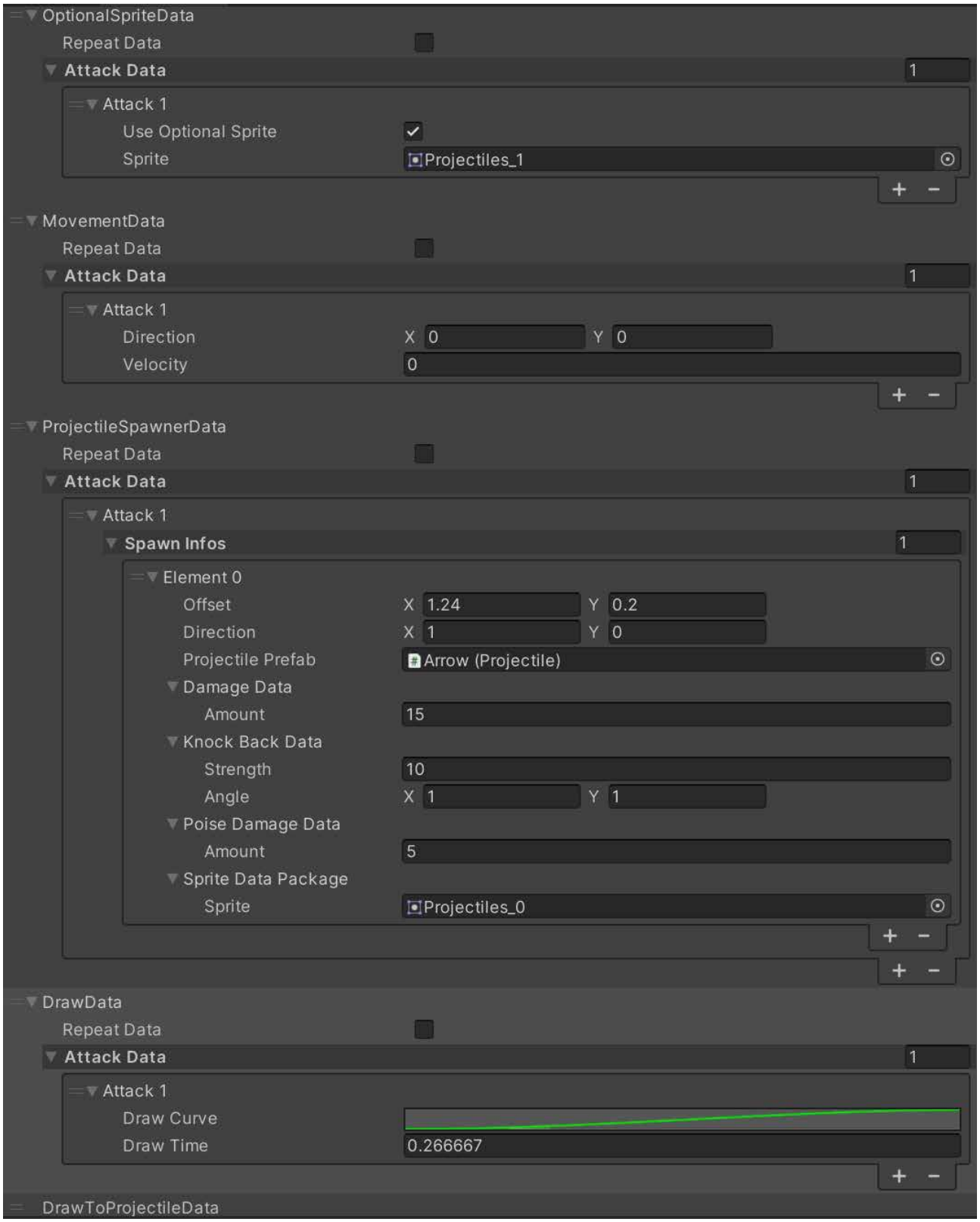



Attack 3
Phase Sprites 1
Element 0
Phase Action
Sprites 5
Element 0 Sword_1_10
Element 1 Sword_1_11
Element 2 Sword_1_12
Element 3 Sword_1_13
Element 4 Sword_1_14
MovementData
ActionHitBoxData
Repeat Data
Attack Data 3
Attack 1
Debug
Hit Box X 0.11 Y -0.9 W 1.94 H 1.41
Attack 2
Debug
Hit Box X 0.11 Y -0.9 W 1.94 H 1.41
Attack 3
Debug
Hit Box X 0 Y -1 W 1.61 H 2.51
Detectable Layers Damageable
DamageOnHitBoxActionData
Repeat Data
Attack Data 3
Attack 1
Amount 5
Attack 2
Amount 5
Attack 3
Amount 10

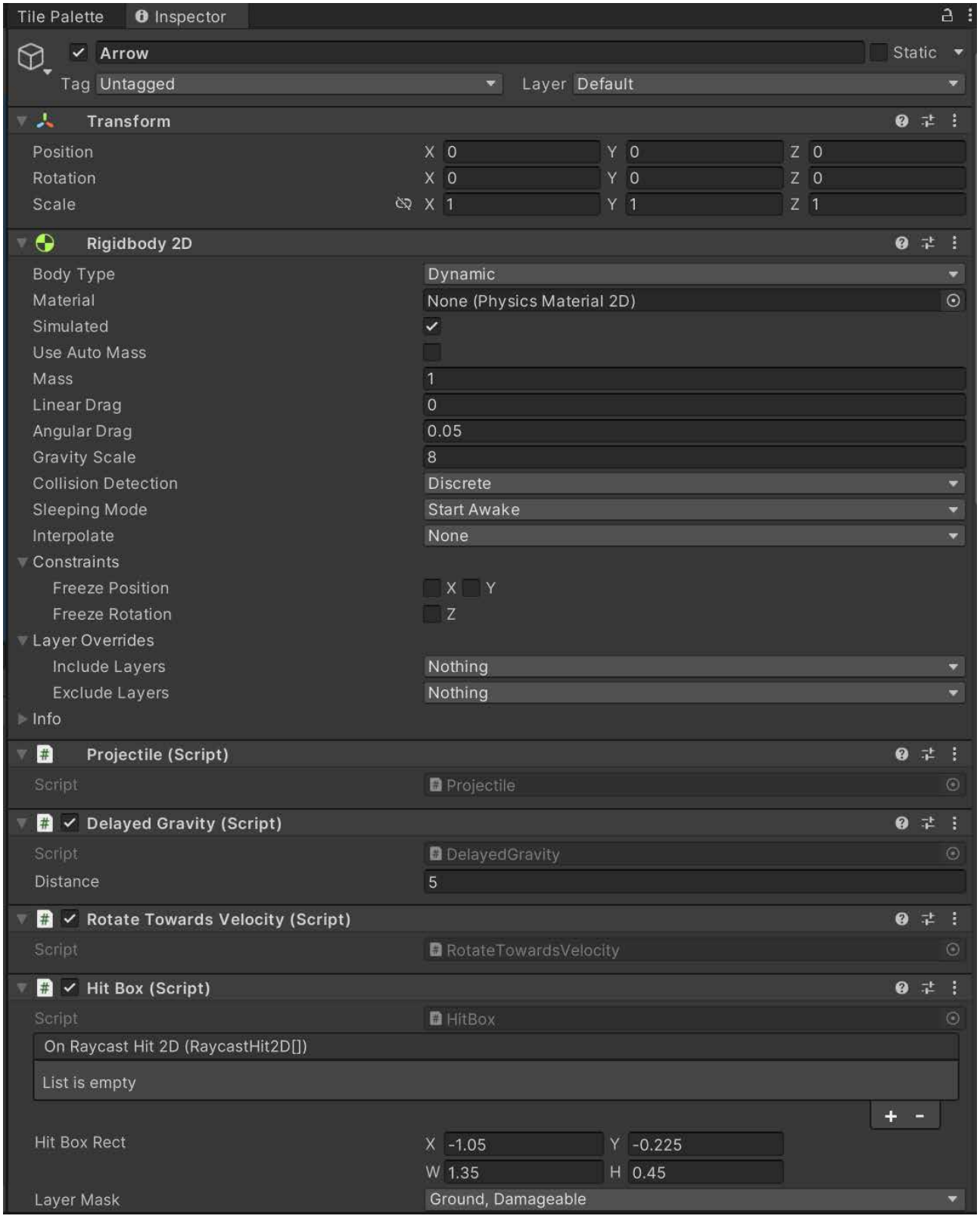

KnockBackData
Repeat Data
Attack Data
3
Attack 1
Angle
X 1
Y 1
Strength
10
Attack 2
Angle
X 1
Y 1
Strength
10
Attack 3
Angle
X 1
Y 3
Strength
20
PoiseDamageData
Repeat Data
Attack Data
3
Attack 1
Amount
10
Attack 2
Amount
10
Attack 3
Amount
20
Set Number of Attacks
Add Components
Force Update

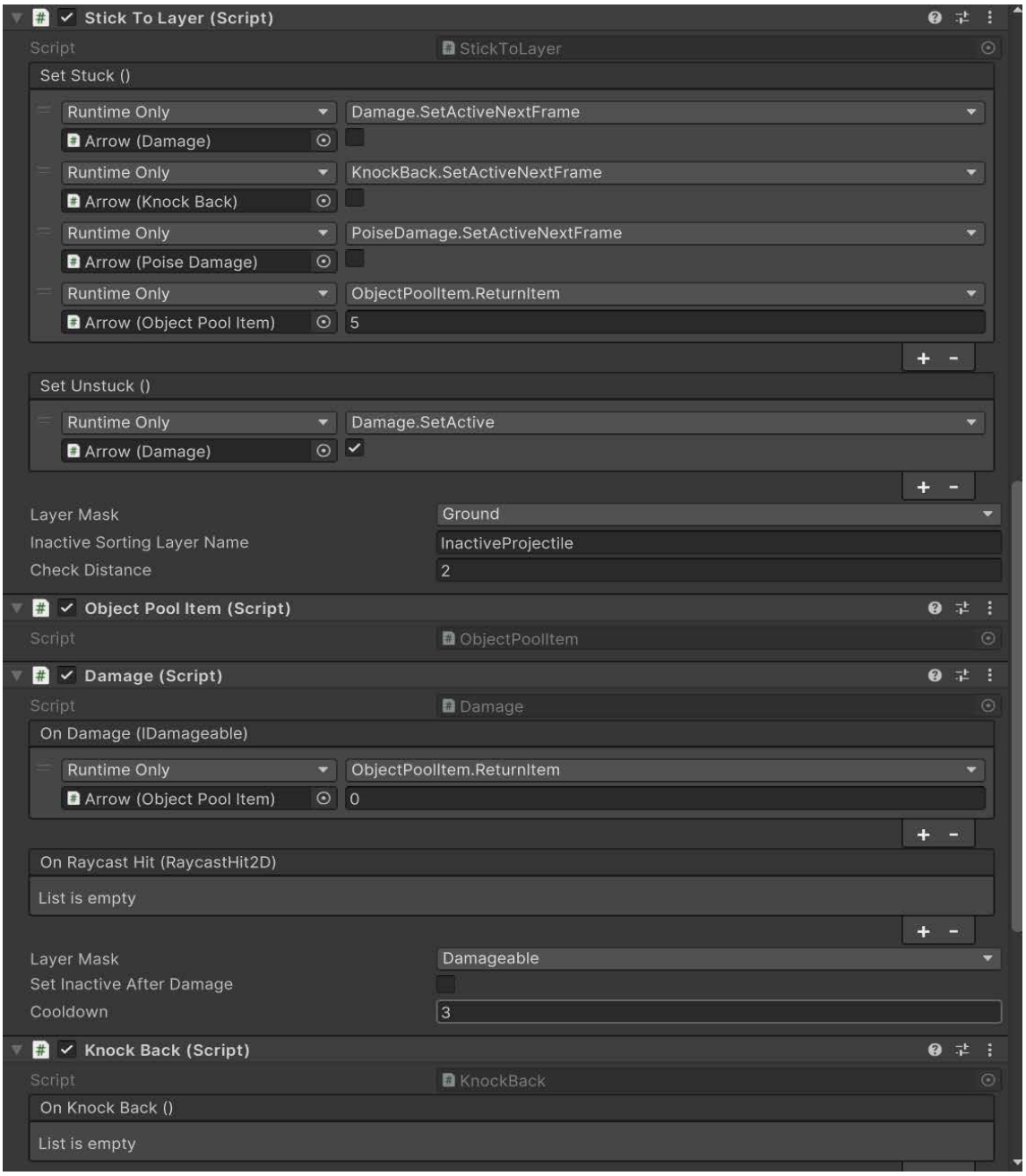
Скриптований об’єкт лука

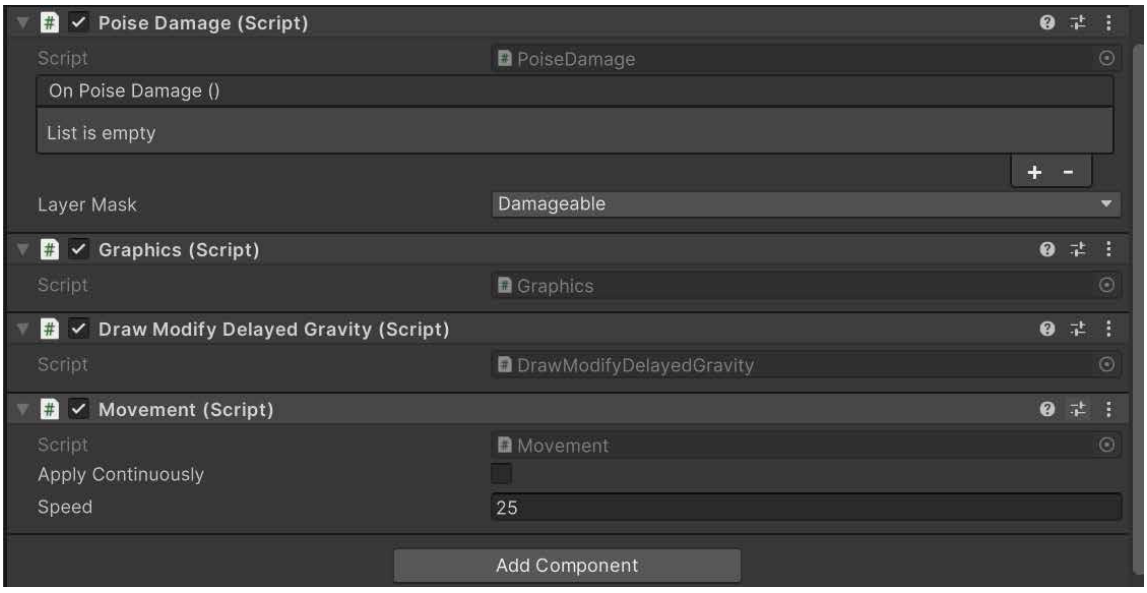



OptionalSpriteData
Repeat Data
Attack Data
1
Attack 1
Use Optional Sprite
Sprite
Projectiles_1
MovementData
Repeat Data
Attack Data
1
Attack 1
Direction
X 0
Y 0
Velocity
0
ProjectileSpawnerData
Repeat Data
Attack Data
1
Attack 1
Spawn Infos
1
Element 0
Offset
X 1.24
Y 0.2
Direction
X 1
Y 0
Projectile Prefab
Arrow (Projectile)
Damage Data
Amount
15
Knock Back Data
Strength
10
Angle
X 1
Y 1
Poise Damage Data
Amount
5
Sprite Data Package
Sprite
Projectiles_0
DrawData
Repeat Data
Attack Data
1
Attack 1
Draw Curve
Draw Time
0.266667
DrawToProjectileData

Шаблон стріли



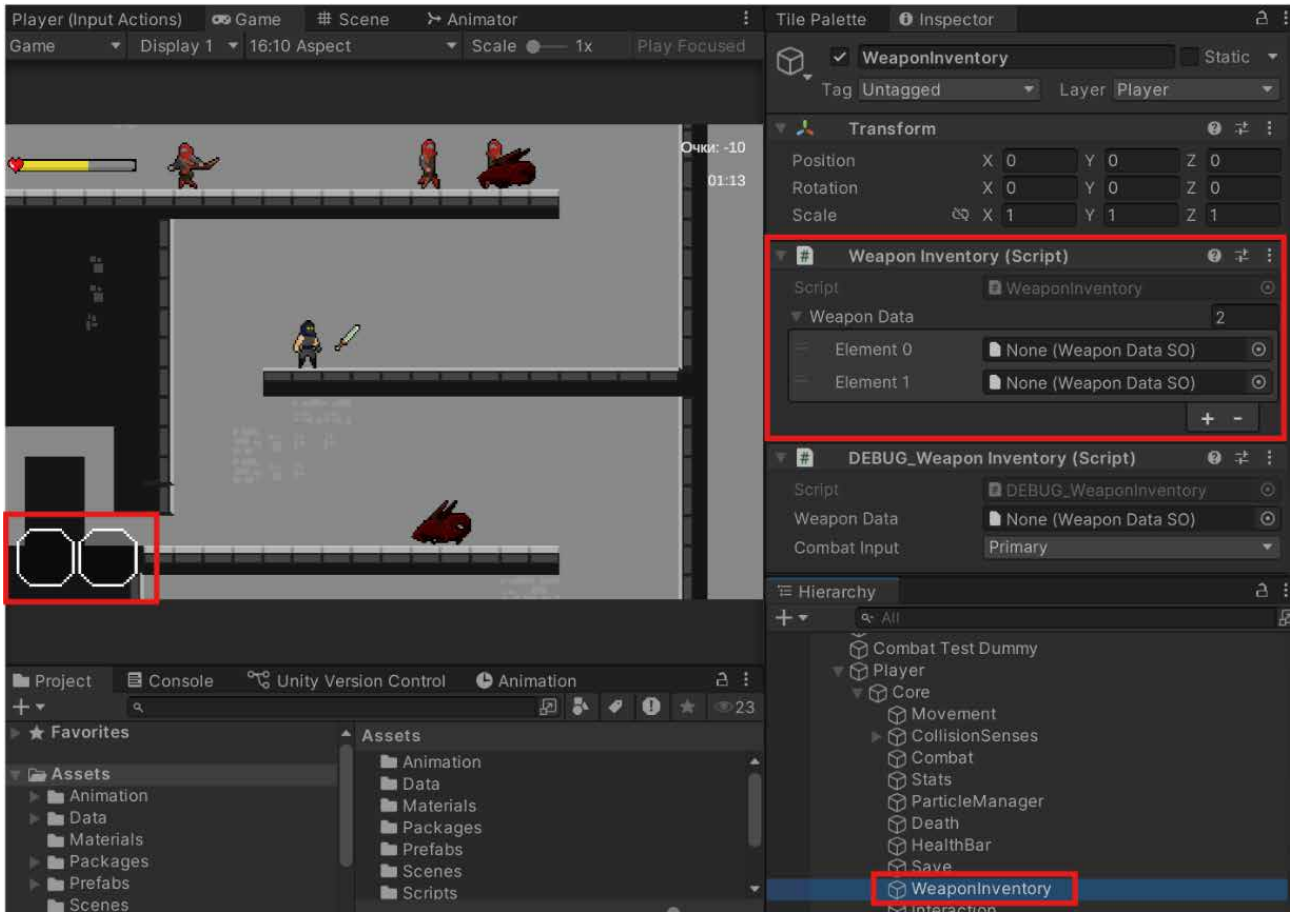

Stick To Layer (Script)
Script
StickToLayer
Set Stuck ()
Runtime Only
Damage.SetActiveNextFrame
Arrow (Damage)
Runtime Only
KnockBack.SetActiveNextFrame
Arrow (Knock Back)
Runtime Only
PoiseDamage.SetActiveNextFrame
Arrow (Poise Damage)
Runtime Only
ObjectPoolItem.ReturnItem
Arrow (Object Pool Item)
5
Set Unstuck ()
Runtime Only
Damage.SetActive
Arrow (Damage)
Layer Mask
Ground
Inactive Sorting Layer Name
InactiveProjectile
Check Distance
2
Object Pool Item (Script)
Script
ObjectPoolItem
Damage (Script)
Script
Damage
On Damage (IDamageable)
Runtime Only
ObjectPoolItem.ReturnItem
Arrow (Object Pool Item)
0
On Raycast Hit (RaycastHit2D)
List is empty
Layer Mask
Damageable
Set Inactive After Damage
Cooldown
3
Knock Back (Script)
Script
KnockBack
On Knock Back ()
List is empty


Poise Damage (Script)
Script
PoiseDamage
On Poise Damage ()
List is empty
Layer Mask
Damageable
Graphics (Script)
Script
Graphics
Draw Modify Delayed Gravity (Script)
Script
DrawModifyDelayedGravity
Movement (Script)
Script
Movement
Apply Continuously
Speed
25
Add Component

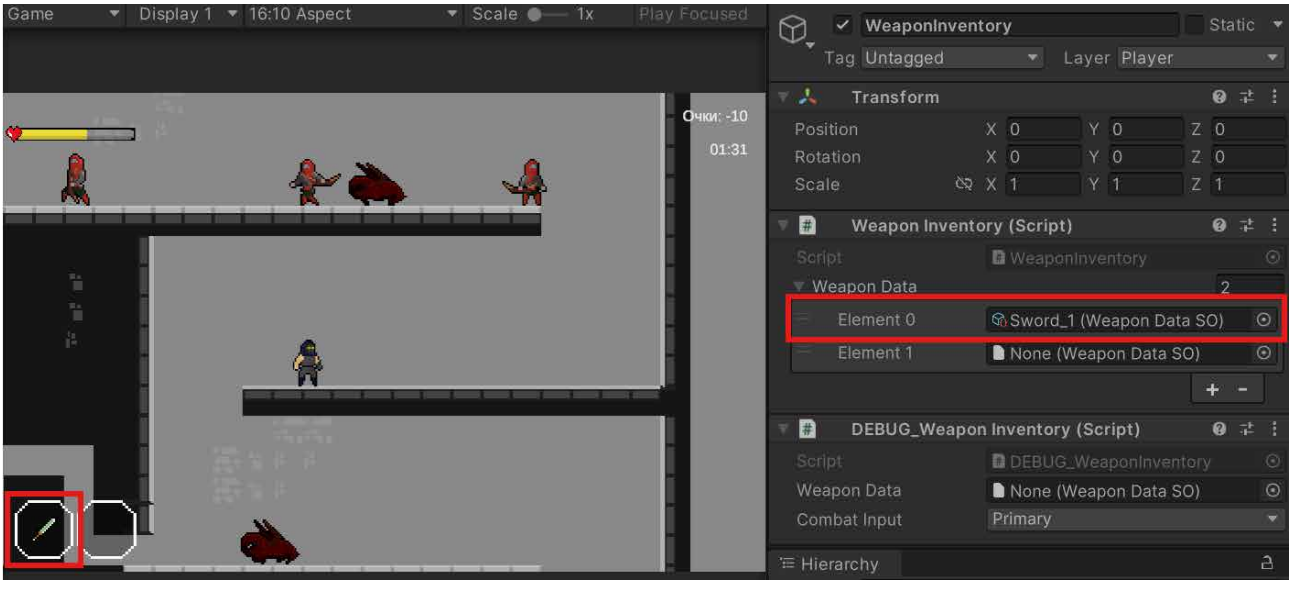
Додаток Д

Перший сценарій

Крок 1

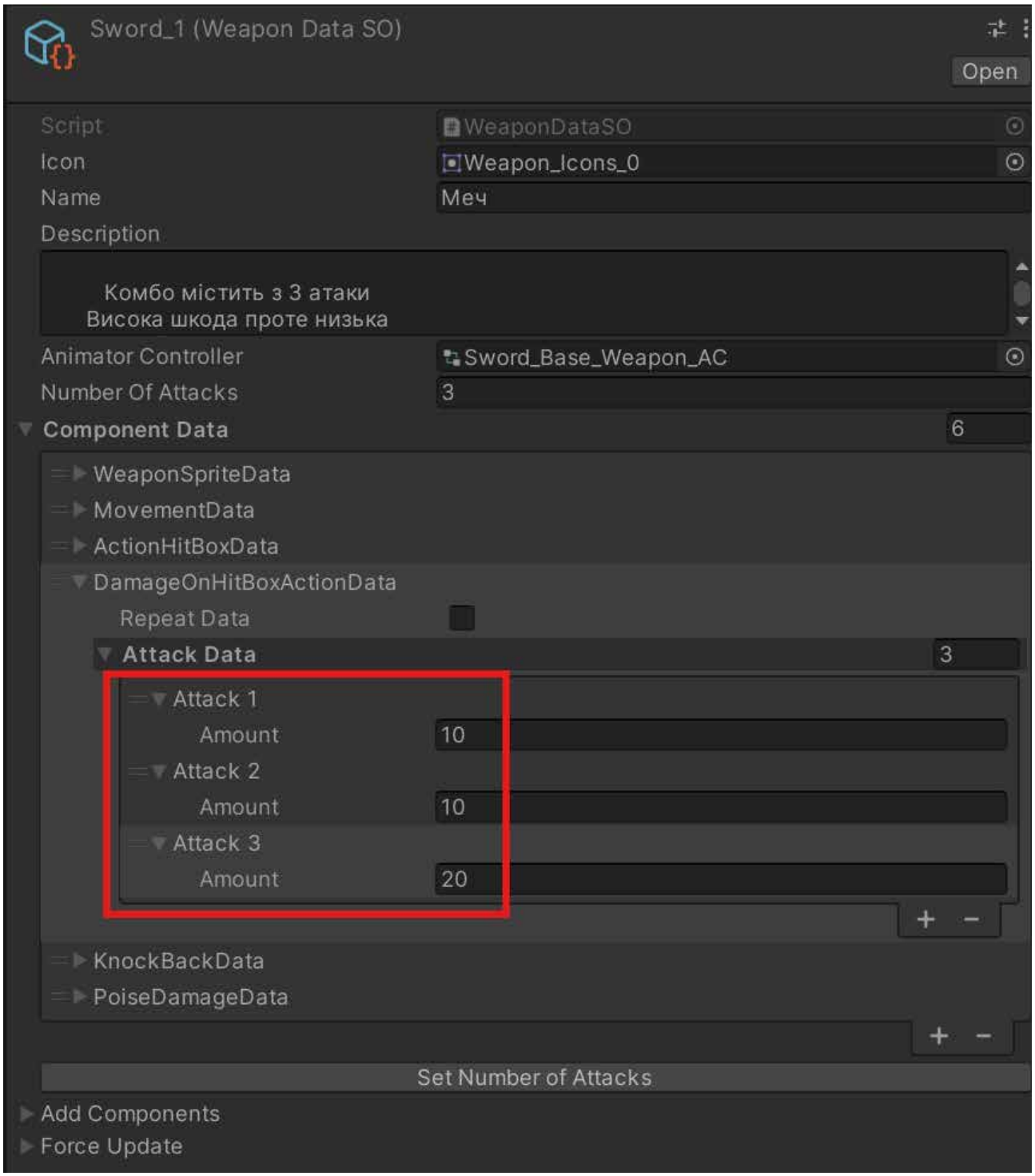


Крок 2



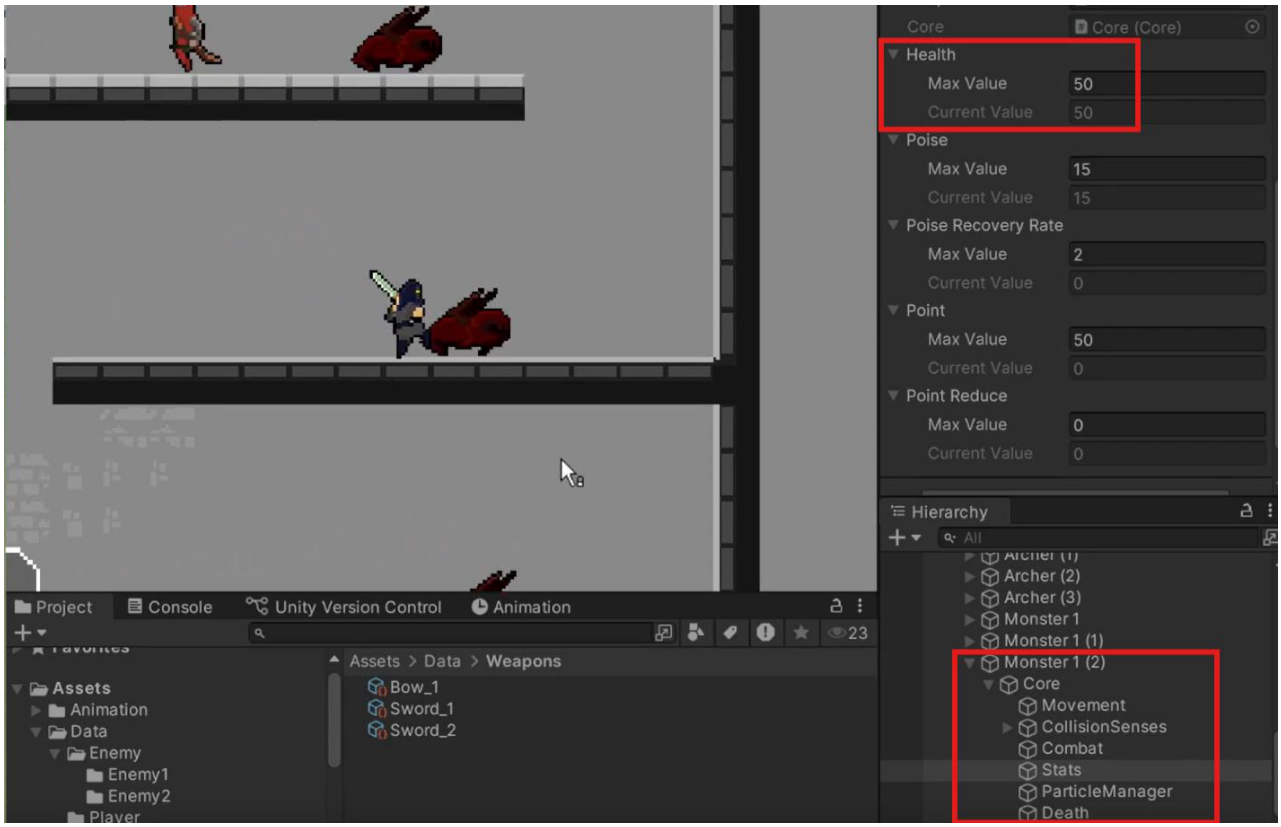
Другий сценарій

Крок 1

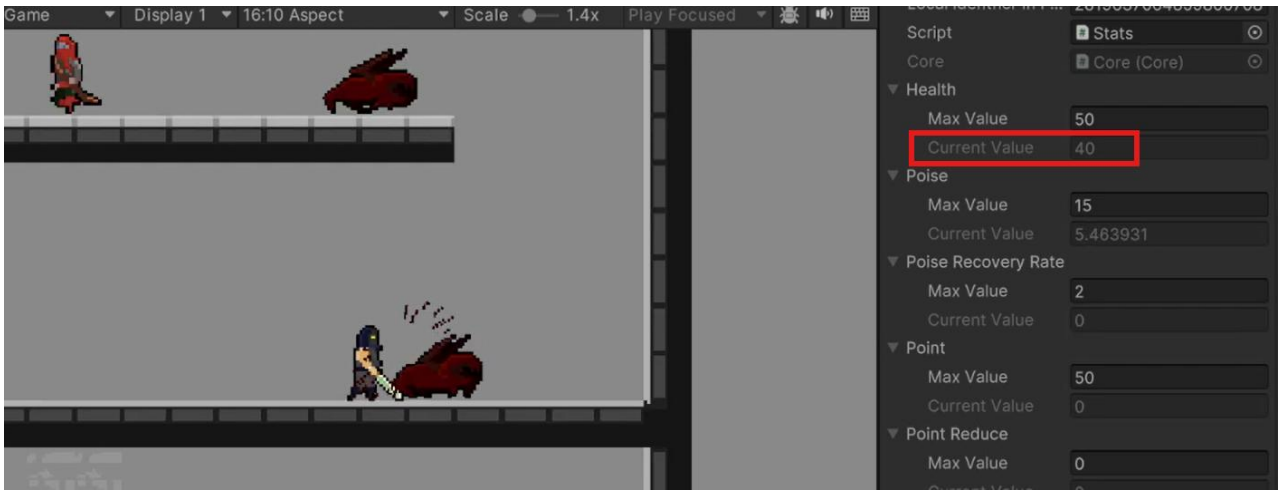


Крок 2

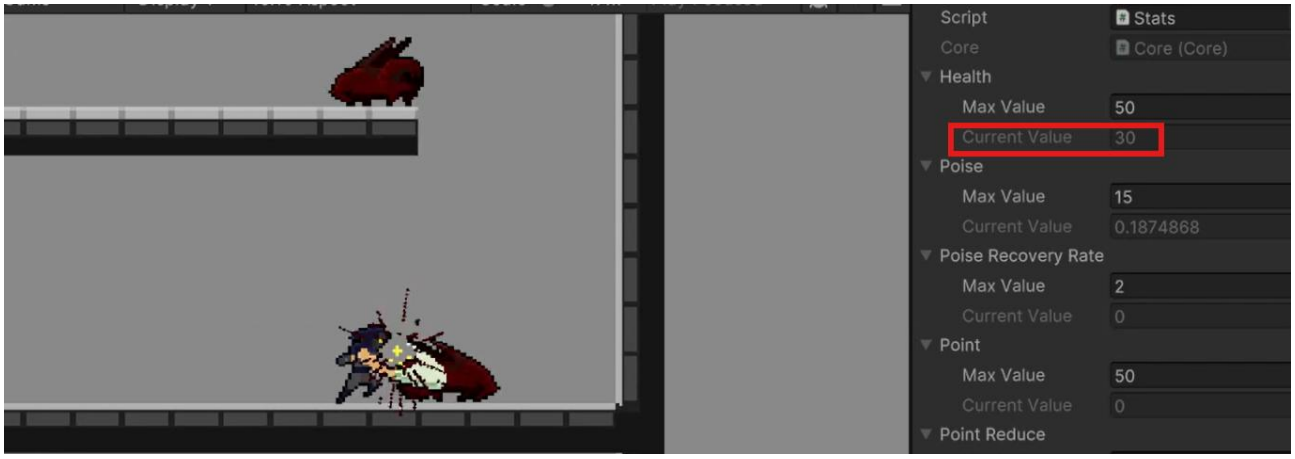
Значення перед серією атак



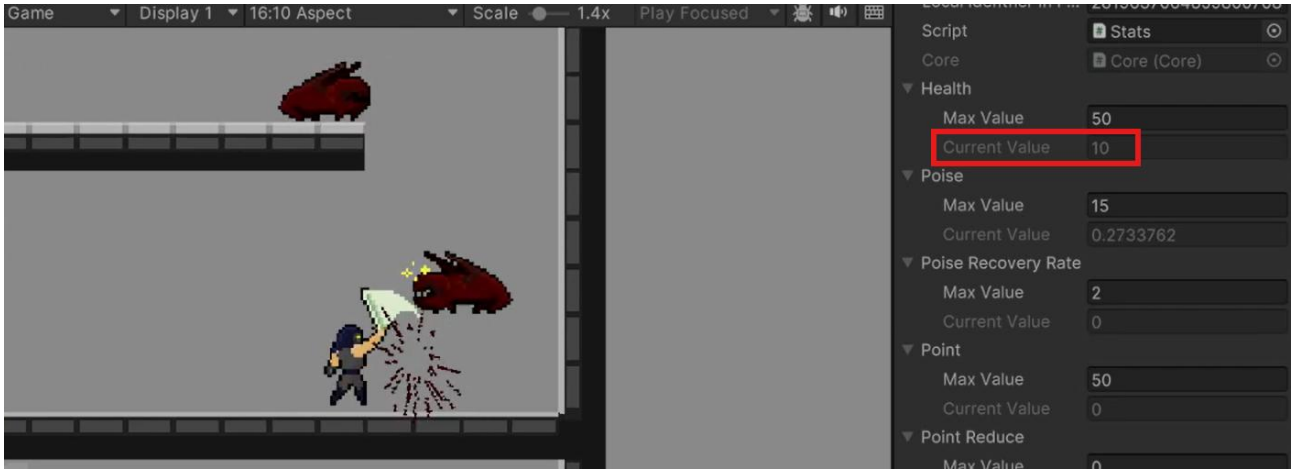
Значення після першої атаки



Значення після другої атаки

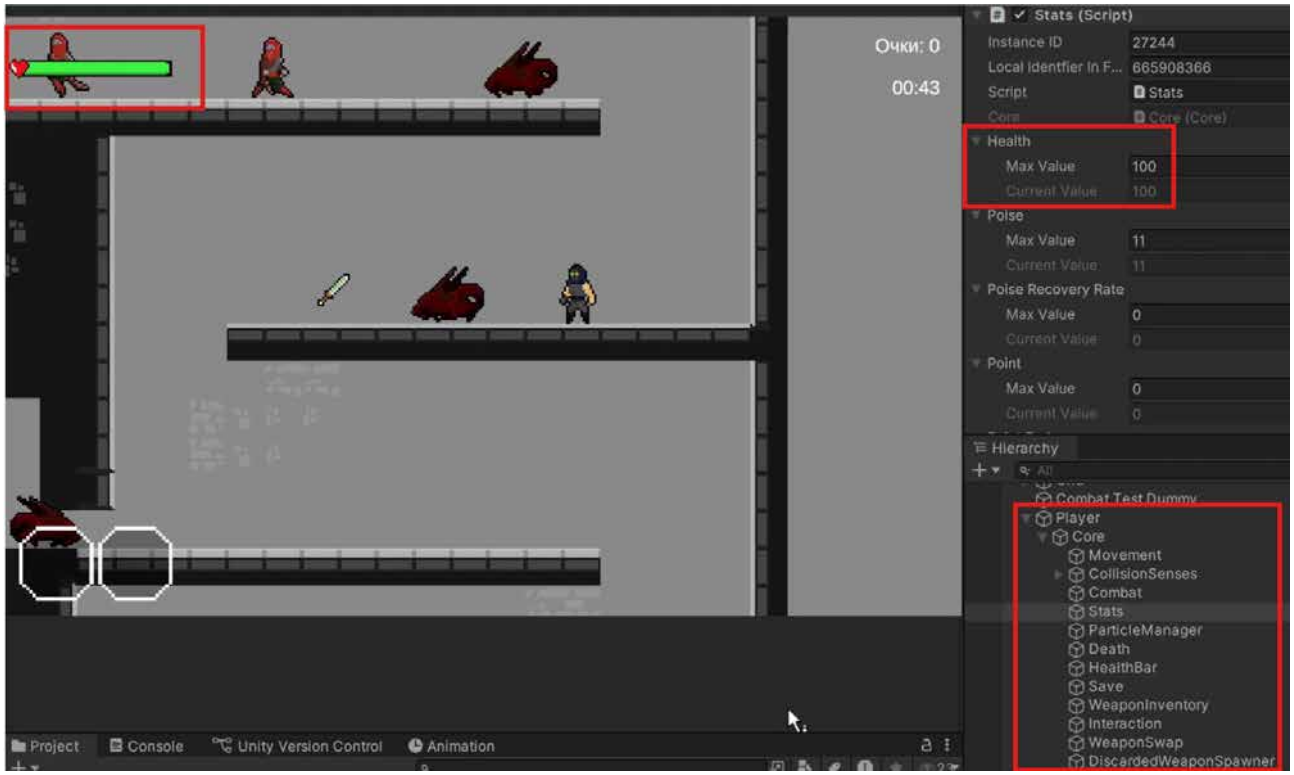


Значення після третьої атаки

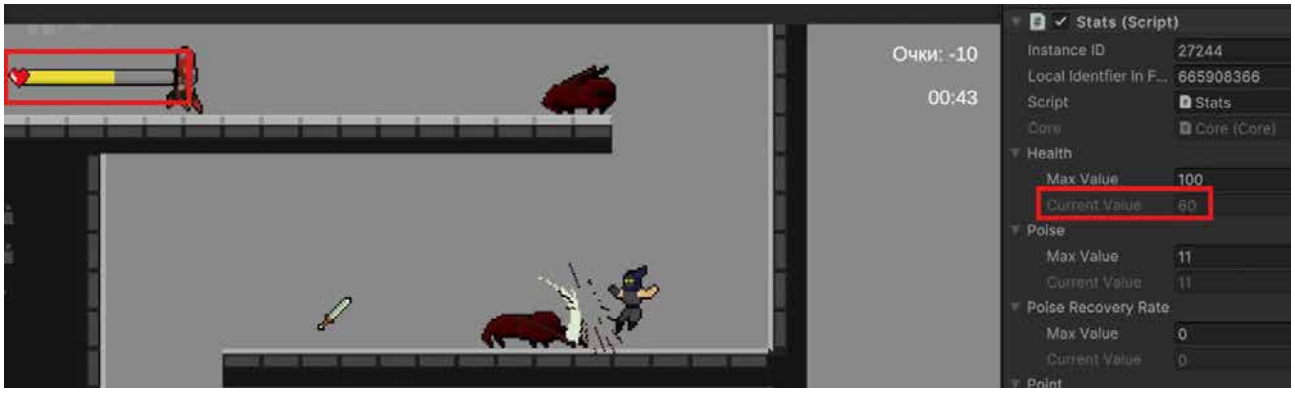


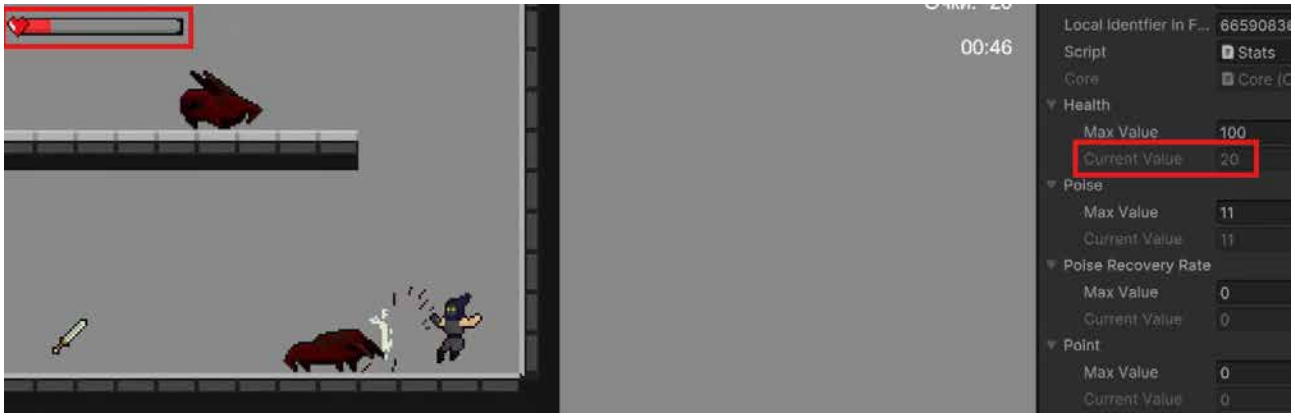
Третій сценарій

Крок 1



Крок 2





Четвертий сценарій