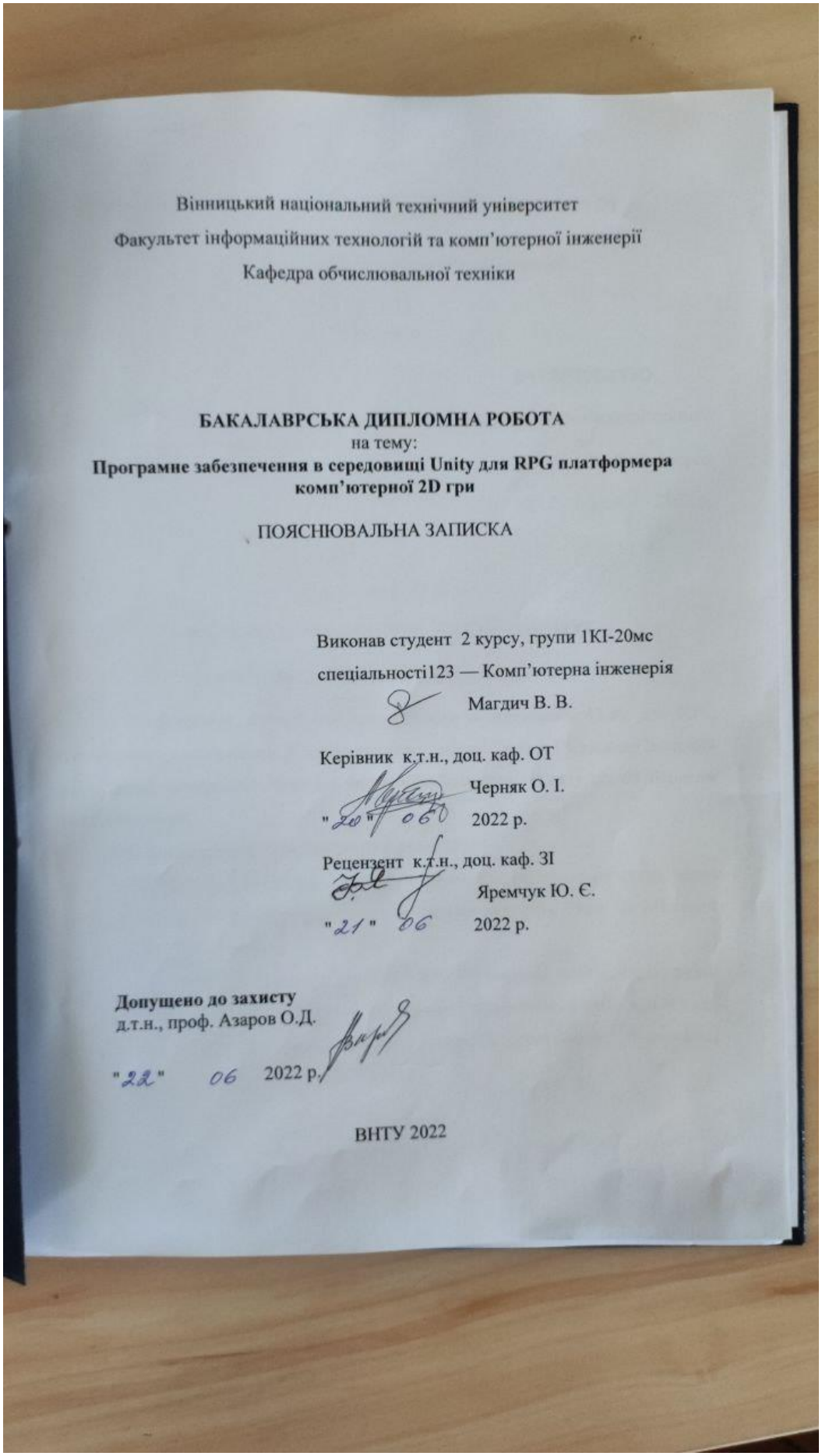


Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА
на тему:
Програмне забезпечення в середовищі Unity для RPG платформера комп'ютерної 2D гри

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав студент 2 курсу, групи 1КІ-20мс
спеціальності123 — Комп'ютерна інженерія
Магдич В. В.

Керівник к.т.н., доц. каф. ОТ
Черняк О. І.
"20" 06 2022 р.

Рецензент к.т.н., доц. каф. ЗІ
Яремчук Ю. Є.
"21" 06 2022 р.

Допущено до захисту
д.т.н., проф. Азаров О.Д.

"22" 06 2022 р.

ВНТУ 2022

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітній рівень — бакалавр
Спеціальність — 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри обчислювальної техніки

О.Д. Азаров

"08" 02 2022 р.

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ

Суденту **Магдич Валентину Володимировичу**

1 Тема роботи «Програмне забезпечення в середовищі Unity для RPG платформера комп'ютерної 2D гри» керівник роботи Черняк Олександр Іванович к.т.н., доц., затверджено наказом вищого навчального закладу від **09 березня 2022** року № **65**

2 Строк подання студентом роботи **14.06.22**.

3 Вихідні дані до роботи: технічний опис програмного застосунку, мова програмування C#, комп'ютерна гра, середовище розробки Unity та Microsoft Visual Studio.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):вступ, аналіз сучасних методів розробки,варіантний вибір засобів для розробки програмного забезпечення, програмна реалізація, тестування розробленого програмного забезпечення.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): блок-схема алгоритму обробки події персонажа.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Черняк О.І., к.т.н., доц.		

7 Дата видачі завдання 09.03.22.

8 Календарний план наведено в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання	Підпис
1	Постановка задачі роботи	09.03.22	
2	Пошук матеріалів по технологіям розробки комп'ютерних ігор	10.03.22 —25.03.22	
3	Обгрунтування та вибір засобів реалізації комп'ютерної гри	26.03.22 — 04.04.22	
4	Розробка програмного забезпечення	05.04.22 —29.04.22	
5	Створення левел дизайну для гри	30.04.22 —27.05.22	
6	Оформлення пояснювальної записки та ілюстративного матеріалу	28.05.22 — 06.06.22	
7	Перевірка якості виконання бакалаврської роботи та усунення недоліків	07.06.22	

Студент Магдич Валентин Володимирович

Керівник роботи к.т.н., доц. Черняк Олександр Іванович

АНОТАЦІЯ

Метою роботи є створення прототипу однокористувацького двовимірного платформера для персональних комп'ютерів. Для досягнення цілі були проаналізовані сучасні тенденції в розробці комп'ютерних ігор, досліджено засоби розробки комп'ютерних ігор та проекти конкурентів. На основі цих елементів створено прототип однокористувацького двовимірного платформера для персональних комп'ютерів, що містить у собі 1 рівень із реалізованою основною ігровою механікою.

Завдяки швидкому та динамічному розвитку ігрової індустрії в Україні та у світі, комп'ютерні ігри незабаром почнуть займати один з основних напрямків у розвитку сучасного суспільства та масової культури.

Ключові слова: комп’ютерна гра, платформер, RPG, C#, Unity.

ABSTRACT

The aim of the work is to create a prototype of a single-user two-dimensional platformer for personal computers. To achieve this goal, current trends in computer game development were analyzed, computer game development tools and competitors' projects were studied. Based on these elements, a prototype of a single-user two-dimensional platformer for personal computers was created, which contains 1 level with implemented basic game mechanics.

Due to the rapid and dynamic development of the gaming industry in Ukraine and in the world, computer games will soon begin to occupy one of the main directions in the development of modern society and mass culture.

Keywords: computer game, platformer, RPG, C#, Unity.

ЗМІСТ

ВСТУП

Сучасна людина щодня у всіх сферах життєдіяльності взаємодіє із комп'ютерними технологіями. З розвитком технологій збільшується рівень взаємодії. Одним із самих поширених проявів взаємодії є «комп'ютерні ігри» або «відеоігри» .

Індустрія відеоігор зародилася в середині 70-х років минулого століття як рух ентузіастів. В даний час ігрова індустрія активно розвивається. Щороку на ринок випускаються десятки тисяч нових ігор, які розходяться мільйонами копій по всьому світу та приносять десятки мільярдів доларів ігрової промисловості [1].

Комп'ютерні ігри — один із головних двигунів прогресу в інформаційній сфері на сьогоднішній день. З кожним роком ігри набувають нові можливості, а системні вимоги зростають, як наслідок, з'являється нове технічне обладнання, що дозволяє повністю розкрити технічний потенціал гри.

Щоб не витрачати час і не розробляти програму з нуля, розробники користуються готовими рішеннями для ігор — ігровими платформами. На даний момент одне з найпопулярніших і ефективних рішень надає компанія Unity Technologies, яка у червні 2005 року випустила платформу розробки для створення 2D та 3D ігор — Unity. Платформа Unity має широкий спектр можливостей, зручним та інтуїтивно зрозумілим інтерфейсом. Великою перевагою Unity є кросплатформова розробка, яка робить легким та швидким портування ігор під такі платформи як Windows, iOS, Android, Windows Phone 8, а також розробляти ігри для Xbox, PS, Nintendo, веб-браузерів і навіть VR [2].

Тому розвиток технологій у цьому напрямі можна вважати одним з найперспективніших.

Об'єктом дослідження є процес створення комп'ютерних ігор.

Предмет дослідження — технології для розробки комп'ютерної гри.

Метою роботи є розробка програмного забезпечення в середовищі Unity для комп’ютерної 2D гри, та поєднання двох жанрів RPG та платформера.

Для досягнення поставленої мети необхідно вирішити наступні **задачі**:

— проаналізувати предметну область;

— проаналізувати існуючі аналоги;

— створення концептуальних моделей;

— виконати моделювання гри;

— здійснити програмну реалізацію ігрового проекту.

Апробація результатів бакалаврської роботи: зроблено доповідь на науково–технічній конференції “молодь в науці” підрозділів Вінницького національного технічного університету.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДЛЯ РОЗРОБКИ КОМП'ЮТЕРНОЇ ГРИ ПЛАТФОРМЕРА

1.1 Поняття платформеру, класифікація та порівняння данного жанру

Існує безліч жанрів комп'ютерних ігор: активні шутери, що вимагають від гравця високого рівня концентрації та швидкої реакції, логічні головоломки та квести, різні рольові ігри, де гравцеві необхідно вжитися в роль сюжетного персонажа і так далі.

Кожен із жанрів може бути виконаний як у 2D, так і в 3D, однак, є деякі особливості, які роблять той чи інший формат кращим. Звичайно, 3D ігри більш привабливі для гравців, ніж 2D, тому що вони пропонують більшу реалістичність, як графічну, так і ігрову, та й, загалом, 2D графіка дещо застаріла, проте, 2D-ігри все ще популярні серед певного, та й досить багатого обсягу гравців.

Переваги та недоліки двох видів виконання ігор представлені у таблиці 1.

Таблиця 1 — Порівняння 2D та 3D ігор

	2D	3D
Переваги	Досить низька вартість розробки. Не вимагає від розробника великого досвіду роботи. Ідеально підходить для створення малих та середніх проектів за невеликий термін. Максимально відповідні жанри: квести, платформери та головоломки. Можливість поєднання незвичайних варіацій графіки та геймплею.	Реалістична та красива графіка. Більш мальовничий, цікавий та пророблений ігровий світ. Більше можливостей зробити захоплюючим геймплей. Можливість створення проекту будь-якої складності.
Недоліки	2D ігри не такі привабливі для рядового користувача. Ринок розробки менший, а отже менша кількість споживачів. Неможливість створення великих проектів через обмежені можливості розробки у двох площинах.	Висока вартість та складність розробки. Часто потрібна велика команда розробників, де кожен відповідає за окремий аспект гри.

Таким чином, можна зробити висновок, що 2D гра буде ідеальним вибором, так як не потрібно великого досвіду розробки відеоігор, проект може бути розроблений за невелику кількість часу і немає потреби у великих витратах на розробку. Також жанром гри стане саме платформер, оскільки це найпопулярніший жанр 2D ігор на даний момент, до того ж, що має досить великі можливості організувати захоплюючий ігровий процес.

2D-платформер — жанр комп'ютерних ігор, що зараховується багатьма журналами до аркадів, де основною рисою ігрового процесу є стрибання по платформах, лазіння по сходах, збирання предметів, зазвичай необхідних для завершення рівня. Деякі предмети, які називають пауер-апамами (англ. power-up), наділяють протагоніста особливою силою, яка зазвичай вичерпується з часом (наприклад: силове поле, прискорення, збільшення висоти стрибків). Колекційні предмети, зброя та паверапи збираються зазвичай простим дотиком протагоніста, і для застосування не потрібно спеціальних дій із боку гравця. Рідше предмети збираються в «інвентар» героя і використовуються спеціальною командою (така поведінка є більш характерною для аркадних головоломок).

Противники (звані «монстрами» незалежно від зовнішнього вигляду), завжди численні і різнорідні, мають примітивний штучний інтелект, прагнучи максимально наблизитися до протагоніста, або не володіють ним зовсім, переміщаючись по круговій дистанції або роблячи повторювані дії. Дотик із супротивником зазвичай забирає життєві сили в героя або зовсім вбиває його. Противник може бути нейтралізований або стрибком йому на голову, або зі зброї, якщо ним володіє герой. Смерть живих істот зображується спрощено або символічно (істота зникає або провалюється вниз).

Ігри подібного жанру характеризуються нереалістичністю, мальованою мультяшною графікою. Героями таких ігор зазвичай бувають міфічні істоти (наприклад: дракони, гобліни) чи антропоморфні тварини.

Платформери з'явилися на початку 1980-х і стали тривимірними у середині 90-х. Через деякий час після утворення жанру у нього з'явилася ця назва, що

відображає той факт, що в платформер геймплей сфокусований на стрибках по платформах всупереч стрільбі. Щоправда, у багатьох платформерах є стрілецька зброя, в таких, наприклад, як Blackthorne або Castlevania.

1.2 Аналіз ігрових рушіїв для створення комп'ютерних ігор

Основними інструментами розробки ігор є ігрові рушії, що відповідають за низькорівневий опис фізики гри, правила відтворення графіки, а також графічні редактори для відтворення графіки. Ігрові рушії відразу включають всі необхідні алгоритми для правильного функціонування гри та її розвиток.

В даний час існує безліч різноманітних ігрових рушіїв. Основні відмінності полягають у мовах програмування, що підтримуються, функціональність та, не в останню чергу, вартість ліцензії. При виборі середовища розробки акцент був зроблений саме на цих параметрах. Для аналізу платформ на яких створюються ігри були обрані найпопулярніші ігрові рушії, такі як: Unity, Game Maker: Studio та Unreal Engine.

Unreal Engine — ігровий рушій, розроблений і підтримуваний компанією Epic Games.

Першою грою на цьому рушії був шутер від першої особи Unreal, випущений у 1998 році. Хоча рушій спочатку був призначений для розробки шутерів від першої особи, його наступні версії успішно застосовувалися в іграх різних жанрів, у тому числі стелс-іграх, файтингах і масових багатокористувацьких рольових онлайн-іграх. У минулому рушій поширювався на умовах оплати щомісячної передплати, з 2015 року Unreal Engine безкоштовний, але розробники ігор зобов'язані перераховувати 5% прибутку від продажів, але якщо припустити, що гра заробляє більше ніж 3000 доларів за кожен квартал [11].

Написаний на C++, рушій дозволяє створювати ігри для більшості операційних систем та платформ: Windows, Mac OS та Linux; консолі PlayStation,

Xbox, PSP, GameCube, PS Vita, Dreamcast і т.д., а також на пристроях iOS та Android.

Рисунок 1.1 — Інтерфейс рушія Unreal Engine 5

Для спрощення портування рушій використовує модульну систему залежні компоненти; підтримує різні системи візуалізації (Direct3D, OpenGL, Pixomatic, в ранніх версіях: Glide, S3, PowerVR), відтворення аудіо (EAX, OpenAL, A3D DirectSound3D), перетворення тексту на мовлення, розпізнавання мови, модулі для роботи в мережі та підтримка різних пристроїв введення.

Для гри в мережі підтримуються технології Windows Live, Xbox Live, GameSpy та інші, включаючи до 64 гравців (клієнтів) одночасно. Таким чином, рушій адаптували і для застосування в іграх жанру MMORPG (один із прикладів: Lineage II).

Game Maker Studio — ігровий рушій, розроблений і підтримуваний YoYo Games, написаний на Delphi (наступна версія на C#). На даний момент це одна з найпопулярніших ігор рушіїв для розробки 2D ігор для всіх сучасних популярних платформ. Для розробки він використовує спрощену мову програмування GML (Game Maker Language) [6].

Цей ігровий рушій має кілька версій розповсюдження:

— безкоштовний;

— настільний;

— веб;

— UWP;

— мобільний.

Кожна з цих версій (крім Trial) відрізняється платформою, для якої буде створено гру. Пробна версія – версія безкоштовна та надається для тих, хто хоче випробувати ігровий рушій, він має ряд обмежень щодо кількості об'єктів, що використовуються в грі, а також дозволить зібрати проект тільки для тестування на операційній системі Windows.

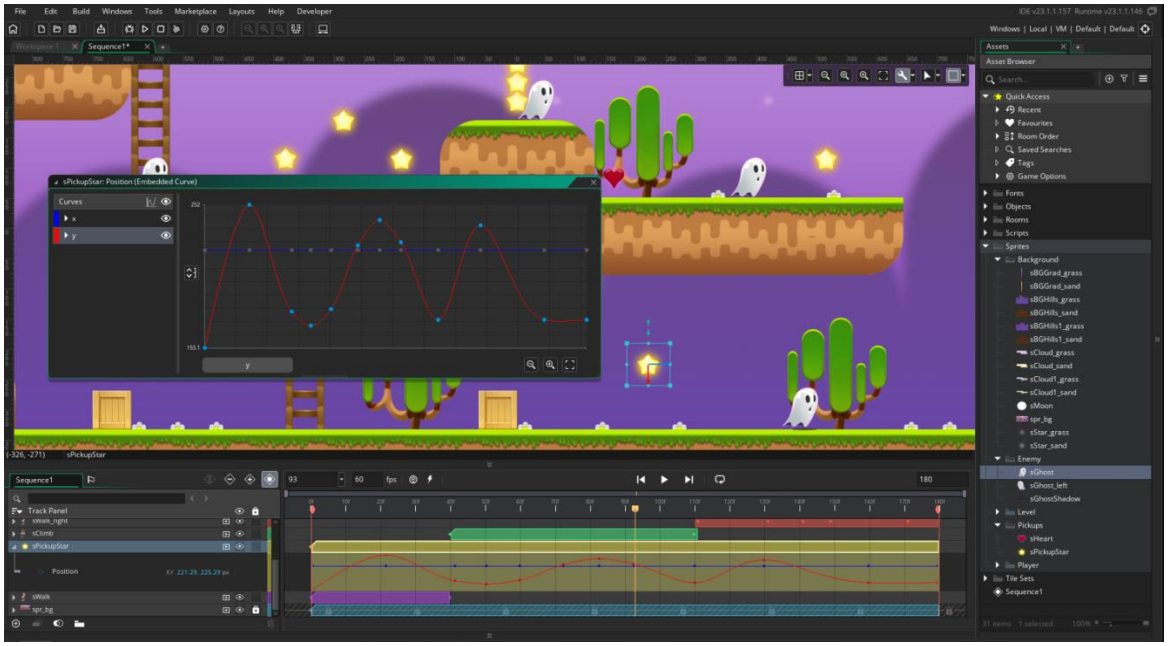


Рисунок 1.2 — Приклад інтерфейсу Game Maker Studio 2

Як і будь-який інший інструмент для програмування чого-небудь, GameMaker Studio має свої переваги і недоліки.

Серед переваг можна виділити:

— наявність графічного інтерфейсу, що дозволяє створювати ігри, не вдаючись до написання скриптів;

— кросплатформенність, що відкриває можливість моделювати ігри для різних операційних систем в одному інструменті;

— невисока ціна та безкоштовний тестовий період, під час якого ви можете спробувати цей інструмент та вирішити, чи варто за нього платити, хоч і коштує він відносно недорого — 99$ за довічну ліцензію;

— власна мова програмування, яка має багато спільного з іншими популярними мовами, що полегшує її вивчення;

— власний магазин готових прикладів для створення ігор: спрайти, анімації, елементи та ін.

Серед недоліків можна виділити:

— погана якість 3D-ігор, тому можна вважати, що для таких ігор цей рушій не підходить;

— ігри, зроблені на GameMaker Studio, неможливо перенести на інші рушії, наприклад, якщо в майбутньому ви захочете перенести свою гру на Unity, зробити це не вийде.

Unity — інструмент для розробки двовимірних і тривимірних ігор, що є однією з найпопулярніших систем розробки на сьогоднішній день. Дозволяє створювати програми, що працюють на більшості сучасних операційних систем (Windows, OS X, Windows Phone, Android, Apple iOS, Linux), а також на ігрових консолях Wii, PlayStation, Xbox та MotionParallax3D (пристрої для відтворення віртуальних голограм), наприклад, Nettlebox. Є можливість створювати додатки для запуску в браузерах за допомогою спеціального плагіна Unity (Unity Web Player), а також використанням реалізації технології WebGL. Останні версії Unity дозволяють створювати програми для окулярів віртуальної реальності.

Редактор Unity має простий інтерфейс Drag&Drop, який легко використовувати, складається із різних вікон, щоб ви могли налагоджувати гру безпосередньо в редакторі. Рушій підтримує два сценарії мови: C# і JavaScript (модифікація). Фізичні розрахунки проводяться за допомогою фізичного рушія NVIDIA PhysX.

Проект Unity поділений на сцени (рівні) — окремі файли, що містять їх ігрові світи з власним набором об'єктів, сценаріїв та налаштувань. Сцени можуть містити як власне об'єкти (моделі), так і порожні ігрові об'єкти — об'єкти, що не мають моделі («пустишки"). Об'єкти, у свою чергу, містять набори компонентів, з якими і скрипти взаємодіють. Також об'єкти мають назву (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися.

Unity поширюється безкоштовно, але крім безкоштовного є чотири збірки — стандартна Unity, Unity iOS Pro, Android Pro та командна ліцензія. Вони відрізняються вартістю та функціональністю.

Безкоштовна версія має деякі обмеження, але можливості розповсюджувати гру можна за умови, що річного доходу від гри не перевищує 100 000 доларів США. З виходом Unity 5 багато обмежень безкоштовної версії було знято, але річний дохід від гри все одно не повинен перевищувати зазначеного.

Все це робить Unity одним із найпопулярніших рушіїв для початківців ігрових розробників.

1.3 Огляд існуючих розробок

Як проект був обраний однокористувацький 2D — платформер, тому основних конкурентів варто шукати у тій же жанровій категорії. Але варто пам'ятати, що цей жанр втратив свою популярність, через чого сучасні ігри цього жанру часто не є чистими платформерами. Найяскравішими представниками останнім часом були:

— Ori and the Blind Forest;

— Mark of the Ninja;

— Dust An Elysian Tail;

— Limbo;

— DuckTales Remastered;

— Wings of Vi.

Ori and the Blind Forest [9] — гра в жанрі платформер розроблена студією Moon Studios та видана Microsoft Studios для платформ Windows та Xbox One. Гра повністю виконана у двовимірному стилі та написана на ігровому рушії Unity.

Сюжет розповідає про лісовий дух Орі, який є невеликим листочком величезного духу-дерева, що захищає чарівний ліс. Якось штормовий вітер відриває Орі і забирає його далеко в ліс. Впавши на землю він перетворюється на невелику білу істоту і знаходить Нару — духа, що живе у лісі і схожого на великого чорного ведмедя, – яке стає для Орі матір'ю.

Згодом, темний дух Куро, краде серце Древа, і ліс починає стрімко в'янути. Незабаром після цього Нару так само гине, і Орі залишається один. Він пускається в мандрівку по лісу що гине, але незабаром теж вмирає. Але Дерево, поряд з яким Орі закінчує своє життя, використовує сили, що залишилися, воскрешає його і просить очистити чарівний ліс від тьми.

Рисунок 1.3 — Приклад ігрового процесу гри “Ori and the Blind Forest”

Гравець керує невеликою білою істотою Орі та невеликим лісовим духом Сейном, який його захищає. Ігрова механіка представляє собою типовий платформер, гравець переміщається великою картою долаючи різні перешкоди і

бореться з різними ворогами. У грі також є проста система прокачування у вигляді невеликого дерева умінь. Для покупки нових умінь потрібно збирати невеликі згустки енергії, розкидані по світу гри. Карта світу завантажується відразу і повністю, та гравець сам вибирає, куди піти. Єдиним обмеженням у данному випадку є необхідність певних здібностей для проходження певних ділянок. Гра відрізняється неймовірно красивим оформленням, що, безсумнівно, виділяє її серед інших ігор цього жанру. За чудовим сюжетом, що розповідається в грі, цікаво спостерігати, а звуковий супровід відмінно задає необхідну атмосферу. Мінусом можна вважати лише дещо ускладнений геймплей через неінтутивне управління, але в усьому іншому, цю гру можна вважати еталоном жанру.

Mark of the Ninja [8] — двомірний стелс-екшен, розроблений канадською студією Klei Entertainment для Xbox 360 та ПК, випущена в сервісах Xbox Live Arcade та Steam. Ця гра не є чистим платформером, але має велику кількість атрибутів даного жанру.

Сюжет розповідає про безіменного ніндзя, який носить звання Чемпіона клану Хісомі, оскільки отримав особливе татуювання, яке посилює його здібності, але натомість поступово зводить його з розуму. Він намагається звільнити членів його клану, які вижили, які потрапили в полон після атаки загону Гессен.

Рисунок 1.4 — Приклад платформера “Mark of the Ninja”

Геймплей гри заснований на потайному проходженні рівнів, уникнення контактів з противниками та знешкодження пасток. Головний герой уміє ховатися у вентиляції, каналізаційних люках та за предметами інтер'єру.

Основна зброя — меч-ніндзято, яким можна виконати безшумне вбивство нічого не підозрюючої жертви та бамбукові дротики, призначені для відволікання та знищення джерел світла. Незважаючи на наявність смертельної зброї, гра рівною мірою орієнтована як на усунення всієї охорони рівня, так і на безкровне проходження. У процесі проходження купується та відкривається додаткове спорядження атакуючої та відволікаючої дії.

Практично за кожну дію нараховуються бали, на які можна відкрити нові прийоми, придбати нове або покращити старе спорядження. Додатково на кожному рівні видається 3 різні бонусні завдання. Після виконання завдань одного типу стає доступним костюм, спеціалізований для певного стилю проходження.

Також на кожному рівні, у прихованих місцях можна знайти артефакти, які дадуть вам кілька сотень балів. Ще на рівні є різні головоломки, які називаються Кімнатами випробувань. Після проходження головоломки ви отримуєте 1 з 3 прихованих на рівні сувоїв Хісомі, який також дає додаткові бали. В основному головоломка полягає в тому, щоб пройти кімнату, уникнувши пасток.

Ця гра є відмінним стелс-екшеном серед двовимірних ігор подібного жанру, яких не так багато виробляється останнім часом. Має стильне графічне оформлення, яке задає атмосферу всієї історії. Мінімалістичний та інформативний інтерфейс, але дещо перевантажене керування, яке може відбити бажання грати далі.

Dust An Elysian Tail [5] — екшен-РПГ з елементами платформера, розроблена незалежним розробником Діном Додріллом та опублікована Microsoft Studios. Ця гра так само не є чистим платформером.

Dust відбувається у вигаданому світі Фалани, населеному антропоморфними тваринами. Гравець управляє почесним головним героєм

Дастом, який намагається згадати своє минуле. Надалі йому доведеться дізнатися таємницю своєї появи, а також врятувати свій світ від тиранії Генерала Гайя. Даст має легендарний меч, Лез Аарах. Фіджет, Зберігач меча, виступає як компаньйон Даста і може використовувати магічні атаки. Поки гравець та його компаньйон подорожують світом, вони можуть придбати бонуси, які постійно змінюватимуть геймплей, такі як можливість подвійного стрибка або вхід на раніше недоступні території. Що включає елементи рольових ігор, Даст може отримувати бали досвіду, перемагаючи ворогів, і в свою чергу підвищувати рівень. Вони можуть бути використані, щоб підняти різні атрибути, такі як здоров'я, сила, броня чи магія.

Рисунок 1.5 — Ігровий процес гри “Dust An Elysian Tail”

Геймплей стандартний для платформерів — гравець пересувається по рівнях, долаючи перешкоди, але варто розуміти, що на відміну від попередніх двох ігор, тут найважливішим аспектом є розвиток персонажа та вбивство ворогів. Через це зрозуміло, що дана гра скоріше підвид платформера, який зветься метроідванія.

Limbo [7] — гра жанру пазл-платформер, створена компанією Playdead.

Як такого сюжету тут немає, все, що відомо гравцеві, це мета гри — дійти до загадкової дівчинки. Геймплей є типовим платформером з елементами пазла — гравцеві потрібно дістатися від початку рівня до його кінця уникаючи пасток і ворогів, попутно вирішуючи прості просторові головоломки. Гра виконана в стилістиці мінімалізму, і в ній переважають переважно темні тони, тому пастки не завжди помітно і гравцеві легко загинути.

Рисунок 1.6 — Сцена з гри “Limbo”

Основною проблемою гри є її власна стилістика, яка хоч і створює потрібну атмосферу, в деяких випадках може дуже сильно завадити при грі. Так само геймплей для багатьох людей може здатися нудним і нецікавим, що так само позначається на ставленні до гри. Таким чином, зрозуміло, що гра з самого початку не розрахована на широку аудиторію.

DuckTales Remastered [4] — платформер, розроблений компанією Capcom є ремайстром (перевиданням) старого платформера DuckTales.

Сюжет гри досить простий і, насправді, є просто каталізатором для початку

гри. Гравець бере під своє управління селезня Скруджа МакДака, яким рухає спрага заволодіти скарбами та стати найбагатшим селезнем у світі.

Геймплей типовий для платформерів. Гравець має подолати всі перешкоди та ворогів, що б дістатися до кінця рівня. Наприкінці гравця зустрічає бос, якого потрібно перемогти. По рівнях розкидані різні предмети, одні з яких збільшують загальний рахунок, інші здатні відновити здоров'я. Стилістика гри, порівняно з оригіналом перероблена, тепер у грі лише персонажі та вороги виконані у вигляді двовимірної графіки, в той час як оточення та предмети створені у вигляді тривимірних моделей.

Рисунок 1.7 — Ігровий процес з гри “DuckTales Remastered”

Гра виконана добре, відмінне графічне виконання та інтуїтивне керування, зрозумілий інтерфейс. Мінусом можна було б назвати тільки застарілу механіку, все ж таки за стільки часу було придумано безліч інших цікавих геймплейних особливостей.

Wings of Vi [10] — платформер, розроблений та випущений студією Grynsoft.

Сюжет нехитрий і типовий для подібних ігор. Якось Ві, з другом Рубі попрямували до сирої в'язниці, де було ув'язнено Зло. З якихось причин темниця не мала охорони, а після приходу «непосид», вони виявили (завдяки подружці садистці), що ланцюги розірвані, а двері де знаходилося Зло вибиті. Зла сутність вирвалася щоб сіяти розлад і смуту, по всьому білому світу. Героям Wings of Vi належить виправити виявлену ними проблему і вирушити в дорогу підземеллями, містами і небесами, за слідом великого Зла.

Геймплейно гра не відрізняється від перших платформерів, гравцю так само необхідно подолати безліч різних перешкод та ворогів, що б пройти рівень від початку до кінця. Наприкінці рівня гравця зустріне бос, якого потрібно вбити, щоб пройти далі. Відмінною особливістю цієї гри є її висока складність. Більшість ворогів завдають великої шкоди персонажу, а рівні спроектовані таким чином, щоб гравцеві спочатку було складно пройти гру. Такого виду ускладнення, ні в якому разі не робить гру гіршою, а скоріше навіть навпаки. Графічно гра виконана у стилі піксель-арт (стиль, при якому зображення малюються попіксельно, для досягнення ефекту ретро ігор).

Рисунок 1.8 — Ігровий процес платформера “Wings of Vi”

Складність у цій грі виступає як гідністю, так і недоліком. Через те, що у гри високий поріг входження, не кожен зможе її пройти, а значить цей продукт є швидше нішевим, ніж призначеним для широкого кола користувачів.

Провівши аналіз існуючих розробок, стало зрозуміло, що чистих платформерів у їхньому первісному вигляді стало дуже мало, тому що вони перестали відповідати збільшеним вимогам гравців. Чистими платформерами на на даний момент є в основному перевидання старих популярних ігор, котрі грають більше на ностальгії гравців.

Проте, не можна сказати, що популярність платформерів як жанру серед гравців сильно впала. Справді, ігор відповідних саме цій жанрової категорії зараз не так багато, але водночас почало виходити більше ігор, які можна зарахувати не лише до платформерів, а й до інших жанрів. Це свідчить, що жанр «Платформер» став більш нішевим продуктом, ніж масовим, отже щоб одержати великих прибутків, розробникам доводиться додавати елементи сторонніх жанрових категорій відеоігор.

2 ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ДЛЯ РОЗРОБКИ ГРИ

2.1 Unity

Unity — це універсальний кросплатформовий ігровий рушій, розроблений Unity Technologies, вперше анонсований тільки для OS X для Всесвітньої конференції розробників Apple у 2005 році, він був розширений до 27 платформ. Програми створені за допомогою Unity3D підтримують обидві специфікації 3D графіки DirectX та OpenGL. На рисунку 2.1 представлена графічна оболонка Unity3D.

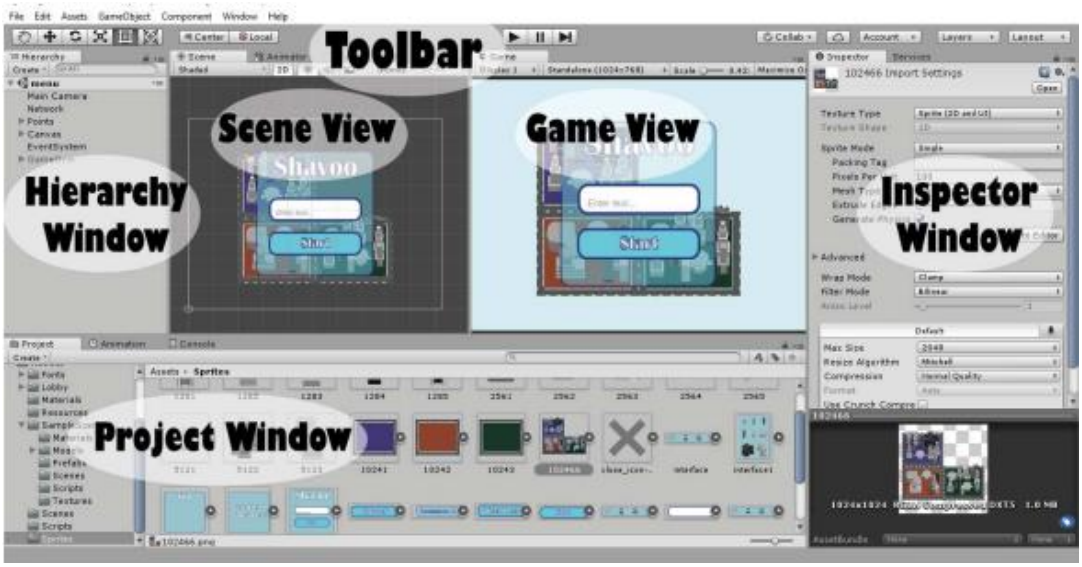


Рисунок 2.1 — Інтирфейс Unity

Project Window (Оглядач проекту). Ця частина інтерфейсу служить для роботи з ресурсами, які знаходяться в ігровому проекті.

У лівій частині оглядача міститься ієрархічний список, який відображає структуру папок проекту, окремі ресурси представлені у вигляді іконок, що вказують їх тип (спрайт, скрипт і т.д.).

Toolbar (Панель інструментів). У цій частині розташовуються елементи, необхідні трансформації, кнопки запуску і зупинки гри, випадають меню, шари.

Hierarchy Window (Ієрархія). Вікно містить усі об'єкти, що знаходяться на сцені. Це можуть бути будь-які UI елементи (кнопки, картинки тощо), 3D моделі, екземпляри префабів (об'єктів користувача) та інше. Кожен об'єкт (GameObject) може містити дочірні об'єкти або входити до об'єктів вищого рангу. Можна вибрати об'єкт в ієрархії та перемістити його в інший об'єкт, таким чином сформується батьківський зв'язок (parenting). Дочірній об'єкт успадковує зміни свого батьківскього об’єкта, пов'язанний з переміщенням, обертанням та масштабуванням.

Scene View (Вигляд сцени). В цьому вікні створюватиметься рівень для своєї гри, сцени або 3D-проекту. Всі ігрові об'єкти будуть розміщуватися та керуватися прямо тут.

Game View (Вигляд гри). В данному вікні можна побачити свої результати, як виглядає рівень чи сцена. Для цього повинна бути камера на сцені, щоб побачити, як це виглядає.

Inspector Window (Інспектор). Він відображає інформацію про конкретний вибраний об'єкт та його властивості. Тут можна змінювати функціонал об'єктів у сцені.

2.2 Середовище розробки Microsoft Visual Studio

Visual Studio — це інтегроване середовище розробки (IDE), яке використовується для розробки консолі, графічних програм інтерфейсу користувача, додатків Windows Forms або WPF, веб-додатків, веб-сайтів, веб-сервісів тощо. З VS розробники також мають доступ до безлічі інструментів для налагодження. Вони допомагають їм профілювати помилки та просто діагностувати їх. Таким чином, вони можуть впевнено розгортати свої програми, знаючи, що вони покінчили з усім, що може спричинити помилки продуктивності. Більше того, Visual Studio IDE також служить платформою для тестування. Саме тут розробники можуть імітувати, як їхні програми

працюватимуть у цільових середовищах, і забезпечити безперебійну роботу після розгортання.

Головними перевагами Visual Studio є:

— точне кодування, з Visual Studio IDE користувачам надається оперативна допомога в кодуванні незалежно від мови програмування, яку вони використовують, а також вбудований у платформу IntelliSense пропонує підказки та описи API та автозавершує рядки для кращої швидкості;

— швидке налагодження, пошук і діагностика помилок може бути складним завданням, але Visual Studio IDE має безліч інструментів, які полегшують роботу, платформа підтримує налагодження для всіх включених мов, процес також можна виконувати локально, віддалено та в середині виробництва, це дозволяє розробникам розгортати програми на робочому столі або емуляторі на мобільних пристроях та інші методи налагодження;

— ретельне тестування, Visual Studio IDE обладнана платформою для тестування програм, яка дозволяє розробникам переконатися, що вони готові до релізу високоякісних продуктів, вони можуть робити це на своїй бажаній мові та тестовій структурі, роблячи це з невеликими зусиллями, таким чином, вони можуть направити більше своєї енергії на інші фази розвитку;

— командна співпраця, розробники Visual Studio розуміють, що більше пар очей краще, ніж одна, а саме тому платформа має можливості спільної роботи, які підвищують продуктивність команди, ці інструменти тісно інтегровані з життєвим циклом розробки, крім того, VS добре працює в режимі співпраці незалежно від бажаної платформи кожного члена;

— параметри налаштування, Microsoft Visual Studio пропонує можливості налаштування кожному користувачеві, вони можуть розширювати функціональні можливості платформи за допомогою розширень і доповнень, доступних на Visual Studio Marketplace, розробники навіть можуть публікувати власні розширення.

Microsoft Visual Studio має небагато конкурентів, таких як IntelliJ IDEA, Eclipse, NetBeans, PhpStorm та Xcode тощо

На рисунку 2.2 представлений інтерфейс роботи програми.

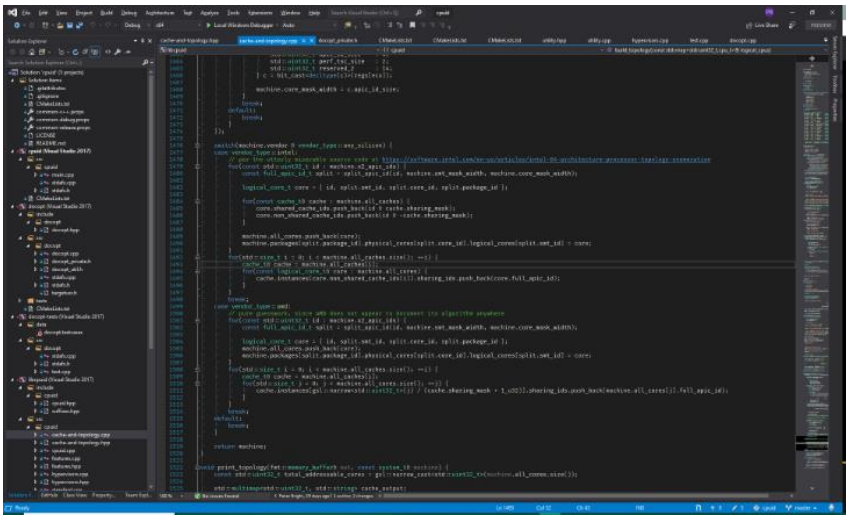


Рисунок 2.2 — Інтерфейс середовища розробки Microsoft Visual Studio

2.3 Мова програмування C#

C# — це строго типізована об'єктно-орієнтована мова програмування. C# є відкритим вихідним кодом, простим, сучасним, гнучким і універсальним. Він розроблений та запущений компанією Microsoft у 2001 році. C# — це проста, сучасна й об’єктно-орієнтована мова, яка надає сучасним розробникам гнучкість та можливості для створення програмного забезпечення, яке не тільки працюватиме сьогодні, але й буде застосовуватися протягом багатьох років.

2.3.1 Головні переваги C#

Метою C# була розробка мови програмування, яку не тільки легко вивчити, але й підтримувати сучасні функціональні можливості для всіх видів розробки програмного забезпечення. Мова C# була розроблена для компаній, щоб створювати всі види програмного забезпечення за допомогою однієї мови програмування. C# надає функціональні можливості для підтримки сучасної

розробки програмного забезпечення. Він підтримує потреби в розробці веб, мобільних пристроїв і програм. Деякі з функцій сучасної мови програмування, які підтримує C#, — це генерики, типи змінних, автоматична ініціалізація типів і колекцій, лямбда-вирази, динамічне програмування, асинхронне програмування, кортежи, зіставлення шаблонів, розширене налагодження та обробка винятків тощо. Синтаксис мови C# перебуває під впливом C, Java, Pascal та кількох інших мов, які легко сприймати. C# також уникає складності та неструктурованих мовних функцій.

C# є відкритим вихідним кодом у рамках .NET Foundation, який керується та працює незалежно від Microsoft. Специфікації мови C#, компілятори та пов’язані з ними інструменти є проектами з відкритим кодом на Github. Хоча дизайн функцій мови C# очолює Microsoft, спільнота з відкритим кодом дуже активна в розробці та вдосконаленні мови.

C# швидкий у порівнянні з кількома іншими мовами програмування високого рівня. C# 8 має багато покращень продуктивності.

C# — це кросплатформна мова програмування. Ви можете створювати програми .NET, які можна використовувати на платформах Windows, Linux і Mac. Програми C# також можна загрузити в хмарі та в контейнерах.

C# — це безпечна мова. Вона не дозволяє перетворення типів, які можуть призвести до втрати даних або інших проблем. C# дозволяє розробникам писати безпечний код. А також фокусується на написанні ефективного коду.

2.3.2 Сценарії в Unity

Unity використовує компонентний підхід. Компонент — це клас, успадкований від MonoBehaviour. Один компонент повинен відповідати за поведінку.

Скрипти можуть бути створені на панелі проектора. Для цього достатньо натиснути кнопку. Create і вибрати мову, якою буде створюватися скрипт.

Прикріпити скрипт до об'єкта можна шляхом перетягування або натискання кнопки Add Component.

Після того, як буде створено скрипт в Unity і відкриється за замовчуванням у Visual Studio, можна буде побачити код, представлений у лістингу 1.1.

Лістинг 2.1 — Представлення коду за замовчуванням

```
using UnityEngine;
using System.Collections;
public class MainPlayer : MonoBehaviour {
 // Use this for initialization
 void Start () {
 }
 // Update is called once per frame
 void Update () {
 }
}
```

Назва створеного скрипта має повністю відповідати назві створеного автоматично класу, який успадковується від вбудованого класу MonoBehaviour.

Функція Update — використовується в Unity. Вона викликається один раз за кадр у кожному скрипті, який використовує її. Майже все, що потребує змін або регуляції на постійній основі, прописується в цій функції. Переміщення нефізичних об'єктів, прості таймери та виявлення введення зазвичай реалізуються у цій функції. Варто відзначити, що ця функція не викликається на основному таймлайн. Якщо один кадр займає більше часу для обробки, ніж наступний, час між викликами функції буде різним.

Функція Start — функція, яка викликається автоматично перед функцією Update, коли скрипт завантажено. Функцію можна використовувати для всього, що потрібно лише тоді, коли компонент скрипта увімкнено. Це дозволяє відстрочити будь-яку частину коду ініціалізації, доки вона не знадобиться.

3 РЕАЛІЗАЦІЯ СТВОРЕННЯ КОМП’ЮТЕРНОЇ ГРИ

3.1 Етап розробки графічного оформлення

Варто взяти до уваги, що основні етапи реалізації проекту, а саме розробки графічного оформлення та розробка гри, проходили паралельно одне одному. Цьому сприяв той факт, що для написання програмного коду необов'язкова наявність готових анімацій та іншого, а для перевірки працездатності самого коду достатньо стандартних примітивів Unity. Також такий спосіб реалізації проекту значно збільшує швидкість роботи.

Насамперед, варто виділити основні поняття, такі як тайл (Tile) та спрайт (Sprite).

Тайл — невеликих розмірів фрагмент, що повторюється, який служить для створення зображень великих розмірів, так звана Тайлова графіка. Часто використовується для створення рівнів для двовимірних ігор.

Спрайт — графічний об'єкт, що представляє собою растрове зображення. Використовується в комп'ютерній графіці як одиниця для анімацій двовимірних об'єктів.

Для реалізації графічного оформлення було взято вже готові набори 2D спрайтів з Unity Asset Store, зі всіма необхідними елементами для гри. А саме анімації рухів для головного персонажа та супротивників на рівні, а також фонове зображення, що відображається позаду рівня. Спрайти персонажа, супротивника та фон зображені на рисунках 3.1-3.3.

Рисунок 3.1 — Фонове зображення

У якості головного героя було обрано лучницю. Вона складається з 24 спрайтів героя та 2 спрайтів стріл, які призначені для анімації бігу, стрибків, стрільби з луку та простою.

Рисунок 3.2 — Спрайти для головного персонажу

Як ворога було обрано вовка, який має 18 спрайтів для анімації дихання, ходьби та кусання.

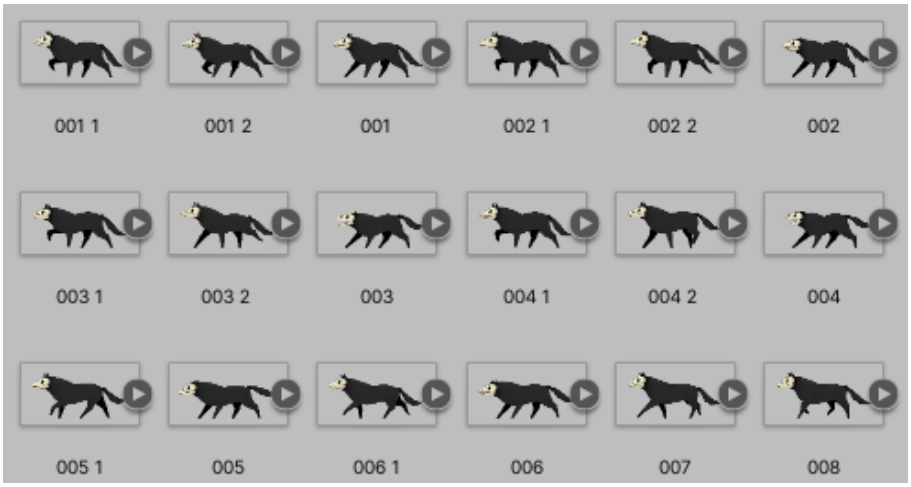


Рисунок 3.3 — Спрайти для вовка

Інтерфейс користувача в середовищі комп'ютерних розробок носить назву UI. UI — це все інформаційне середовище, яке відображається на екрані, і з яким взаємодіє гравець. Раніше для створення UI потрібно писати велику кількість

скриптів. На даний момент створено об'єктну модель створення інтерфейсів за допомогою візуалізації.

Для створення будь-якого UI елемента в Unity необхідно клікнути правою кнопкою миші у вікні ієрархії, вибрати UI, і, відповідно, вибрати потрібний графічний елемент. На рисунку 3.4 представлені види UI у Unity.

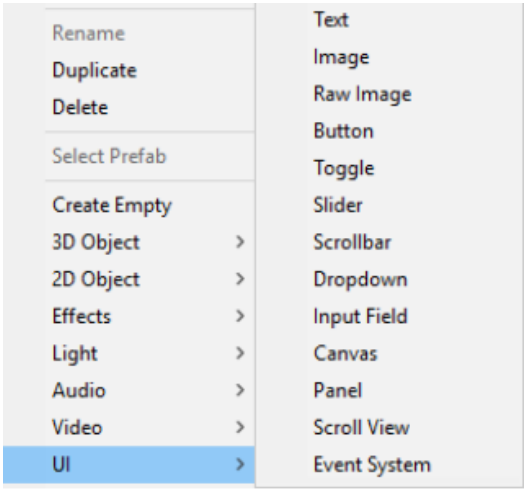


Рисунок 3.4 — UI в Unity

Будь-який графічний елемент створюється дочірнім об'єктом Canvas. Об'єктна модель створення інтерфейсу в Unity передбачає, що всі елементи мають бути дочірніми у Canvas.

Canvas (Полотно) — це компонент, який відображає графічні елементи (картина, текст, кнопки та інше). Він автоматично змінюється в залежності від роздільної здатності екрану. Canvas зображений на рисунку 3.5

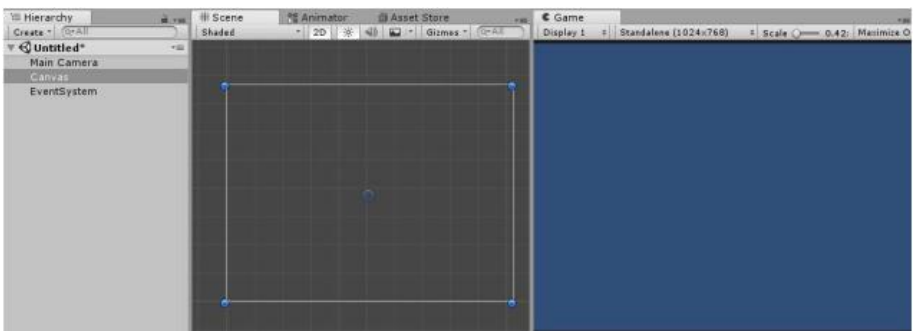


Рисунок 3.5 — Canvas

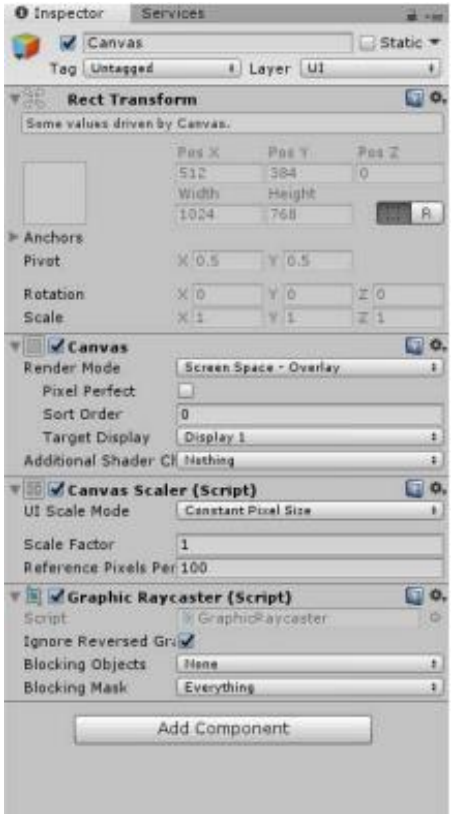


Рисунок 3.6 — Властивості Canvas

Render Mode — властивість, яка дає можливість налаштувати, як графічні елементи, що знаходяться всередині Canvas, повинні відображатися на екрані.

Screen Space Overlay — всі графічні елементи відображаються поверх ігрової сцени. У цьому режимі розміри полотна автоматично підлаштовуються до розмірів екрана.

Screen Space Camera — всі графічні елементи відображаються за допомогою камери. Для цього потрібно помістити елемент Camera як Render Camera. Змінюючи параметри камери, можна впливати на зовнішній вигляд Canvas.

World Space — всі UI знаходяться в 3D просторі та вважаються звичайними 3D об'єктами.

Будь-який створений на Canvas графічний елемент можна прив'язати до однієї з дев'яти точок (до кутів, середин сторон або центру екрана). Для цього в

панелі Rect Transform (рисунок 3.7) використовується Anchor Presets (рисунок 3.8).

Рисунок 3.7 — Rect Transform

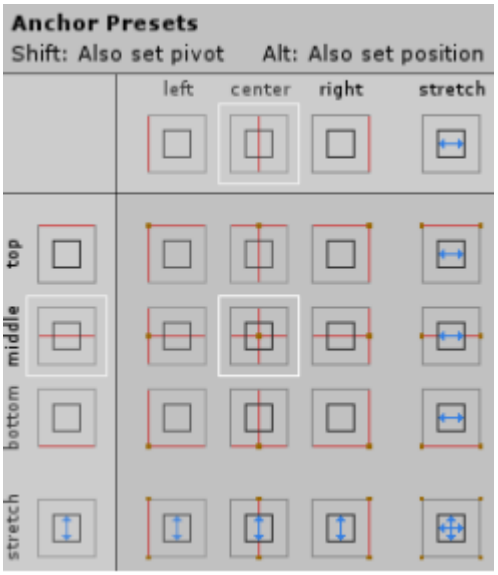


Рисунок 3.8 — Anchor Presets

Координати розташування елемента будуть відштовхуватись від точки, яка обрана як "якор"

Також у будь-якого елемента в панелі React Transform є властивість Privot - це та точка на самому елементі, від якої будуть відштовхуватись її координати.

За допомогою елемента Image можна розташувати будь-яке зображення на екрані. За замовчуванням об'єкт картинки створюється завжди того самого розміру, але за допомогою кнопки Set Native Size в інспекторі Image можна підігнати розміри зображення під розмір спрайту.

Text є елементом, який відображає текст, який можна задавати в спеціальному полі або в скриптах.

Button є складовим елементом, тому текст усередині кнопки редагується в дочірньому до кнопки елементі Text. На кнопку можна натискати та обробляти це натискання. На компоненті Button є властивість, що дозволяє налаштовувати функції, які будуть оброблятися за натисканням на кнопку. Щоб кнопка розуміла, яку функцію викликати, ця функція описується в окремому скрипті, при цьому ця функція має бути public. Подію можна налаштовувати як на натискання кнопки (On Click ()), так і на відпускання кнопки (додатковий компонент Event Trigger).

За допомогою елемента Slider можна робити будь-які налаштування, наприклад, гучність звуків або яскравість екрана. Елемент також є складовим: що складається з простіших. Завдяки вбудованій властивості, яка дозволяє слайдеру зафарбовувати ту область, яка знаходиться лівіше за повзунок, його дуже часто використовують в іграх для організації в інтерфейсі таких об'єктів як смужку стану здоров'я або мани.

Toggle — складний елемент, який складається з кількох картинок та тексту. Текст вказується в дочірньому елементі Text, а зображення вказується в елементі Image. Цей елемент має лише одну подію, яка спрацьовує при зміні стану об'єкта.

Елемент Scrollbar майже те саме, що Slider, тільки використовується у складі складніших компонентів, щоб взаємодія зі Scrollbar давала який-небудь ефект.

Dropdown є складовим і являє собою список, що розкривається. Елементи списку задаються у налаштуваннях самого компонента.

Розроблений проект містить каталоги, у яких зберігаються:

— шаблони об'єктів;

— ігрові сцени;

— скрипти;

— зображення персонажів, фонів;

— анімація персонажа.

Файлова структура представлена рисунку 3.9.

У директорії Prefabs є готові шаблони ігрових об'єктів, за допомогою яких можна скласти ігровий рівень після його проектування. У папці Audio зберігаються музика так звуки для гри. Директорія Scenes містить усі сцени гри, в які може перейти користувач. У папці Scripts знаходяться скрипти з описом всіх класів та взаємодій. У каталозі Sprites містяться шість підкаталогів:

— Background містить у собі базові блоки платформ (ліва, центральна, права) та стіни рівня;

— Character містить у собі спрайт головного персонажа та кулю, якою здатний стріляти гравець;

— Coin містить спрайти для монет;

— Menu містить елементи для меню гри;

— Wolf містить у собі спрайти супротивника вовка;

— Health містить табличку зі здоров'я персонажа.

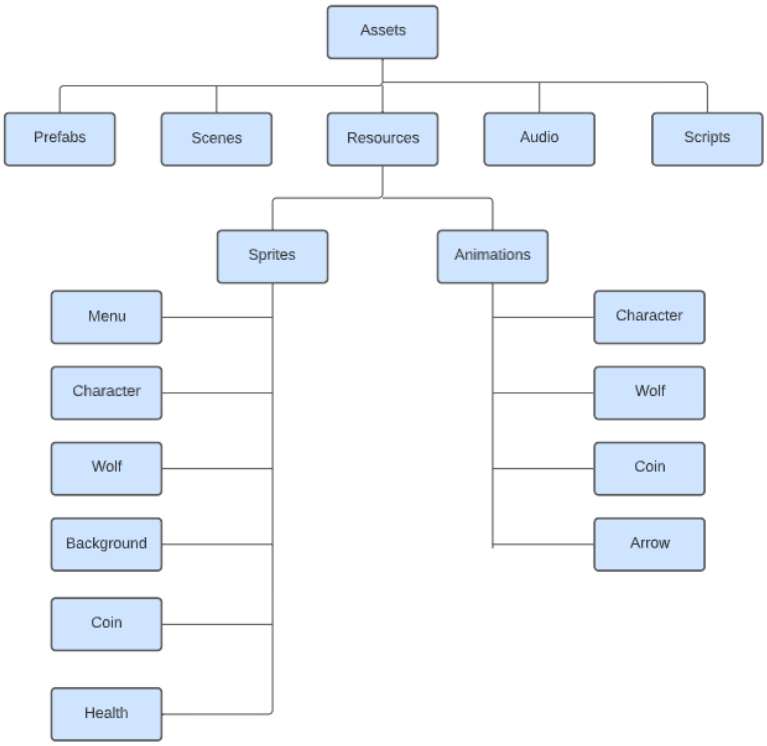


Рисунок 3.9 — Файлова структура

3.2 Етап розробки гри

Для реалізації гри буде створено дві сцени: Menu та Level_1. Menu міститиме 3 кнопки: початок гри, інформацію про гру та вихід з гри. Level_1 — основна сцена, в якій відбуватимуться всі ігрові дії.

Перемістимо створені сцени у Scenes in build, як на рисунку 3.10

Рисунок 3.10 — Параметри складання сцен

Тепер перейдемо до формування самих сцен. Відкриємо сцену Menu і створимо Canvas (полотно), на який розміщуватимемо елементи інтерфейсу (рисунок 3.11).

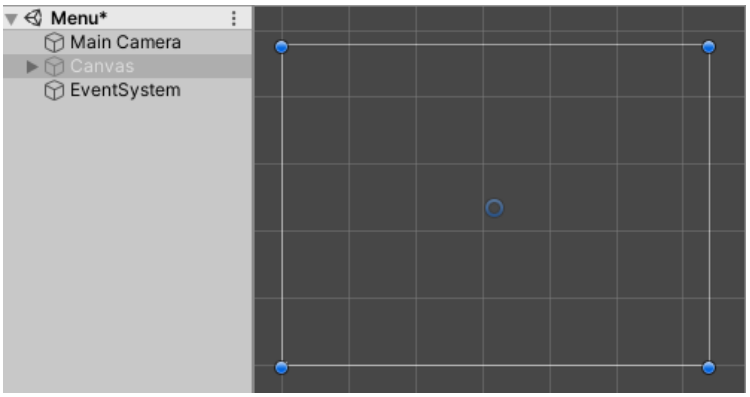


Рисунок 3.11 — Створення Canvas у сцені Menu

У цьому завданні ми будемо використовувати властивість Render mode — Screen Space — Overlay.

Налаштуємо колір панелі. Можна також додати зображення на фон. Для цього потрібно звернути увагу на компонент Image в панелі Інспектора.

Тепер із папки Menu перемістимо раніше створені елементи інтерфейсу на нашу сцену.

Сцена Menu повинна містити наступне:

— назву гри;

— кнопку New Game, для початку гри;

— кнопку About яка містить інформацію про гру;

— кнопку Exit для виходу з гри;

— зображення фону.

Створимо дочірні UI об'єкти для Canvas: одне зображення (Background), який має додатковий елемент Text, що відповідає за назву гри, 3 кнопки (StartButton, AboutButton, ExitButton), а також об’єкт AboutMenu типу Image, в якому зберігаєтються елементи тексту та кнопок (рисунок 3.12).

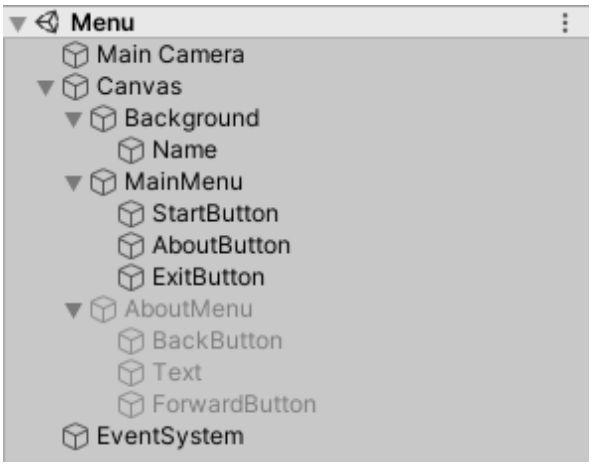


Рисунок 3.12 — Дочірні об’єкти Canvas в Menu

Елемент Background відповідає за вигляд заднього фону, тому помістим в нього це зображення. Назва гри “My Way”, міститься в об’єкті Name типу Text. Також до кнопок було додано спрайти уже з текстом, тому в них не має елемента Text. AboutMenu являється зображенням поверх головного, на якому розміщено дві кнопки та текст.

На рисунку 3.13 зображено підсумковий вид сцени Menu.

Рисунок 3.13 — Результат виконання сцени Menu

Насамперед для сцени Level_1, нам необхідно додати платформу, якою буде переміщатися наш герой та його вороги. Без неї гравець просто буде нескінченно падати у просторі. Спочатку нам необхідно додати сам спрайт платформи, який було створено раніше, для цього досить просто перетягнути його з папки у вікно проекту, після чого він має з'явитися серед усіх об'єктів уже доданих у гру. Після цього нам потрібно додати нашу платформу до поточної сцени. Це можна зробити двома способами: перетягнути з вікна Проекту до вікна Сцени, або спочатку у вікні Ієрархії створити окремий порожній об'єкт, до якого згодом і застосувати спрайт платформи (рисунок 3.14).

Після цього нам необхідно додати до об'єкту колайдер. Він необхідний для того, щоб рушій розумів межі об'єкта і гравець міг з ним взаємодіяти. У Unity існує два види колайдерів: для тривимірних та двовимірних об'єктів. Так як створювана гра двовимірна, то і коллайдер потрібно додавати двовимірний. Щоб додати колайдер до об'єкта, необхідно виділити його у вікні ієрархії, а потім додати компонент колайдера у вікні інспектора. Для налаштування розмірів коллайдера необхідно скористатися кнопкою Edit Collider [15].

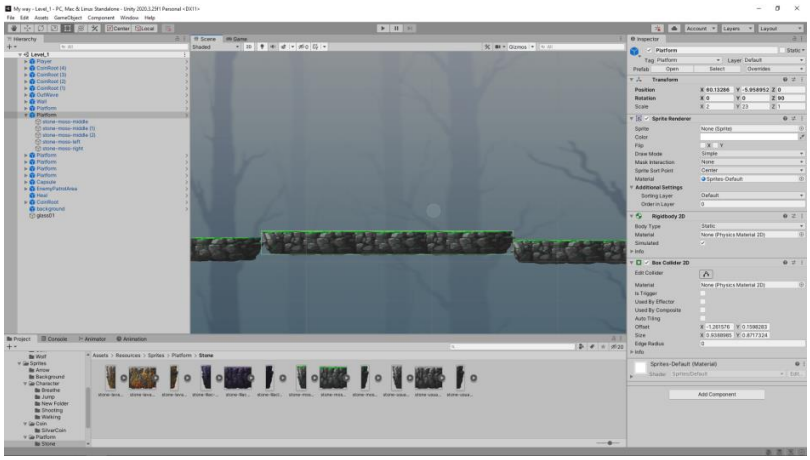


Рисунок 3.14 — Платформа у вікні сцени та її властивості

Unity дозволяє зберігати об'єкти, з усіма доданими властивостями та скриптами для того, щоб потім їх можна було повторно використовувати, просто перемістивши на сцену, не створюючи заново новий об'єкт. Такий об'єкт називається Префаб і для його створення досить просто перемістити створений об'єкт із вікна Ієрархії у вікно Проекту. Для префабів має сенс створити окрему папку, щоб вони не губилися серед інших елементів гри.

Тепер створюємо головного героя для гри. Для цього також додаємо двомірний об'єкт і додаємо до нього спрайт героя. Для реалізації героя нам знадобиться коллайдер, а також компонент Rigidbody 2D, через нього налаштовуватиметься фізична модель об'єктів (рисунок 3.15).

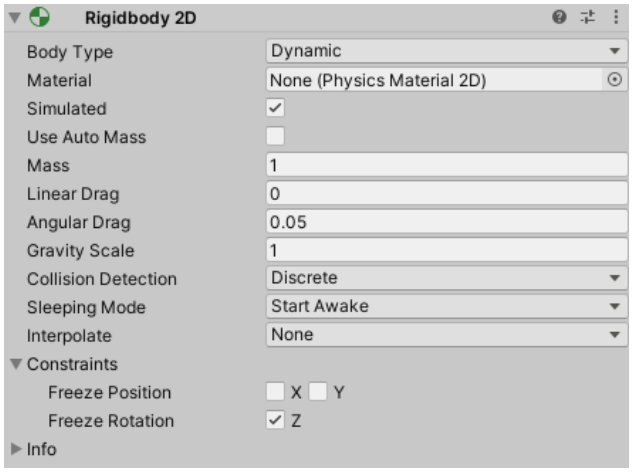


Рисунок 3.15 — Властивості Rigidbody 2D

У Unity скрипти розглядаються як об'єкти, які відповідають за поведінку. Як і інші компоненти Unity, вони можуть бути прикріплені до об'єктів.

Скрипти дозволяють реалізувати будь-які варіанти поведінки об'єкта. Скрипт — це інструмент, за допомогою якого можна вибрати поведінку будь-якого об'єкта в грі. Це можуть бути персонажі, об'єкти оточення або, наприклад, розширення функціональності ігрового геймплею. Скрипти можуть використовуватися навіть для створення графічних ефектів або реалізації штучного інтелекту.

Оскільки головний герой повинен рухатися і робити різні дії, нам необхідно створити скрипт мовою C#. Це можна зробити декількома способами: створити у вікні Проекту або створити відразу на необхідному об'єкті у вікні Інспектора.

Коли скрипт буде створено, відкриваємо його і після цього має запуститися Microsoft Visual Studio. У коді вже будуть підключено основні бібліотеки, а також створено стандартні методи Update та Start. Записуємо код для руху персонажем (рисунок 3.16). Після чого додаємо його до нашого героя.

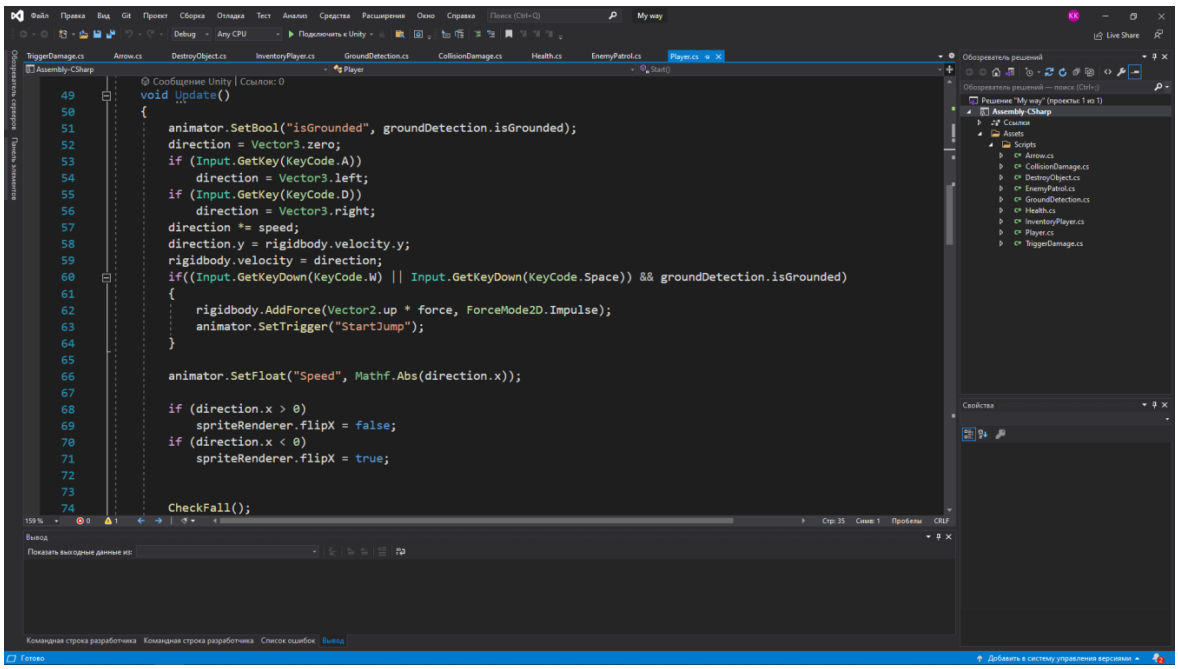


Рисунок 3.16 — Код для руху героєм

Для реалізації стрибків необхідно створити у героя скрипт. Цей скрипт назвемо GroundDetection (рисунок 3.17), за допомогою нього ми визначатимемо, перебуває персонаж на землі чи ні. Це необхідно, щоб гравець не міг нескінченно підніматися вертикально вгору при постійному натисканні клавіші стрибка.

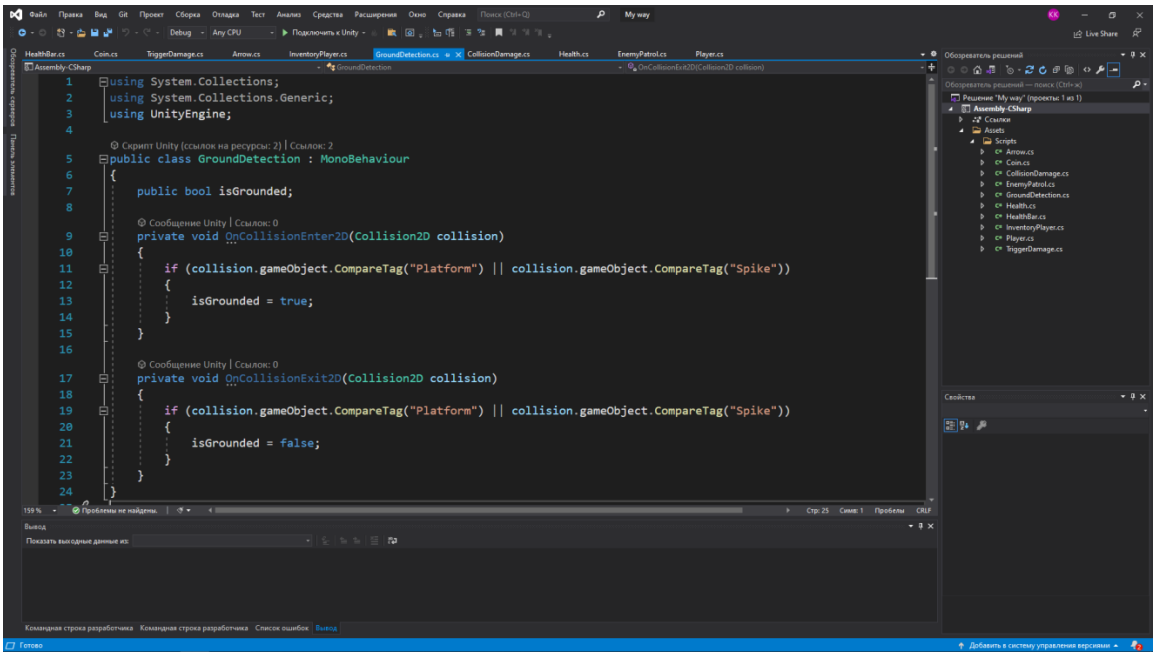


Рисунок 3.17 — Скрипт для перевірки стрибків

Тепер, якщо запустити проект, ми зможемо побігати по платформі з боку на бік.

Так само створюємо другий скрипт, який ми додаємо до камери і в ньому прописуємо умови, за якими камера слідуватиме за героєм по рівневі. Якщо ми цього не зробимо, не зможемо стежити за діями персонажа.

У грі передбачено небезпечний статичний об'єкт як кристалів. Для їх реалізації на полі був створений порожній GameObject, до якого підключені компоненти SpriteRenderer, PolygonCollider2D, CollisionDamage.

Компонент SpriteRenderer (рисунок 3.18) відповідає за двовимірне графічне уявлення ігрового об'єкта. У полі Sprite було підключено спрайт кристалів. Поле Order in Layer визначає положення спрайту кристалів щодо інших об'єктів, тим самим можна сховати частину кристалів за платформу або повісити в повітрі.

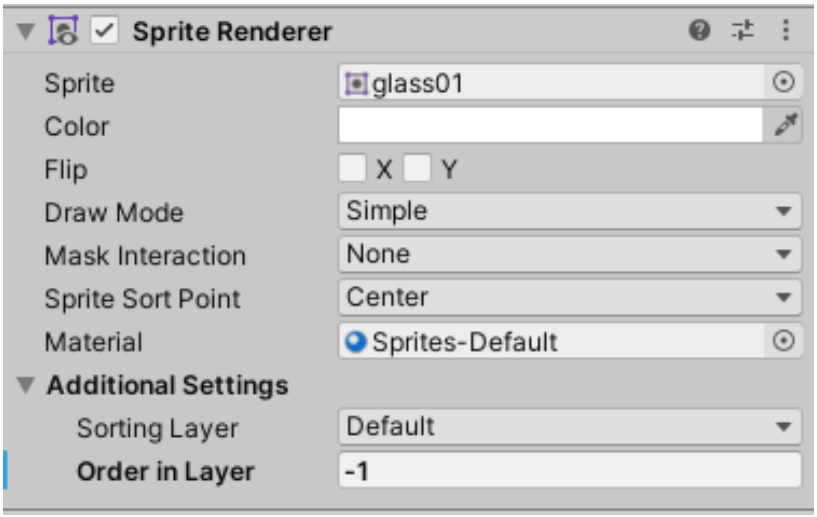


Рисунок 3.18 — Компонент SpriteRenderer

Компонент PolygonCollider2D є колайдером, форма якого визначається шляхом вільного визначення граней сегментів ліній, з яких вона складається. Тому можна налаштувати форму так, щоб Sprite був розміщений у ній з максимальною точністю.

Компонент CollisionDamage відповідає за завдання шкоди гравцеві в той момент, коли колайдер ігрового об'єкта стикається з колайдером персонажа, до якого підключений скрипт. Після чого слідує віднімання життів у головного героя.

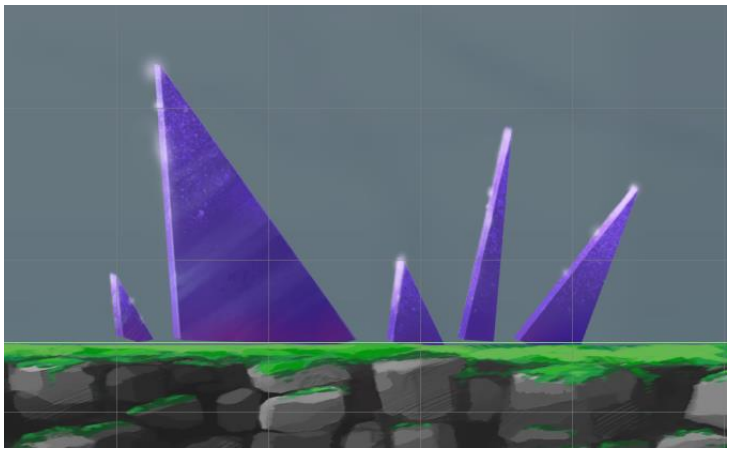


Рисунок 3.19 — Небезпечний статичний об’єкт

Тепер нам потрібно створити ворогів. Вороги створюються подібним способом, за винятком, що гравець ними управляти не може. Так само відмінною особливістю ворогів є те, що в залежності від ситуації, вони самі вибирають необхідну дію, доступне їм за правилами гри.

Так само пересування ворогів обмежені межами їхніх місць проживання. Для цього потрібно створити два порожніх об’єкти, які не буде видно під час гри, та створити скрипт (рисунок 3.20), який буде обмежувати рух вовка.

Рисунок 3.20 — Межа пересування ворога

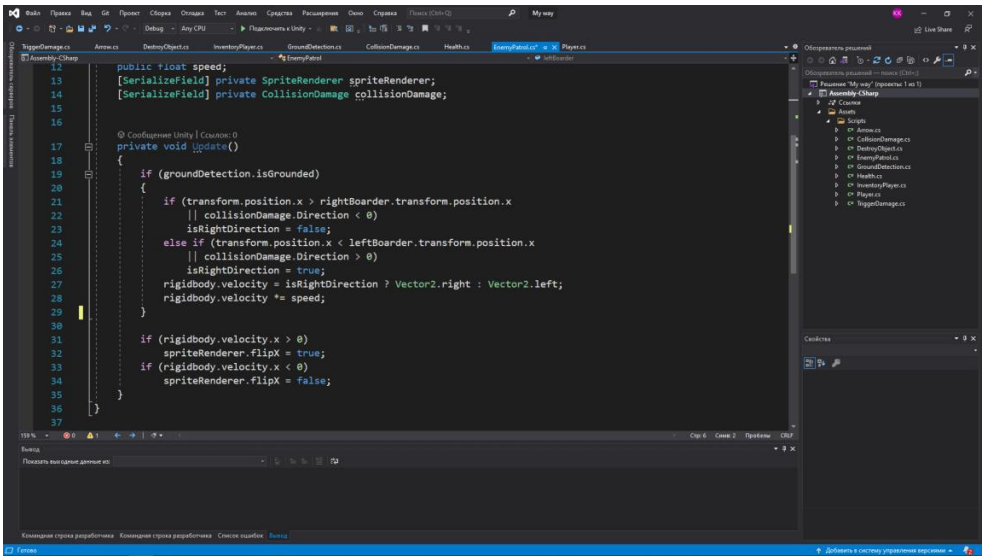


Рисунок 3.21 — Фрагмент коду сценарію поведінки ворога

Після створення скриптів для головного героя та ворогів нам необхідно створити для персонажа чотири анімації:

— простою;

— бігу;

— атаки;

— стрибка.

Для цього додаємо до проекту спрайти усіх цих анімацій. Потім створюємо Animator Controller та додаємо до нашого персонажа. Він визначає правила та зв'язки для анімацій певного об'єкта.

Для реалізації самих анімацій потрібно у вікні Animation створити новий кліп анімації, перед цим вибравши потрібний об'єкт, а потім перенести усі кадри необхідної анімації у дане вікно у правильному порядку. Тут налаштовуємо швидкість анімації (рисунок 3.22).

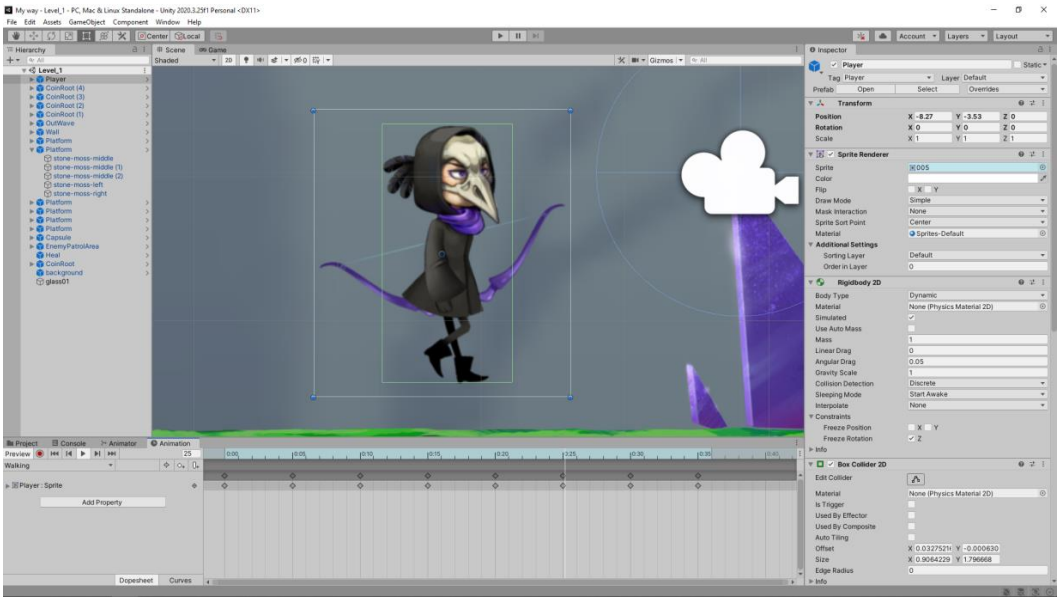


Рисунок 3.22 — Створення анімації

Після створення всіх необхідних кліпів анімацій, переходимо у вікно Аніматора (рисунок 3.23). Тут нам необхідно створити зв'язки та настроїти умови переходів між різними анімаціями. Для ворогів робимо те саме.

Якщо запустити рівень, ми зможемо побігати в різні боки, пострибати і робити інші доступні гравцеві дії.

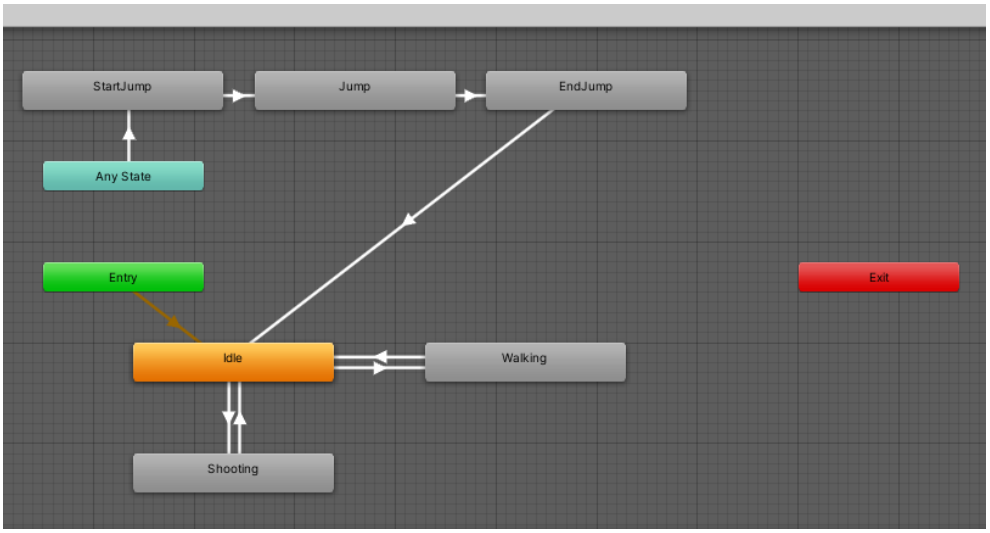


Рисунок 3.23 — Створення концептуальної моделі головного героя

Анімація стрибка розділена на три етапи: підстрибування, зависання у повітрі та приземлення. Його можна відтворити з будь-якого стану. Стрілянина проводиться тільки коли персонаж стоїть на місці. Якщо під час руху натиснути кнопку стрільби, вона буде проведена лише тоді, коли персонаж зупиниться.

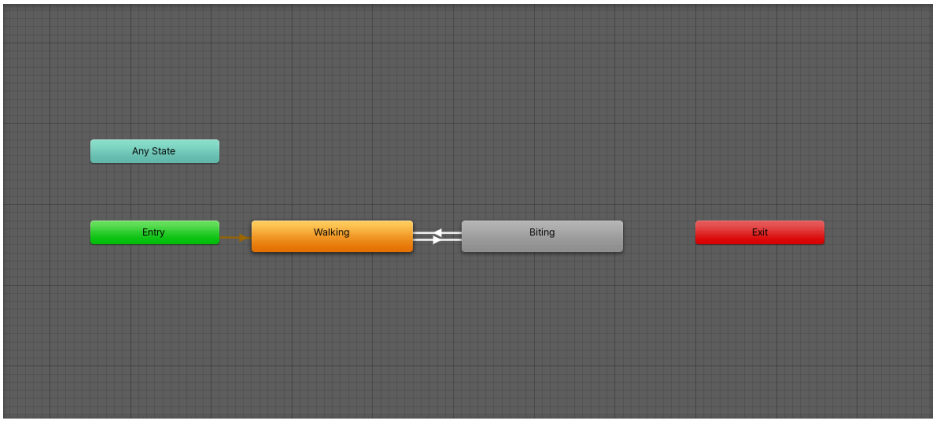


Рисунок 3.24 — Концептуальна модель вовка

Для реалізація вовка створено дві анімації: пересування та атаки. Анімація атаки відтворюється тоді, коли вовк доторкається до головного героя.

Для створення головного героя були додані необхідні об’єкти, які представлені на рисунку 3.25.

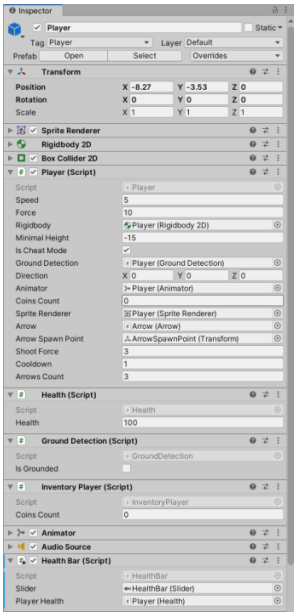


Рисунок 3.25 — Структура ігрового об’єкту Player

У кожній грі є музика, і ця не буде виключенням. Для того щоб її додати, потрібно створити пустий об’єкт і додати його до головного героя, так вона буде слідувати за ним (рисунок 3.26).

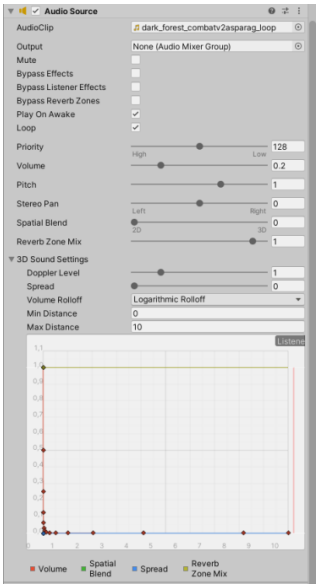
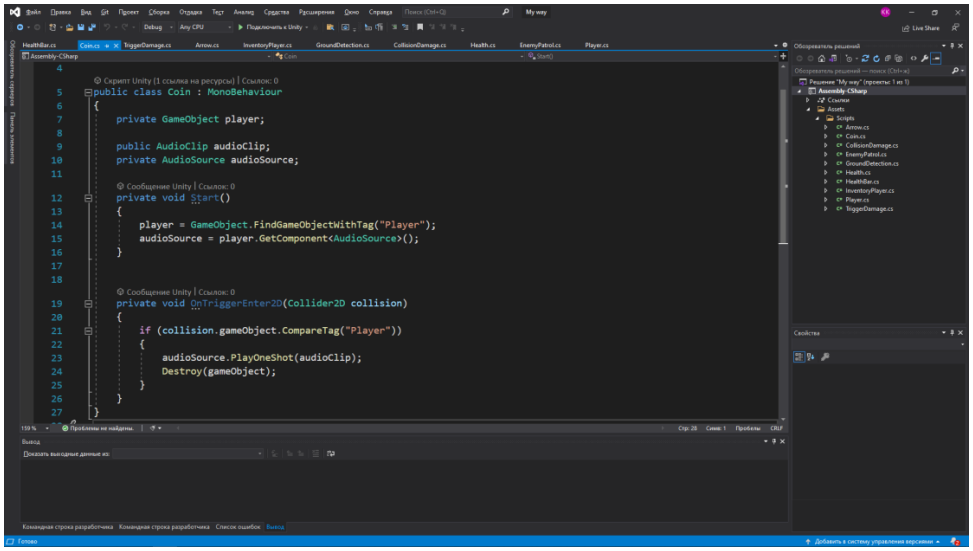


Рисунок 3.26 — Структура об’єкта програвання музики

Також при підборі монети програється аудіо кліп, для якого потрібно написати код, який представлений на рисунку 3.27.



```
public class Coin : MonoBehaviour
{
    private GameObject player;

    public AudioClip audioClip;
    private AudioSource audioSource;

    private void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        audioSource = player.GetComponent<AudioSource>();
    }


    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            audioSource.PlayOneShot(audioClip);
            Destroy(gameObject);
        }
    }
}
```

Рисунок 3.27 — Код для відтворення звука для підбору монети

Інтерфейс в іграх буває різних типів. Для даної гри був вибраний тип - недієгетичний: ці показники відображаються в кутку екрана, приховані від героїв гри та доступні лише самому гравцю.

Для реалізації відображення здоров’я персонажа, потрібно було створити порожній об’єкт в тілі Player та за допомогою властивостей Canvas було створено іконку та анімацію здоров’я (рисунок 3.28).

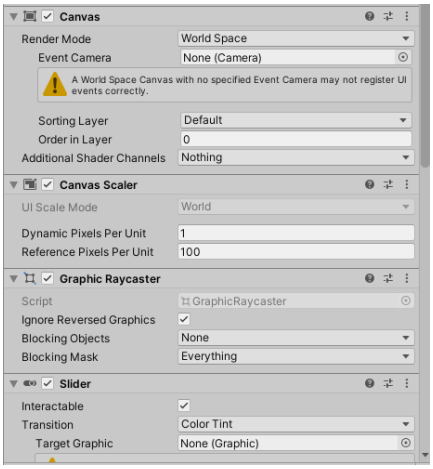


Рисунок 3.28 — Властивості панелі здоров’я

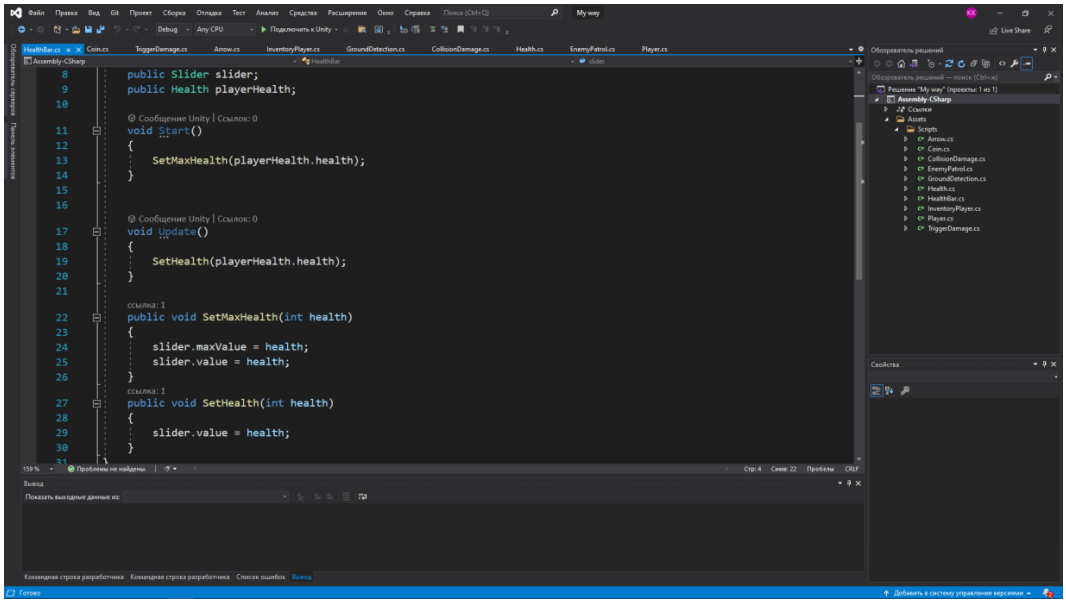


Рисунок 3.29 — Код для маніпуляції здоров’ям

Тепер нам необхідно створити рівень, після чого розставити на ньому різні об'єкти та ворогів. Насамперед потрібно прикинути зразковий план рівня, а також розташування ворогів та об'єктів. Після того, як загальний план рівня буде достатньо опрацьований, можна приступати до реалізації самого рівня. Рівень можна зібрати прямо у вікні сцени Unity.

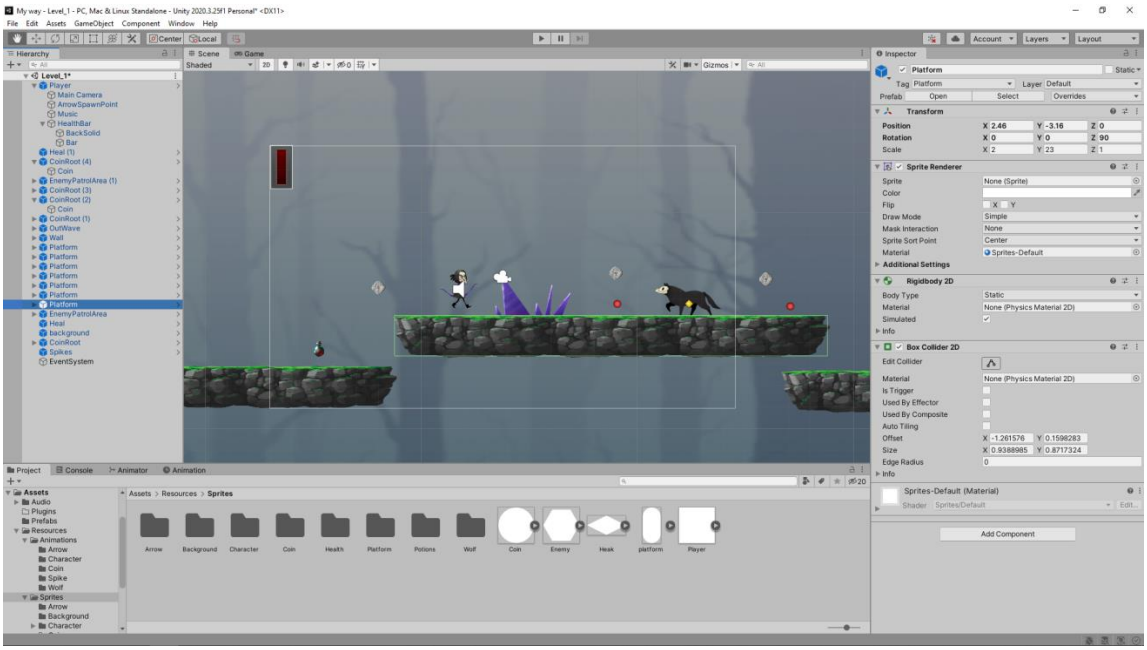


Рисунок 3.30 — Ресурси для створення рівня гри

3.3 Технічні характеристики

Насамперед, варто розуміти, що даний проект не є повністю готовою до релізу комп'ютерною грою, і за своєю суттю є прототипом, що стоїть приблизно на рівні ранньої альфа-версії. Викликано це було тим, що виробництво якісного товару займає колосальною кількість часу. Так, наприклад, розробка невеликої гри для мобільних телефонів з рівнями, що процедурно генеруються, може займати більше року, за умови, що над проектом працює не більше трьох осіб. Збільшення кількості людей може прискорити розробку, але не сильно.

На даний момент дуже складно визначити, які системні вимоги матиме проект у майбутньому, оскільки велика кількість елементів просто не реалізовані. На даний момент ми маємо лише елементи, що відповідають за основну ігрову механіку, а отже, у майбутньому технічні характеристики для комп'ютерної гри, що розробляється дуже сильно зміняться.

Зараз можна з упевненістю стверджувати лише те, що для запуску даного прототипу потрібен комп'ютер, що має системні характеристики не нижче мінімально допустимих для коректної роботи ігрового рушія Unity, для якого найбільш важливим аспектом є тільки відеокарта. Вона повинна підтримувати DirectX 9 з шейдерами не нижче за версію 3.0. Це означає, що в даний момент створений прототип з великою ймовірністю піде на більшості комп'ютерів.

Оскільки для розробки було обрано ігровий рушій Unity, а для написання та редагування програмного коду використовується Microsoft Visual Studio 2019, то знадобиться обладнання, що має технічні характеристики не нижче мінімально необхідних для коректно роботи даних засобів розробки. Тому розробка велася на комп’ютері з такими параметрами:

— Процесор Intel Core i3-2100 3,1 ГГц;
— Відеокарта NVidia GeForce GTX 1060 6Гб VRAM;
— ОЗУ16Гб;
— ОС Windows 10.

Для реалізації проекту було використано безліч різних елементів, починаючи від статичних картинок і закінчуючи різноманітними скриптами для реалізації механіки гри.

Під час розробки гри було створено 2 сцени: Головне меню та Ігровий рівень. У головному меню присутні фонове зображення, назва гри та 3 кнопки, що відповідають за запуск Ігрового рівня, інформацію про саму гру та вихід з гри.

Загальна кількість об'єктів, які були використані або створені під час реалізації проекту, представлені у таблиці 2.

Таблиця 2 — Калькуляція елементів гри

Елемент	Коментар
Головне меню	7 спрайтів
Фонове зображення	1 растрове зображення
Тайли	3 тайли
Головний герой	4 анімації, 24 спрайтів
Ворог	2 анімації, 18 спрайтів
Статичний небезпечний об’єкт	3 спрайти
Колекціоновані об'єкти	7 спрайтів
Скрипти	12 скриптів
Звук	3 аудіозаписи
Інші елементи	4 спрайти

В Ігровому рівні присутні всі основні елементи гри, і загалом містить понад 90 об'єктів, як повторюваних, так і ні. У тому числі головний персонаж, 6 ворогів, кілька об’єктів, які перешкоджають шлях, а також об'єкти, що збираються.

Варто відзначити, що на даному етапі реалізації проекту проводити якесь велике тестування сенсу немає, тому що в грі реалізована лише мала частина можливостей, що передбачалося реалізувати. Саме з цієї причини дане тестування (таблиця 3) є лише невеликою перевіркою на відповідність основним функціональним вимогам.

Таблиця 3 — Тест-кейс

№	Призначення	Дія	Очікуємий результат	Отриманний результат	Підсумок
1	Перевірка на коректне реагування персонажа на команди про пересування	Використову-ючи клавіші руху, пробігти частину рівня	Персонаж рухатиметься в належному напрямку, коректне програвання анімації	Персонаж пересувався у потрібному напрямку, коректне відображення анімації	Пройдено
2	Перевірка на коректний стрибок	Використовую чи клавішу стрибка, підстрибнути кілька разів поспіль	Гравець може зробити стрибок тільки із землі та небезпечного статичного об'єкта, коректне програвання анімації	Персонаж здатний зробити стрибок тільки з землі та небезпечного статичного об'єкта, вірне виконання анімації	Пройдено
3	Проверка поведения врагов	Підійти до ворога на досить близьку відстань, втекти від ворога	Поки гравець перебуває поза увагою ворога, патрулює місцевість. Якщо герой потрапляє у поле зору супротивника, ворог починає його переслідувати, якщо під час переслідування гравець виходить за межу патрулювання противника, він повертається до патрулювання	Патрулює місцевість, якщо перебуває у пасивному стані, коли помічає гравця, переслідує його, якщо гравець втік, повертається на стартову точку	Пройдено
4	Перевірка на коректну реалізацію події програшу гравця	Дозволити противнику атакувати гравця	Коли ворог атакує гравця, програється анімація смерті персонажа та анімація атаки у супротивника. Час зупиняється та з'являється меню вибору дії	Ворог атакує гравця, анімація програється правильно, але не програється анімація смерті головного героя, з'являється меню вибору дії	Частково пройдено
5	Перевірка на коректну роботу меню програшу	1) Програти та перезапустити гру 2) Програти та вийти в головне меню	Можна перезапустити гру або вийти в головне меню.	Залежно від вибору перезапускається гра або виходить в головне меню	Пройдено
6	Перевірка на коректну роботу головного меню	1) Запустити гру 2) Інформація про гру 3) Вийти з гри	Є можливість запустити гру, переглянути інформацію про неї або вийти з гри	Залежно від вибору запускається гра, відображається інформація про неї або закривається гра	Пройдено

Дане тестування показує, що на даному етапі реалізації проекту гра відповідає більшості функціональних вимог виділених на етапі постановки задачі. Але все ж таки варто приділити особливу увагу програмному коду і виправити помилку, через яку ситуація програшу гравця відпрацьовується неправильно.

ВИСНОВКИ

Під час аналізу доступних джерел проведено дослідження поняття платформеру, під час якого проведена класифікація платформерів за двома критеріями.

Проаналізовано популярні засоби розробки. У ході аналізу, проведено їх порівняння та обрано найбільш актуальні засоби розробки для розробників-початківців. Вибір пріоритетних засобів розробки проходив за двома критеріями: доступність та функціональність.

При аналізі існуючих розробок, проведено їх порівняння та виділено їх переваги та недоліки. У ході аналізу стало зрозуміло, що при розробці комп'ютерної гри з простою ігровою механікою варто звернути увагу на додаткові елементи гри, такі як графічне оформлення. Це потрібно для того, щоб утримати потенційного гравця і продовжити життєвий цикл розробки.

Грунтуючись на отриманій в ході дослідження інформації, вирішено розробити прототип двовимірного платформера для одного гравця на ігровому рушії Unity. Таке рішення було ухвалено з кількох причин:

— двовимірна графіка, на відміну від тривимірної легше у створенні;

— з ігрової механіки, гра жанру платформер простіше реалізується;

— ігровий рушій Unity поширюється безкоштовно і дозволяє розробляти програми мовою програмування C#.

Після вибору засобів розробки розпочато ознайомлення з Unity, а також розробка самого проекту. У ході розробки був розглянутий ігровий рушій Unity та були набуті необхідні навички, а саме:

— створення сцен;

— створення анімацій;

— створення та написання скриптів;

— налаштування об'єктів;

— створення UI;

— компіляція проекту.

Освоєння середовища розробки Unity несе не маловажний характер, оскільки у світі, індустрія розробки ігор дедалі більше поширюється у суспільстві. Ігри перестали бути лише предметом для розваг, і тепер використовуються і в інших областях, наприклад, у науці чи навчанні користувачів. Тому розвиток у цьому напрямі вважатимуться однією з найважливіших у суспільстві.

У ході реалізації проекту було виконано такі завдання:

— обрані жанр, вид та платформа для комп'ютерної гри;

— проаналізовано особливості існуючих 2D платформерів;

— розроблено концепцію основних елементів;

— обрано та вивчено засіб реалізації;

— підготовлено необхідні для гри матеріали;

— реалізовано прототип гри.

Таким чином всі поставлені задачі вирішено і мета дослідження досягнута.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Хокинг Джозеф Unity в действии. – Питер: Видавництво Пресс, 2018. — 352 с.

2. Рушій для розробки комп’ютерних ігор Unity [Електронний ресурс] Режим доступу: https://unity.com/

3. Харрісон Ферроне Вивчаємо C# через розробку ігор Unity. п’яте видання. — 2021. — 400 с.

4. Duck Tales Remastered [Електронний ресурс] Режим доступу: http://store.steampowered.com/app/237630/DuckTales_Remastered/

5. Dust An Elysian Tail (wiki) [Електронний ресурс] Режим доступу: https://elysiantail.fandom.com/ru/wiki/Dust:_An_Elysian_Tail_(%D0%B8%D0%B3%D1%80%D0%B0)

6. Game Maker Studio [Електронний ресурс] Режим доступу: https://www.yoyogames.com/gamemaker

7. Limbo [Електронний ресурс] Режим доступу: http://playdead.com/games/limbo/

8. Mark of the Ninja [Електронний ресурс] Режим доступу: https://www.kleientertainment.com/games/mark-ninja

9. Ori and the Blind Forest [Електронний ресурс] Режим доступу: http://www.oriblindforest.com/

10. Wings of Vi [Електронний ресурс] Режим доступу: http://www.grynsoft.com/wings-of-vi

11. Unreal Engine [Електронний ресурс] Режим доступу: https://www.unrealengine.com

12. UNITY3D Мини-урок [9] — Звуки та їх відтворення — audio.Play(), YouTube [Електронний ресурс] відеоролик. — Режим доступу: https://www.youtube.com/watch?v=EGI32DEBfsM

13. Торн А. Основи анімації в Unity / Алан Торн ред.: Д. Мовчан, 2016 — 176с.

14. Unity Manual, Unity Documentation [Електронний ресурс]: довідник. — Режим доступу: https://docs.unity3d.com/Manual/

15. Об'єктно-орієнтоване програмування (C#) [Електронний ресурс]: Режим доступу: https://docs.microsoft.com/ru_ru/dotnet/csharp/fundamentals/tutorials/oop

16. Ігровий рушій [Електронний ресурс]: Режим доступу: https://uk.wikipedia.org/wiki/Ігровий_рушій

17. Unity 5 tutorial for beginners: 2D Platformer — Creating platforms, YouTube [Електронний ресурс]: відеоролик. — Режим доступу: https://www.youtube.com/watch?v=hrKx5KdG4oA

18. Хокінг Дж. Unity в действии. Мультиплатформенная разработка на C# / Джозеф Хокінг — Київ: Нора-Друк, 2016 — 336с.

19. Unity 5 tutorial for beginners: 2D Platformer — Damage & Death animations, YouTube [Електронний ресурс]: відеоролик. — Режим доступу: https://www.youtube.com/watch?v=QXsYE0AKOoU

20. Unity Asset Store 2D [Електронний ресурс] Режим доступу: https://assetstore.unity.com/2d

21. Unity 5 tutorial for beginners: 2D Platformer — Player health bar [Електронний ресурс] Режим доступу: https://www.youtube.com/watch?v=aWR3Lb34nYQ&list=PLX_uZVK_0K_6VXcSajfFbXDXndb6AdBLO&index=36

ДОДАТОК А

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ ВНТУ
д.т.н., проф.
_____ О. Д. Азаров
«29» квітня 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської дипломної роботи
«Програмне забезпечення в середовищі Unity для RPG платформера комп’ютерної 2D гри»
08–23.БДР.036.00.000 ПЗ

Науковий керівник к.т.н. доц. каф. ОТ
_____ Черняк О.І.
Студент групи 1КІ–20мс
_____ Магдич В.В.

Вінниця 2022

1 Підстави для виконання бакалаврської дипломної роботи (БДР)

1.1 Актуальність розробки, яка полягає у необхідності розвитку сучасного суспільства та масової культури в розробці ігор та популяризації їх, тому розвиток технологій у цьому напрямі можна вважати одним з найперспективніших;

1.2 Наказ про затвердження теми бакалаврської дипломної роботи.

2 Мета і призначення БДР

2.1 Метою БДР є розробка програмного забезпечення в середовищі Unity для RPG платформера комп’ютерної 2D гри, яка дозволить управляти своїм персонажем.

2.2 Призначення розробки полягає у виконанні бакалаврської дипломної роботи із подальшим впровадженням та розвитком.

3 Вихідні дані для виконання БДР

2.1 Було проаналізовано популярні засоби розробки. У ході аналізу, було проведено їх порівняння та обрано найбільш актуальні засоби розробки.

2.2 Грунтуючись на отриманій в ході дослідження інформації, було вирішено розробити прототип двовимірного платформера для одного гравця на ігровому движку Unity.

2.3 Для написання коду середовищем розробки було обрано Microsoft Visual Studio.

4 Вимога до виконання БДР

Головна вимога — створення прототипу однокористувацького двовимірного платформера для персональних комп'ютерів.

5 Етапи БДР та очікувані результати

Робота виконується за вісім етапів, таблиця А.1.

Таблиця А.1 — Етапи виконання роботи

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Постановка задачі роботи	09.03.22	09.03.22	Вступ
2	Пошук матеріалів по технологіям розробки комп'ютерних ігор	10.03.22	25.03.22	Розділ 1
3	Обгрунтування та вибір засобів реалізації комп'ютерної гри	26.03.22	03.04.22	Розділ 2
4	Структурне проектування персонажу, супротивника та рівня комп'ютерної гри	04.04.22	09.04.22	Розділ 3
5	Розробка головних механік для комп'ютерної гри	10.04.22	15.05.22	Розділ 3
6	Підготовка матеріалів та розробка рівня комп'ютерної гри	16.05.22	27.05.22	Розділ 3
7	Оформлення пояснювальної записки та ілюстративного матеріалу	28.05.22	06.06.22	Пояснювальна записка
8	Перевірка якості виконання бакалаврської роботи та усунення недоліків	07.06.22	07.06.22	Презентація

6 Матеріали, що подаються до захисту БДР

До захисту подаються: пояснювальна записка БДР, графічні і ілюстративні матеріали, протокол попереднього захисту БДР на кафедрі, відгук наукового керівника, відгук опонента, анотації до БДР українською та іноземною мовами, нормоконтроль про відповідність оформлення ДР діючим вимогам.

7 Порядок контролю виконання та захисту БДР

Виконання етапів графічної та розрахункової документації ДР контролюється науковим керівником згідно зі встановленими термінами. Захист ДР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення БДР

При оформлюванні БДР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— Документами на які посилаються у вище вказаних.

ДОДАТОК Б

Файл головного героя

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

public class Player : MonoBehaviour
{
    [SerializeField]private float speed = 2.5f;
    public float Speed
    {
        get { return speed; }
        set
        {
            if (speed > 0.5)
                speed = value;
        }
    }
    public float force;
    public Rigidbody2D rigidbody;
    public float minimalHeight;
    public bool isCheatMode;
    public GroundDetection groundDetection;
    public Vector3 direction;
    public Animator animator;
    public int coinsCount;
    [SerializeField] private SpriteRenderer spriteRenderer;
    [SerializeField] private Arrow arrow;
```

```
[SerializeField] private Transform arrowSpawnPoint;
[SerializeField] private int shootForce = 3;
[SerializeField] private float cooldown;
private List<Arrow> arrowPool;
[SerializeField] private int arrowsCount = 5;
private Arrow currentArrow;
private bool isCooldown;

private void Start()
{
    arrowPool = new List<Arrow>();
    for(int i = 0; i < arrowsCount; i++)
    {
        var arrowTemp = Instantiate(arrow, arrowSpawnPoint);
        arrowPool.Add(arrowTemp);
        arrowTemp.gameObject.SetActive(false);
    }
}
void Update()
{
    animator.SetBool("isGrounded", groundDetection.isGrounded);
    direction = Vector3.zero;
    if (Input.GetKey(KeyCode.A))
        direction = Vector3.left;
    if (Input.GetKey(KeyCode.D))
        direction = Vector3.right;
    direction *= speed;
    direction.y = rigidbody.velocity.y;
    rigidbody.velocity = direction;
```

```
        if((Input.GetKeyDown(KeyCode.W) || Input.GetKeyDown(KeyCode.Space)) &&
groundDetection.isGrounded)
        {
            rigidbody.AddForce(Vector2.up * force, ForceMode2D.Impulse);
            animator.SetTrigger("StartJump");
        }
        animator.SetFloat("Speed", Mathf.Abs(direction.x));

        if (direction.x > 0)
            spriteRenderer.flipX = false;
        if (direction.x < 0)
            spriteRenderer.flipX = true;

        CheckFall();
        CheckShoot();
    }
    void CheckFall()
    {
        if (transform.position.y < minimalHeight && isCheatMode)
        {
            rigidbody.velocity = new Vector2(0, 0);
            transform.position = new Vector2(0, 0);
        }
        else if (transform.position.y < minimalHeight)
            Destroy(gameObject);
    }
    void CheckShoot()
    {
        if (Input.GetMouseButtonDown(0) && !isCooldown)
```

```
    {
        animator.SetTrigger("StartShoot");
    }
}
public void InitArrow()
{
    currentArrow = GetArrowFromPool();
    currentArrow.SetImpulse(Vector2.right, 0, this);
}
public void Shoot()
{
    currentArrow.SetImpulse(Vector2.right, spriteRenderer.flipX ?
        -force * shootForce : force * shootForce, this);

    StartCoroutine(Cooldown());
}

private IEnumerator Cooldown()
{
    isCooldown = true;
    yield return new WaitForSeconds(cooldown);
    isCooldown = false;
}
private Arrow GetArrowFromPool()
{
    if(arrowPool.Count > 0)
    {
        var arrowTemp = arrowPool[0];
        arrowPool.Remove(arrowTemp);
```

```
            arrowTemp.gameObject.SetActive(true);
            arrowTemp.transform.parent = null;
            arrowTemp.transform.position = arrowSpawnPoint.transform.position;
            return arrowTemp;
        }
        return Instantiate(arrow, arrowSpawnPoint.position, Quaternion.identity);
    }

    public void ReturnArrowToPool(Arrow arrowTemp)
    {
        if (!arrowPool.Contains(arrowTemp))
            arrowPool.Add(arrowTemp);

        arrowTemp.transform.parent = arrowSpawnPoint;
        arrowTemp.transform.position = arrowSpawnPoint.transform.position;
        arrowTemp.gameObject.SetActive(false);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Coin"))
        {
            coinsCount++;
            Debug.Log("Количество монет = " + coinsCount);
        }
    }


     }
```

ДОДАТОК В

Файл поведінки вовка

```
using UnityEngine;

public class EnemyPatrol : MonoBehaviour
{
    public GameObject leftBoarder;
    public GameObject rightBoarder;
    public Rigidbody2D rigidbody;
    public GroundDetection groundDetection;
    public bool isRightDirection;
    public float speed;
    [SerializeField] private SpriteRenderer spriteRenderer;
    [SerializeField] private CollisionDamage collisionDamage;

    private void Update()
    {
        if (groundDetection.isGrounded)
        {
            if (transform.position.x > rightBoarder.transform.position.x
                || collisionDamage.Direction < 0)
                isRightDirection = false;
            else if (transform.position.x < leftBoarder.transform.position.x
                || collisionDamage.Direction > 0)
                isRightDirection = true;
            rigidbody.velocity = isRightDirection ? Vector2.right : Vector2.left;
            rigidbody.velocity *= speed;
        }
```

```
        if (rigidbody.velocity.x > 0)
            spriteRenderer.flipX = true;
        if (rigidbody.velocity.x < 0)
            spriteRenderer.flipX = false;
    }
}
```

ДОДАТОК Г

Графічні матеріали

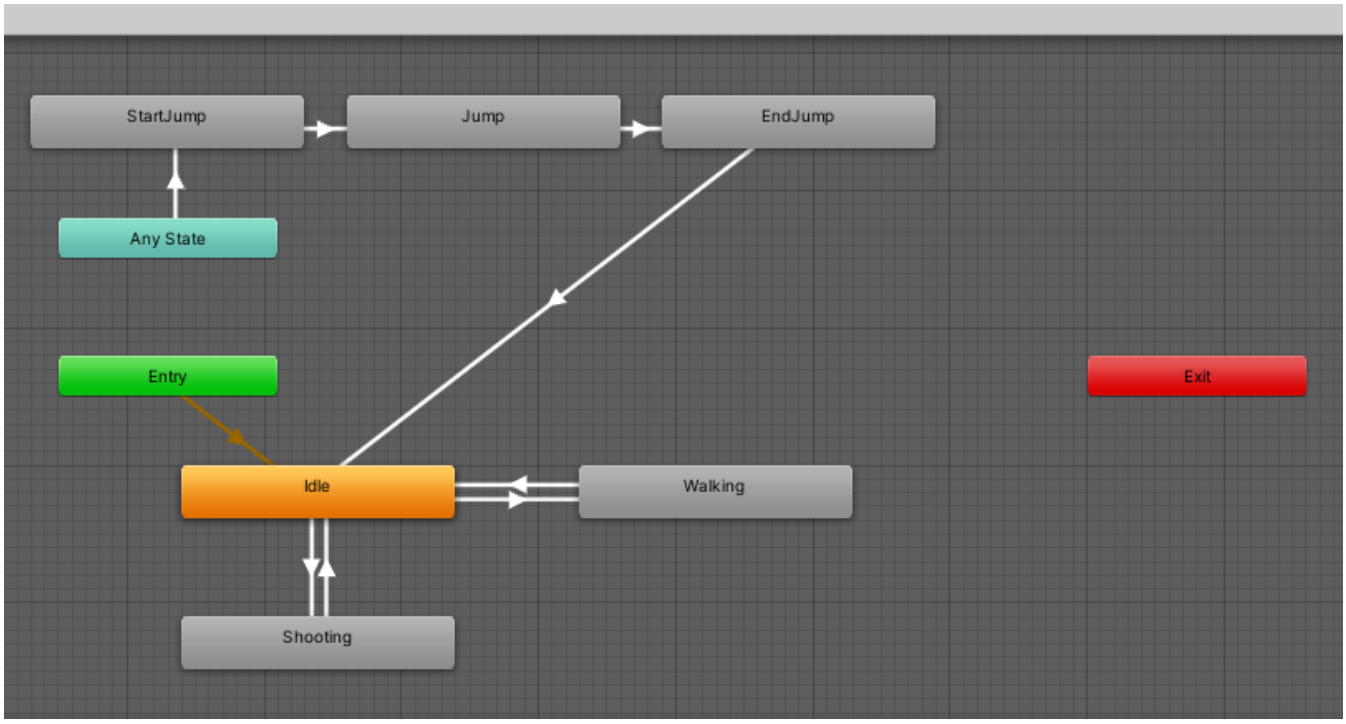


Рисунок Г.1 – Концептуальна модель головного героя

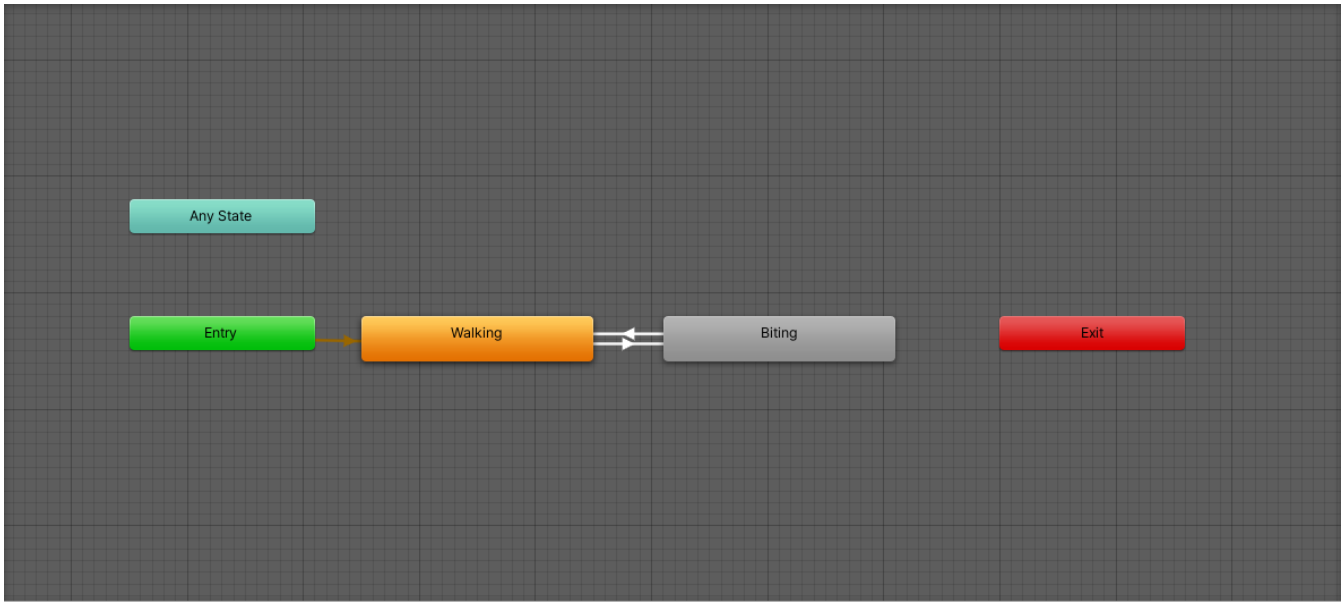


Рисунок Г.2 – Концептуальна модель вовка

ДОДАТОК Д

Код маніпуляції здоров’ям

```
using UnityEngine;
public class Health : MonoBehaviour{
    public int health;
    public void TakeHit(int damage)    {
        health -= damage;
        Debug.Log(health);
        if (health <= 0)
            gameObject.SetActive(false);
    }
    public void SetHealth(int bonusHealth) {
        health += bonusHealth;
        if (health > 100)
            health = 100;
    }
    private void OnTriggerEnter2D(Collider2D collision) {
        if (collision.gameObject.CompareTag("Heal"))
        {
            SetHealth(15);
            Debug.Log(health);
            Destroy(collision.gameObject);

        }
    }
}
```

ДОДАТОК Е

Файли для нанесення урона та їх анімації

TriggerDamage.cs

```
using UnityEngine;

public class TriggerDamage : MonoBehaviour
{
    [SerializeField] private int damage;
    public int Damage
    {
        get { return damage; }
        set { damage = value; }
    }
    [SerializeField] private bool isDestroyingAfterCollision;
    private GameObject parent;
    public GameObject Parent
    {
        get { return parent; }
        set { parent = value; }
    }
    private IObjectDestroyer destroyer;
    public void Init(IObjectDestroyer destroyer)
    {
        this.destroyer = destroyer;
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject == parent)
```

```
            return;
        var health = collision.gameObject.GetComponent<Health>();
        if (health != null)
        {
            health.TakeHit(damage);
        }
        if (isDestroyingAfterCollision)
        {
            if (destroyer == null)
                Destroy(gameObject);
            else destroyer.Destroyer(gameObject);
        }
    }
}
public interface IObjectDestroyer
{
    void Destroyer(GameObject gameObject);

}
```

CollisionDamage.cs

```
using UnityEngine;

public class CollisionDamage : MonoBehaviour
{
    public int damage = 10;
    private Health health;
    [SerializeField] private Animator animator;
    [SerializeField] private SpriteRenderer spriteRenderer;
```

```
    private float direction;
    public float Direction => direction;
    private void OnCollisionStay2D(Collision2D collision)
    {
        health = collision.gameObject.GetComponent<Health>();
        if (health != null)
        {
            direction = (collision.transform.position - transform.position).x;
            animator.SetFloat("Direction", Mathf.Abs(direction));
        }
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if(collision.gameObject.tag == "Player")
        {
            health = collision.gameObject.GetComponent<Health>();
            SetDamage();
        }
    }
    public void SetDamage()
    {
        if(health != null)
            health.TakeHit(damage);
        health = null;
        direction = 0;
        animator.SetFloat("Direction", 0f);
    }

}
```

ДОДАТОК Ж

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ
ЗАПОЗИЧЕНЬ

Назва роботи: Програмне забезпечення в середовищі Unity для RPG платформера комп'ютерної 2D гри

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності
Unicheck

Оригінальність 94,4% Схожість 5,6%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Магдич В. В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Черняк О. І.
(підпис) (прізвище, ініціали)