ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**Розробка гри «Cave» в жанрі 2D платформер з використанням двигуна Unity**»

Виконала: студентка 4 курсу, групи ПД-44
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Козлова Ю.С.
(прізвище та ініціали)

Керівник Дібрівний О.А.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Київ –2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти -«Бакалавр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Негоденко О.В.
"____" _____2022 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТА

КОЗЛОВОЇ ЮЛІЇ СЕРГІЇВНИ
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка гри «Cave» в жанрі 2D платформер з використанням двигуна Unity»

Керівник роботи:____Дібрівний О.А. Доктор філософії (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «16» лютого 2022 року №22.
2. Строк подання студентом роботи___«6» червня 2022 року
3. Вхідні дані до роботи
Інтегроване середовище розробки Visual Studio. Методи та засоби проектування та розробки програмного забезпечення, графічний двигун Unity
4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).
4.1 Аналіз предметної області
4.2 Вибір інструментальних засобів реалізації
4.3 Реалізація та тестування гри
4.4. Висновки.

5. Дата видачі завдання «11» квітня 2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	16.02.22 - 12.04.22	Виконано
2	Проблематика дослідження	12.04.22 - 15.04.22	Виконано
3	Аналіз ігор	15.04.22 - 18.04.22	Виконано
4	Дослідження програмних засобів	21.04.22 - 24.04.22	Виконано
5	Моделювання об’єкту проектування	24.04.22 - 27.04.22	Виконано
6	Розробка функціоналу гри	30.04.22 – 05.05.22	Виконано
7	Розробка презентації	10.05.22 – 13.05.22	Виконано
8	Попередній захист роботи	31.06.22	
9	Здача роботи	06.06.22	

Студент _____ Козлова Ю.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Дібрівний О.А.

РЕФЕРАТ

Текстова частина магістерської роботи 60 с., 41 рис., 14 джерел.

UNITY, CAVE, PLAYER, PC, 2D

Об'єкт дослідження – процес проектування та розробки гри у жанрі 2D платформер

Предмет дослідження – ігровий двигун Unity

Мета роботи – створення гри Cave в жанрі 2D платформер на ігровому двигуні Unity

Методи дослідження – методи наукового пізнання такя як метод порівняння, аналогії, проведення експерименту, аналіз отриманих результатів.

Для досягнення поставленої мети в даній роботі:

1. Розроблено алгоритм для реалізації гри Cave в жанрі 2D платформер з використанням ігрового двигуна Unity.
2. Встановлено, що використання ігрового двигуна Unity – це вдале рішення для досягнення поставленої мети.
3. Показано, що ігровий двигун Unity зручний у використанні та надає можливість ефективної розробки застосунку.
4. На основі результатів виконаних досліджень розроблено гру Cave в жанрі 2D платформер з використанням ігрового двигуна Unity.

   Результат роботи полягає в тому, щоб отримати якісну та цікаву гру в жанрі 2D платформер на ігровому двигуні Unity.

Галузь використання – завдяки вільному доступу гру може використовувати кожен, хто хоче пограти у гру Cave.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Engine – основне ядро гри, на основі якого будуються складові гри та спрощують роботу для розробника.

Asset – частина ігрового контенту, яка оброблюється грою для відображення її на сцені.

Player – основний персонаж гри яким керує гравець.

Enemy – об’єкт на сцені який необхідно знищити.

GUI (Graphical User Interface) – інтерфейс за допомогою якого користувач має можливість взаємодіяти із програмою через візуальні елементи управління.

IDE (Integrated Drive Electronics) – інтегрована середовище розробки, яка використовується програмістами для розробки програмного забезпечення.

ВСТУП

Актуальність дослідження. Будь-яка гра – це засіб випробування альтернативної, дещо спрощеної, моделі чогось. Так, спортивні ігри, від футболу до шахів – зазвичай, спрощення війни. Саме тому агресивні політичні режими так люблять Олімпіади і приділяють професійному спорту, можливо, навіть забагато державної уваги. Бо спортивна перемога для них ввижається різновидом військової перемоги.

Комп'ютерні ігри – інші. Цей наджанр створено не для державного самоствердження, а для персонального випробування. Для того, щоб ми могли побувати у будь-якому спектрі змодельованих ситуацій, не встаючи з крісла. Наприклад, побути президентом чи депутаткою.

За роки з простої переважно спинномозкової розваги комп'ютерні ігри еволюціонували не тільки кількісно – як індустрія, але і якісно – як вид мистецтва. Сучасна популярна гра – це завжди шедевр на перетині жанрів. Наратив якісної рольової гри – від польської серії «Відьмаків» до канадського Dragon Age не має поступатися роману. Над графічним виконанням працюють професійні художники та аніматори, музику записують симфонічні оркестри та культові групи (німецька The Dwarves – досить-таки посередня гра, але музика Blind Guardian!), і лише один вдалий проект може зробити вчорашню невелику студію у всевідому (World of Tanks, що перетворив невеличку білоруську студію на світову корпорацію, чи Minecraft, що зробив кількох скромних шведів власниками компанії на кілька мільярдів).

Мета та завдання. Метою дипломної роботи є створення гри «Cave» в жанрі 2D платформер з використанням ігрового двигуна Unity для ПК

Для виконання дипломної роботи потрібно вирішення слідуючих завдань:

- Розробити теоретичні аспекти створення якісного 2D платформера
- Сформувати основні завдання та рішення проблематики дослідження
- Розробка основного функціоналу гри

– Тестування створенної гри

Об’єктом дослідження є процес розробки гри в жанрі 2D платформер.

Предметом роботи є ігровий двигун Unity.

Завданням роботи є розробка гри в жанрі 2D платформер.

Практична значущість результатів. Практичне значення отриманого результату полягає в написанні гри «Cave» в жанрі 2D платформер з використанням ігрового двигуна Unity для ПК, яка може бути використана за потребою.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття про 2D ігри

Найчастіше, 2D ігри - ігри з видом зверху, тобто камера розташована прямо перпендикулярно або під кутом над персонажем, або ігри з видом збоку або від третьої особи, коли ігровий процес постає у вигляді панорами. Спочатку 2D графіка використовувала бітове зображення, пізніше еволюціонувала до 8-бітної, а після і до 15/16 біт, так званого HighColor, а вже потім 24 біти, відомої як TrueColor, і, нарешті, 48-бітне зображення – DeepColor.

Історія 2D платформерів починається ще з 1980-х років. З появленням гри Super Mario Bros 2D платформери перейшли на інший рівень. В іграх почали з’являтися рівні з так званою прокруткою, коли камера слідувала за гравцем вліво або вправо. Гравець переміщається в сторони, вгору, вниз, стрибає з платформи на платформу, знищує ворогів, отримує бонуси та зброю.

Першими найпопулярнішими 2D платформерами були Super Mario Bros, Donkey Kong, Space Panic та Jungle King.

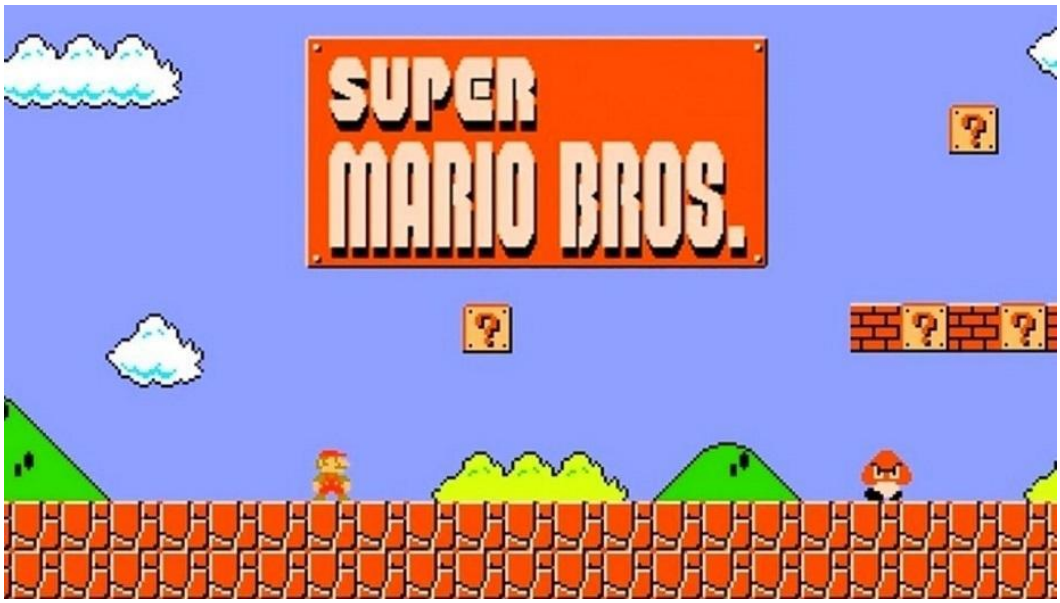

Рисунок 1.1 Гра «Super Mario Bros»

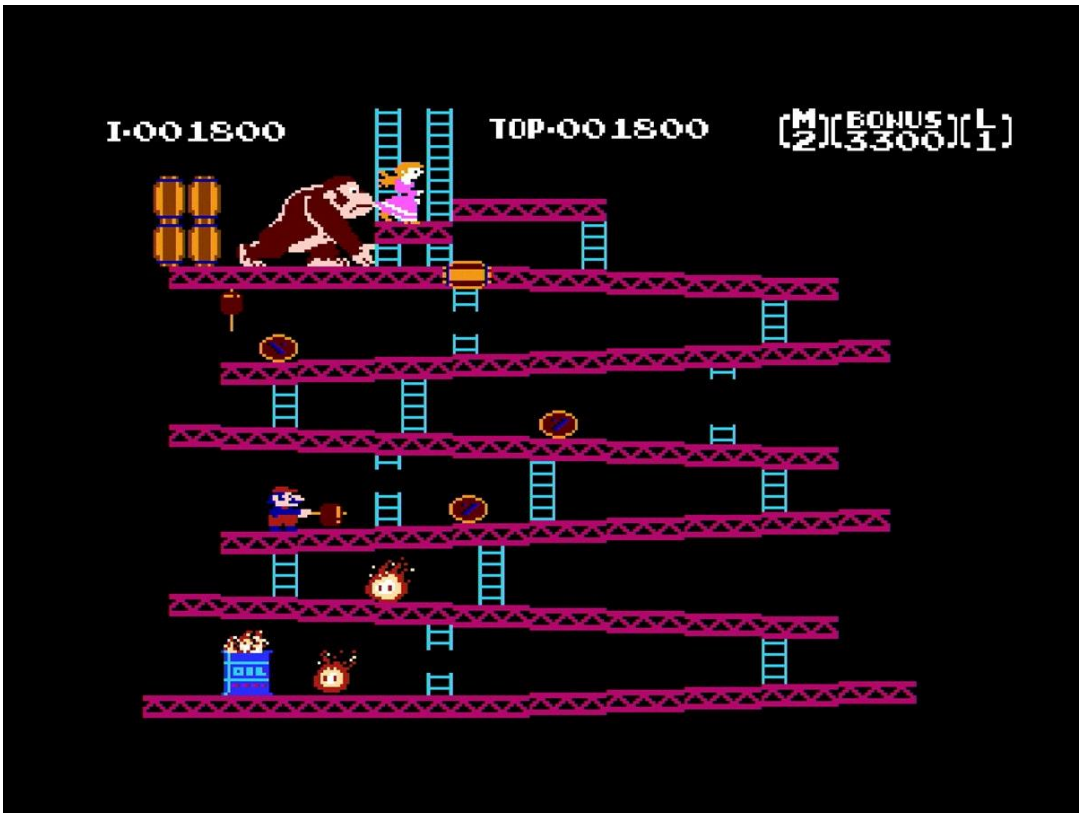

Рисунок 1.2 Гра «Donkey Kong»

Ігрова індустрія з 2D ігор стартувала і досі цей вид залишається актуальним як серед деяких невеликих компаній, і серед самоучок. Серед 2D ігрових проектів поширене використання піксельної графіки через простоту використання. З такою графікою малим групам розробників або розробникам-одинакам легше створити анімації, промалювати рівні та інше, приділивши більше уваги таким тонкощам, як опрацювання сюжету та персонажів.

Однак не варто забувати, що на світі дуже багато шедеврів ігрової індустрії, що привнесли нові фарби в індустрію, що створили нові сетінги і всесвіти, що не вмирають досі. Деякі з них, звичайно, перекочували в руки іншої компанії, яка частково занапастила, а частково дала нове життя - дивлячись як подивитися - і все через банкрутство початкових авторів (серія ігор Fallout). Загалом можна сказати, що було б бажання, вміння та фантазія – одна маленька компанія могла б створити щось велике, яке могло б залучити величезну кількість гравців.

2D-ігри також називаються платформерними іграми. Слово "платформа" також описує те, що знаходиться на платформі. Тут гравець може бігати, стрибати,

стріляти, збирати сили на платформі. Це жанр відеоігор; 2D-ігри стали дуже старими. Але деяким розробникам все ще подобається грати у 2D-ігри, щоб отримати інноваційну ідею, тому що ми вчимося всьому з минулого. Ось чому вони хочуть грати в 2D-ігри, щоб зробити свою гру більш цікавою, і у них з'являється ідея додати до своєї гри якусь нову функцію. Здебільшого персонажі 2D-ігор мультяшні та нереалістичні. Ми не можемо надати реалістичності нашому 2D-персонажу. Але в 3D це можливо через глибину. Використовуючи глибину, ми можемо створити персонажа, який виглядатиме як справжній. В основному платформери засновані на декількох рівнях, якщо гравець може вбити всіх ворогів або перетнути певну частину, яку він повинен перетнути (наприклад, Маріо), то тільки гравець може перейти на наступний рівень. На наступному рівні може бути більше ворогів, яких гравець має вбити.

Недоліки таких ігор полягають лише у візуальній складової, т.к. в 21-му столітті багато хто просто не переносить виду піксельної графіки. Але групі недосвідчених розробників, особливо якщо це їх дебютний проект, це можна пробачити, тому що їх можливості обмежені ресурсами та навичками. Також варто сказати, що багато сучасних сеттингів стартували саме з 2D: Grand Theft Auto, The Elder Scrolls, Warcraft, Warhammer 4000, Fallout та інші. Наприкінці 90-х років отримав широку популярність стиль псевдо-3D, який все ще залишався 2D, створюючи ілюзію 3D. Найвідомішими представниками такої стилістики є Doom, Doom 2, Wolfenstein 3D. Остання, незважаючи на назву, яка є рекламним ходом, гра двомірна.

І також головним недоліком такої графіки є неможливість деталізувати деякі деталі - плоскі моделі одного персонажа іноді доводиться малювати чотири або більше разів - оскільки сам движок гри не дозволяє зробити одну цільну модель. Також не варто забувати про анімації персонажам - у багатьох випадках вони рвані і виглядають як картинки, що змінюються, а не рух.

1.2. Класифікація 2D ігор

Графіка у грі також грає свою роль. Завдяки їй можна зробити гарну атмосферу, задати спеціальний стиль та просто приємний вигляд. Варіантів графіки доволі багато, від піксельної до реалістичної. Варіанти графіки:

- Реалістична
- Low Poly
- Cartoon
- Піксельна

Реалістична графіка приваблює доволі багато нових гравців, бо складно заперечувати той факт, що вона приваблює. Але не всім компаніям вистачає коштів на таку графіку та вони використовують іншу.

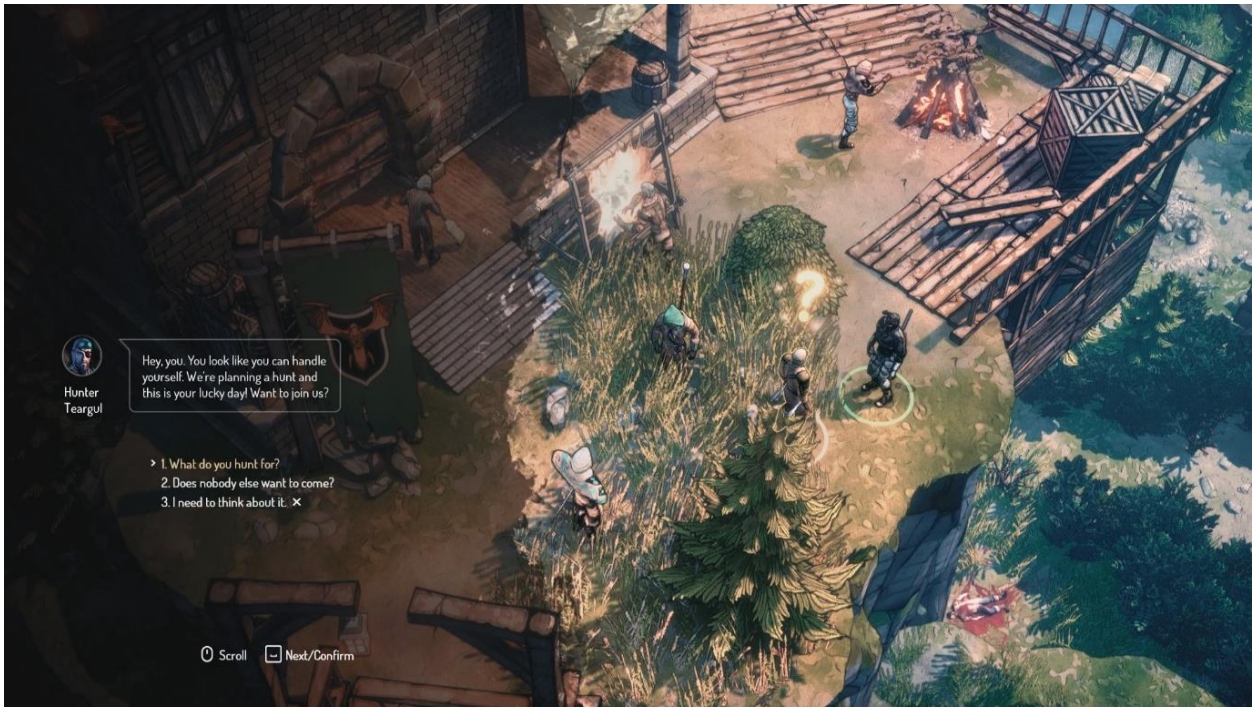

Рисунок 1.3 Реалістична графіка у грі «Seven: The Days Long Gone»

Low Poly - це тривимірна модель із малою кількістю полігонів. Тобто це моделі, які складаються з мінімальної кількості полігонів. Доволі цікавий варіант графіки, який має свою групу прихильників

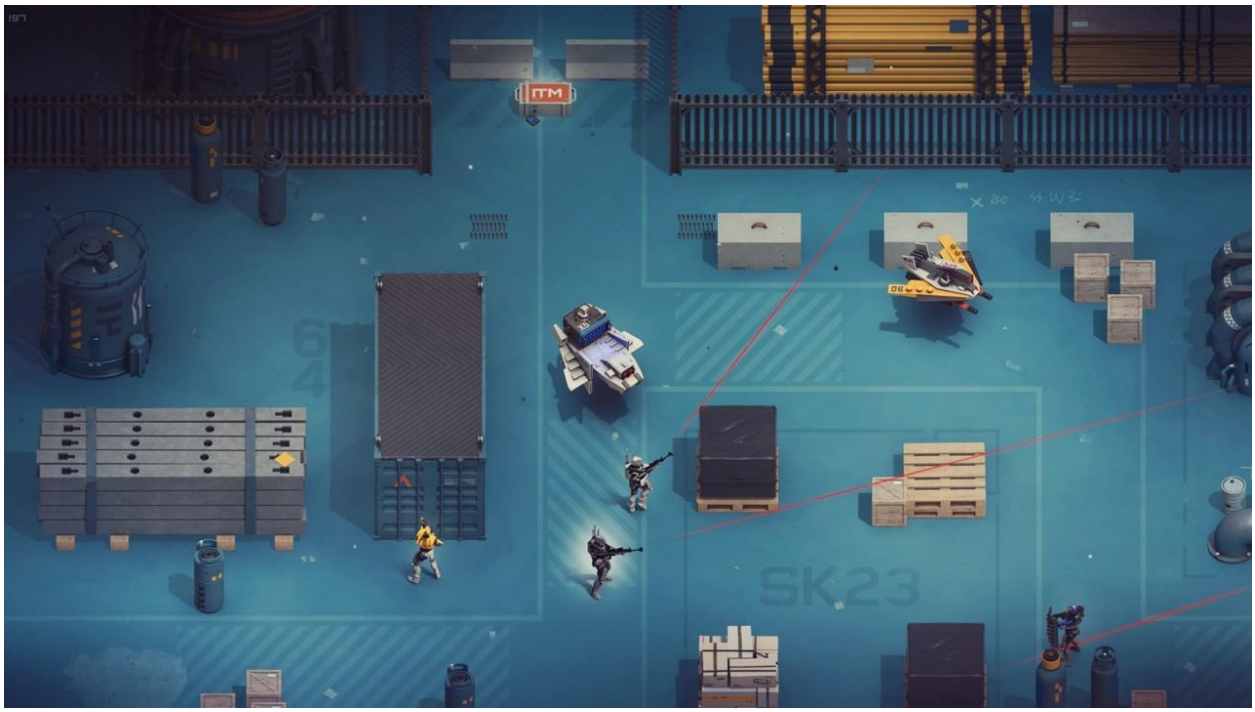

Рисунок 1.4 Low Poly графіка у грі «Synthetik: Legion Rising»

Cartoon (намальована) графіка містить в собі барвисті кольори та відносну простоту. Ігри виходять привабливими та доволі цікавими на зовнішній вигляд.

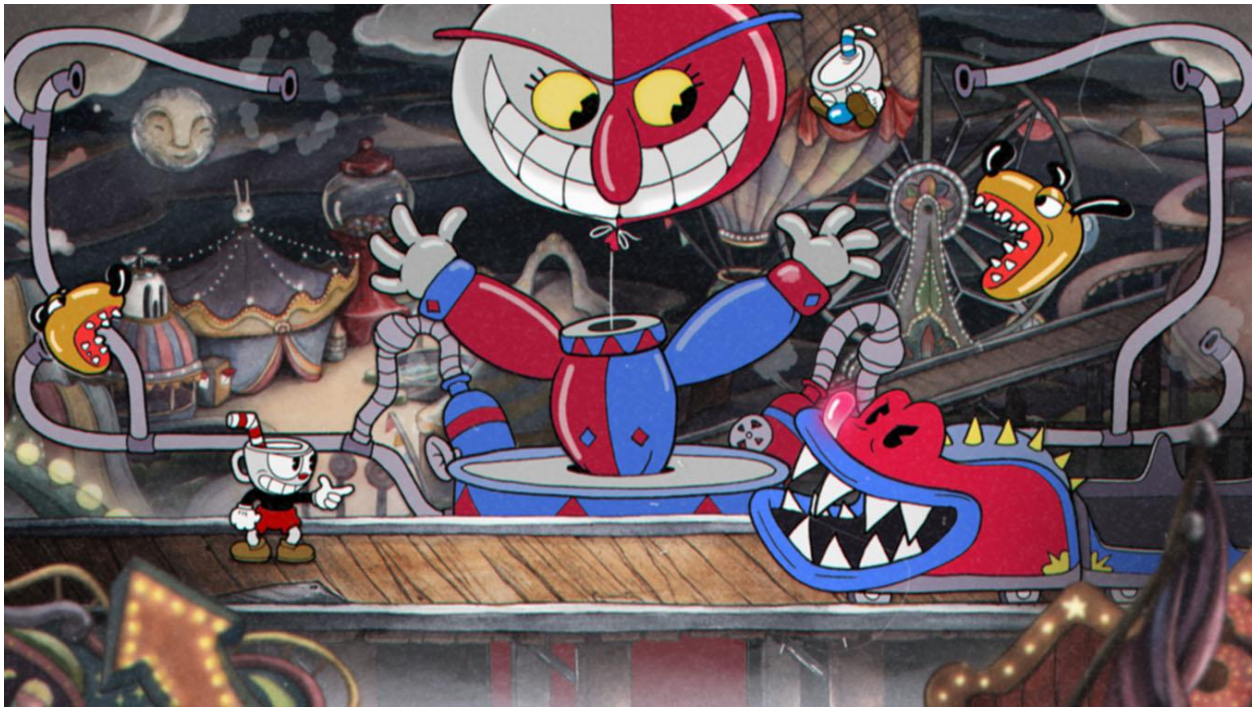

Рисунок 1.5 Cartoon графіка у грі «Cup Head»

Піксельна графіка має велику аудиторію фанатів. Це доволі стара за походженням графіка, бо старі ігри саме таки піксельні. Вдалий приклад, це гра «Hotline Miami», яка має свій цікавий стиль, за яким потім надихалися розробники інших подібних проєктів.

Рисунок 1.6 Піксельна графіка у грі «Hotline Miami»

Рисунок 1.7 Гра «Risk of Rain»

1.3. Проблематика та постановка завдань

Ігри це дуже добрий спосіб провести цікаво час. Вони захоплюють своїм зовнішнім виглядом, сюжетом, функціоналом та різноманітністю. І чесно кажучи від ігор іноді буває складно відірватися. Як раз однією з проблем ігор є те, що діти проводять за ними досить багато часу. Іноді навіть дорослі поринають в ігри на довгий час. Тому треба себе контролювати і не забувати про інші важливі речі в житті. А батькам слід дивитися за тим, як довго їх дитина сидить за грою.

Нажаль деякі розробники ігор випускають лише для того аби заробити, а не для того щоб залучити до себе аудиторію. Тому іноді попадаються сирі та не цікаві ігри, на які людина витрачає свій час без усілякого задоволення.

Ігри можуть не тільки приносити задоволення, а й розвиток. Деякі студії спеціально розробляють ігри для того, щоб користувач дізнався щось нове граючи. Це доволі цікавий досвід, який робить навчання більш привабливим. Більшість гравців є діти, бо такого роду ігри є гарним способом зацікавити дитину щось вивчити

Отже основні наші задачі, це зробити таку гру, щоб вона була з цікавою графікою, приваблювала своєю різноманітністю та геймплеєм сприяла розвитку розуму. Для цього треба:

- розробити основні модулі гри в жанрі 2D платформера з використанням ігрового двигуна Unity та опис структурних компонентів.
- провести тестування гри на відповідність наших задач

2. РОЗРОБКА ГРИ CAVE

2.1. Аналіз задачі

Ігри еволюціонують з неймовірною швидкістю. Кожен рік вигадають нові способи залучити більше аудиторії та уваги. Деякі великі студії пишуть свій ігровий двигун і роблять на ньому дуже гарні із зовнішнього вигляду ігри. Створюють нові механіки та пишуть цікаві сюжети для гри. І дійсно, з кожним роком гравців по всьому світу стає ще більше і більше, не зважаючи на вік людини. Ігри мають величезну аудиторію і вона продовжує збільшуватися. Зараз буде складно найти людину, яка хоча б раз не грала в гру.

Сьогодні ігри перетворилися на новий вид мистецтва. Зараз навіть фільми знімають за деякими сюжетами з ігор. Планка якості гри підвищується майже кожен рік, відповідно й рівень професійності розробника треба розвивати. І це дійсно відбувається, бо ігри стають ще кращими та кращими.

Гравцям потрібні враження та емоції. І щоб досягти цього потрібно дійсно зробити важку роботу. Деякі з розробників навіть вигадують нові жанри і це також змушує людей звернути на це увагу.

Щоб гравець затримався у грі якнайдовше, потрібен добре пророблений функціонал головного персонажу. В кожній грі він реалізований по-різному. Наприклад, підвищення рівня для того, щоб вивчити нове вміння, добування певних ресурсів для створення нової броні, накопичення золотих монет, щоб купити руну та багато іншого. І саме це є одним із найголовніших задач для цікавої гри.

2.1.1 Опис гри та правила

Ігровий простір являє собою двомірний рівень (кожен рівень відрізняється від попереднього) по якому рухається головний персонаж.

Щоб геймплей був цікавішим, на рівнях будуть знаходитися перешкоди та вороги, яких необхідно подолати.

На рівні ще будуть знаходитися золоті монети, які обов’язково треба зібрати, щоб пройти на наступний рівень.

2.1.2 Вимоги та характеристики

- Графіка не дуже навантажує зайвими ефектами та має цікавий стиль
- Механіка не складна та інтуїтивно зрозуміла для гравця
- Ігровий процес не викликає великої складності для проходження гри
- Гра дає варіативність дій
- Плавність дій на екрані
- Запуск гри на будь-якій версії Windows
- У користувача не повинно бути проблем із запуском гри

Також ігровий додаток повинен мати гнучку архітектуру. Програміст повинен мати можливість легко розширювати можливості ігрового додатку, допрацьовувати логіку гри (змінювати властивості існуючих в грі об'єктів, додавати нові об'єкти).

2.2. Вибір програмного забезпечення

2.2.1. Огляд платформ для розробки гри

Спочатку треба обрати для якої платформи буде розроблятися гра та для якої саме аудиторії. На кожній платформі, а саме: телефони, комп’ютери та ігрові консолі – є свій відсоток гравців.

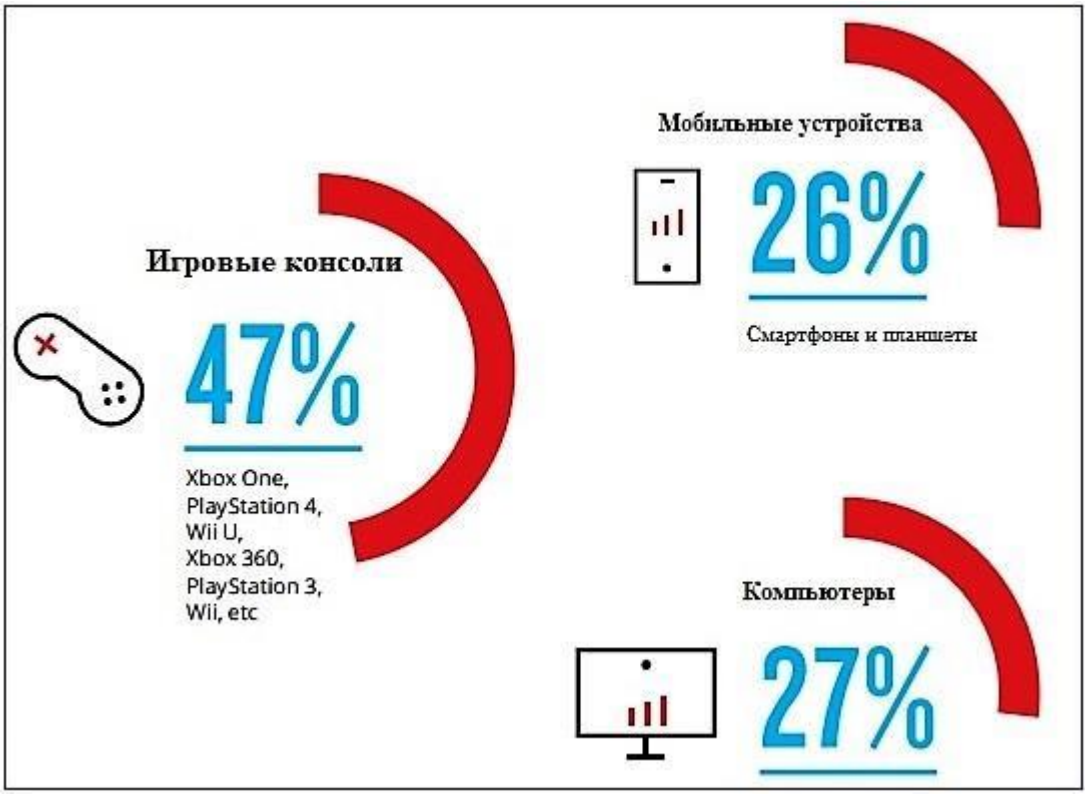

Рисунок 2.1 Відсоток гравців на кожній з платформ

Можна побачити, що кількість гравців на консолі – це майже половина від всіх гравців. Це доволі добре для студій які роблять ігри для ігрових консолей, бо вони сможуть отримати великий прибуток. І взагалі, ігри на консолях відрізняються по якості від інших двох варіантів, комп'ютерів та мобільних пристроїв. Це пояснюється тим, що ігрові консолі доволі дорого коштують.

Відсоток гравців на мобільних пристроях виявився найменшим. Ігри не дуже високої якості. Це зумовлено тим, що мобільні пристрої не такі потужні як консолі чи комп'ютери.

Отже зваживши усі за та проти я вирішила обрати розробляти гру для комп'ютера. Але все одно треба мати на увазі, що не всі комп'ютері настільки потужні, щоб запускати всі нові ігри. В зв'язку з цим, у грі треба використовувати не сильно великі за якістю асети та візуальні ефекти. Отже ми обрали платформу для якої буде створюватися гра, тепер треба обрати IDE.

2.3 Вибір мови програмування

Сьогодні нараховують приблизно 300 мов програмування. Кожна по своїй суті унікальна, але є й ті, які схожі або одна походить від іншої. Одна мова програмування більш підходить для розробки сайтів, інша для програмного забезпечення, а третя для розробки ігор.

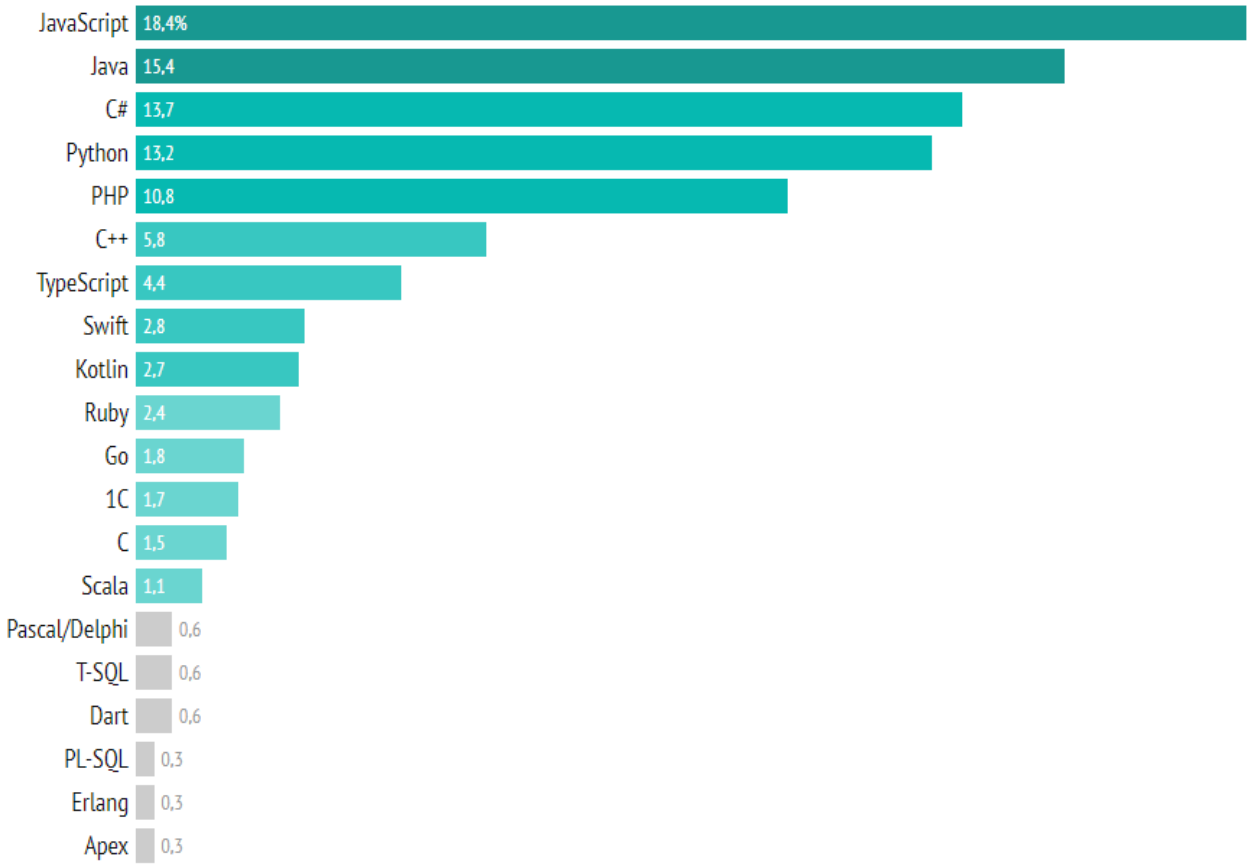

Рисунок 2.2 Рейтинг використання мов програмування

Проглянувши рейтинг використання мов програмування, можна побачити, що на 3-му місці знаходиться C#. Данну мову програмування використовують приблизно 13,7 відсотків ком’юніті розробників. Така популярність пов’язана з тим, що C# доволі простий для вивчення та може використовуватися у декількох напрямків, наприклад написання програмного забезпечення або розробки ігор. Розробка ігор це те що нам і потрібно.

C# – сучасна об'єктно-орієнтована і типобезпечна мова програмування. Ця мова програмування дозволяє розробникам створювати різні типи надійних

програм, що виконуються в .NET. C# відноситься до широко відомого сімейства мов C.

Складання сміття автоматично звільняє пам'ять, зайняту недосяжними об'єктами, що не використовуються. Типи, що допускають значення null, забезпечують захист від змінних, які посилаються на виділені об'єкти. Обробка винятків надає структурований та розширюваний підхід до виявлення помилок та відновлення після них. І це лише невеликий перелік функцій, який допомагає розробити надійне програмне забезпечення

Використання мови програмування C#:

- Розробка ігор
- Віртуальна реальність
- Бекенд
- Дескторні та мобільні додатки

Переваги мови програмування C#:

- Висока популярність
- Інтегрується з Windows та .NET
- Легка для вивчення
- Велика кількість бібліотек

Програми на C# виконуються в .NET, віртуальній системі виконання, яка викликає середовище Common Language Runtime (CLR) та набір бібліотек класів. Common Language Runtime (CLR) – це реалізація загальномовної інфраструктури Common Language Infrastructure (CLI) Microsoft. CLI – це основа для створення середовищ виконання та розробки, в яких мови та бібліотеки працюють прозоро одна для одної.

Вихідний код, написаний C#, компілюється в проміжну мову (IL), відповідний специфікації CLI. Код та ресурси IL, включаючи растрові зображення та рядки, зберігаються у збірках, зазвичай з розширенням .dll.

Платформа .NET Framework складається із загальномовного середовища виконання (середовища CLR) та бібліотеки класів .NET Framework. Основою

платформи .NET Framework є середовище CLR. Середовище виконання можна вважати агентом, який керує кодом під час виконання та надає основні служби, такі як управління пам'яттю, управління потоками та віддалену взаємодію.

При цьому середовищем накладаються умови суворої типізації та інші види перевірки точності коду, що забезпечують безпеку та надійність. Фактично основним завданням середовища виконання є керування кодом. Код, який звертається до середовища виконання, називають керованим кодом, а код, який не звертається до середовища виконання, називають некерованим кодом.

Бібліотека класів є комплексною об'єктно-орієнтованою колекцією повторно використовуваних типів, які застосовуються для розробки додатків - починаючи зі звичайних додатків, що запускаються з командного рядка, та додатків з графічним інтерфейсом (GUI) і закінчуючи програмами, що використовують останні технологічні можливості ASP.NET, такі як веб-форми та веб-служби XML.

```
using System;

class Hello
{
    static void Main()
    {
        Console.WriteLine("Hello, World");
    }
}
```

Рисунок 2.3 Приклад написання коду на C#

2.4 Вибір середовища розробки

Незалежно від того, чи займаєтеся ви професійною розробкою або тільки починаєте, будь-якому розробнику програмного забезпечення необхідне середовище програмування для візуалізації результатів написання коду. Інтегроване середовище розробки зазвичай називається програма, яка створює програмне забезпечення. Він поєднує кілька інструментів, призначених для

спрощення написання і тестування коду, включаючи редактори коду, засоби автоматизації, налагоджувачі та компілятори або інтерпретатори.

Структура IDE дуже функціональна, але для недосвідчених програмістів це може бути недоліком, тому що для використання потрібні базові знання щодо настроювання середовища та, по можливості, потужний комп'ютер.

Редактори текстового коду швидше запускаються і легше освоюються, але їх можливості обмежені стандартним набором функцій: підсвічуванням синтаксису, модифікацією коду і форматуванням.

Переваги використання IDE:

- Автозаповеннння
- Навігація
- Пошук помилок
- Економія часу
- Рефакторинг
- Організація файлів
- Зручне додавання бібліотек
- Можливість налаштування

Для написання коду, встановлення бібліотек та побудови великих систем на мові програмування C# існує багато середовищ розробки, а саме:

- Eclipse
- NetBeans
- Visual Studio
- Project Rider

Eclipse - одна з найбільш функціональних IDE з відкритим вихідним кодом. Спочатку він в основному використовувався для розробки Java, але тепер він підтримує більш широкий спектр мов. Ця IDE має відмінний графічний інтерфейс користувача і функцію перетягування.

Eclipse доступний для Windows, Linux та MacOS. Фреймворк надає широкі можливості, такі як автоматичний аналіз коду, інтеграція з git, статичний аналіз коду і т.д.

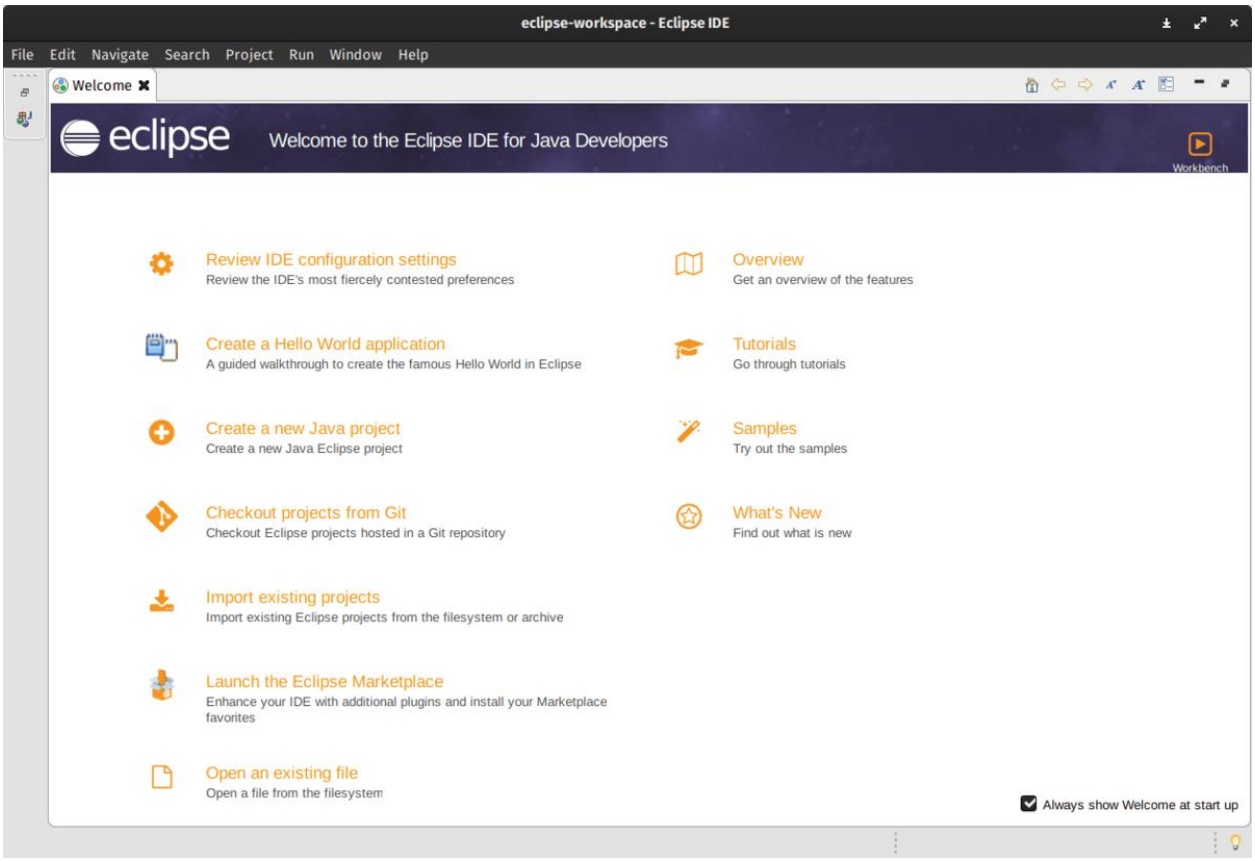

Рисунок 2.4 Редактор коду Eclipse

NetBeans – ще одна найкраща IDE для програмування на C#. Він має зручний інтерфейс, а також кілька дуже корисних шаблонів проектів. Має функцію перетягування. Netbeans написаний на Java, але надає повний набір підтримки та інструментів для розробників C#.

Найкраще в Netbeans – це простий та ефективний інструмент управління проектами. Цей функціонал можна розширити за допомогою різних корисних плагінів. Використовуючи NetBeans, ви можете віддалено контролювати розробку вашого проекту.

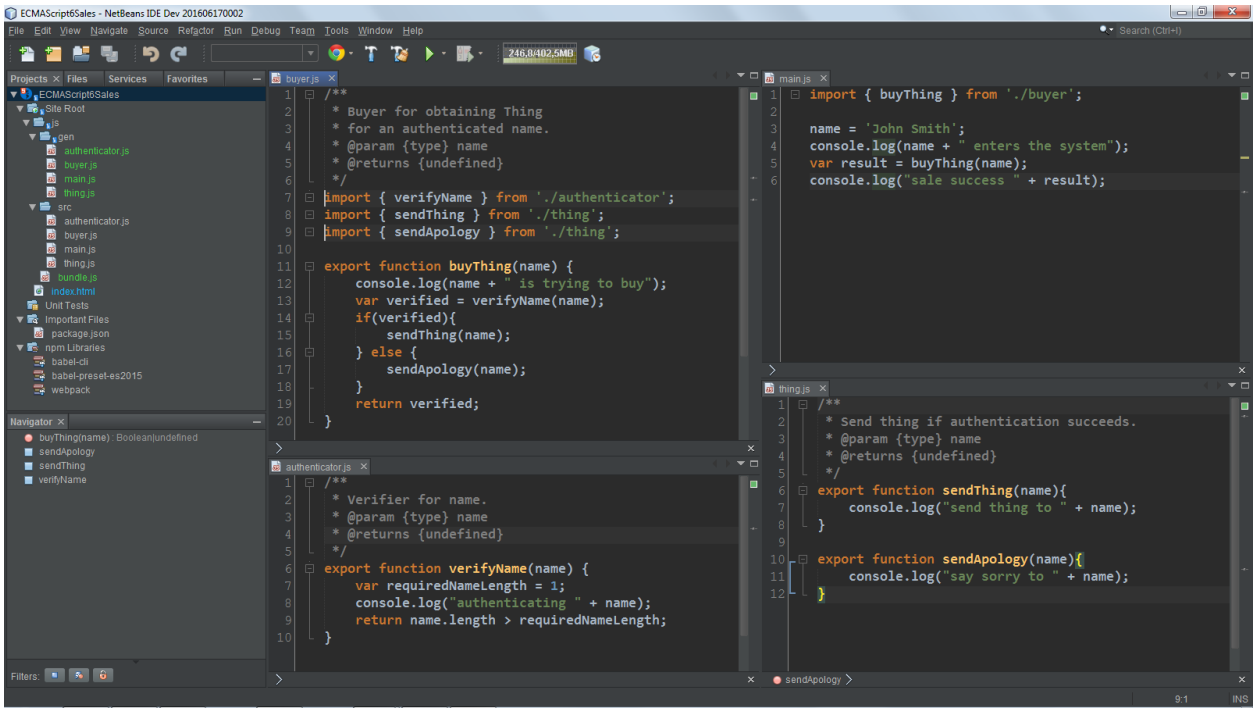

Рисунок 2.5 Редактор коду NetBeans

Visual Studio – одна з найпотужніших та багатофункціональних IDE, доступних для Windows, Linux та MacOS. Ця IDE базується на платформі Electron. Що стосується функцій, Visual Studio має всі необхідні функції, такі як інтелектуальне завершення коду, підсвічування синтаксису, рефакторинг коду, підтримка фрагментів коду, можливості налагодження, вбудований елемент управління Git і багато іншого. Крім того, ви можете налаштувати цю IDE кількома способами, включаючи комбінації клавіш та параметри.

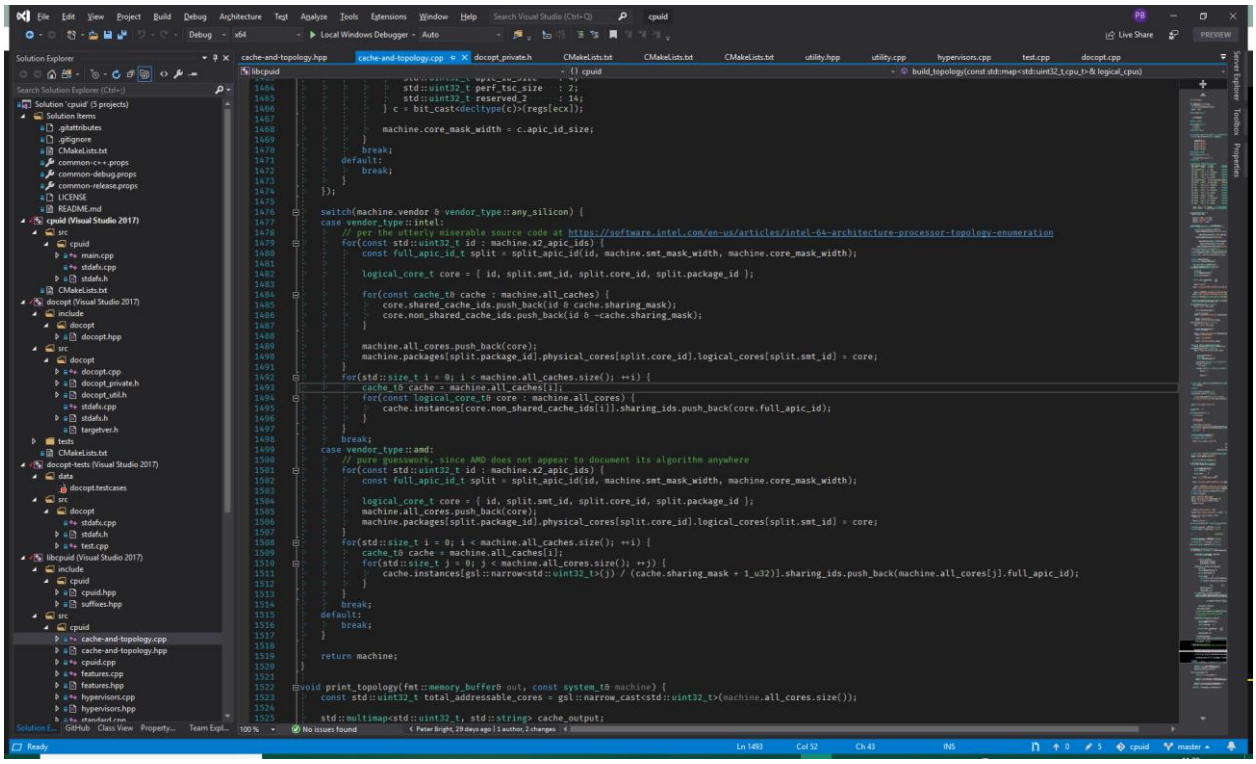

Рисунок 2.6 Редактор коду Visual Studio

Project Rider є кросплатформовою .Net IDE. Вона підходить для використання під Windows, Linux, Mac OS X. Продукт базується на IntelliJ IDEA та Resharper. До її переваг входить таке:

- Підтримка C#, HTML, JavaScript та інших мов.
- Відмінно підходить для створення різного програмного забезпечення.
- Потужна підтримка навігації та рефакторингу.
- Чудово реалізована підтримка інтелектуальних сполучень клавіш.
- Інтеграція з Visual Studio та Unity.

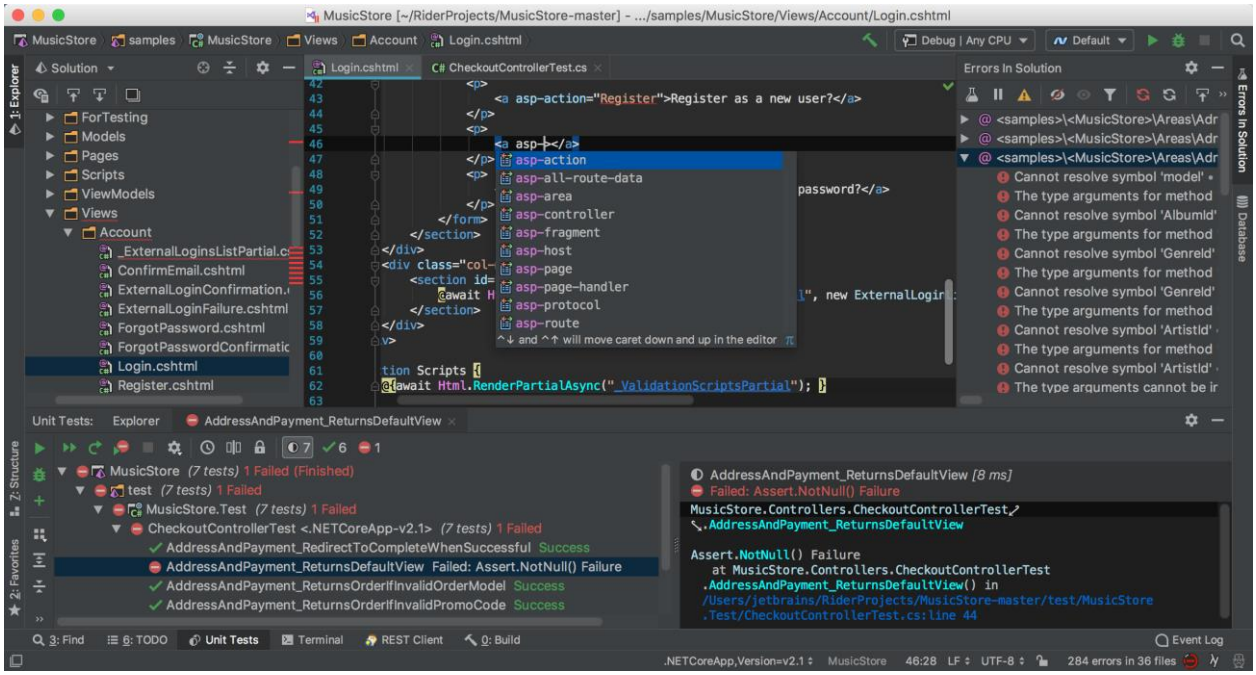

Рисунок 2.7 Редактор коду Project Rider

2.5 Вибір ігрового двигуна

Ігровий двигун має в собі певний контроль над областями гри. Більша частина двигунів мають в собі контроль над наступними областями:

- графіка
- звук
- введення-виведення
- мережа
- ядро

Завдяки цим вбудованим функціоналам простіше буде створення анімацій, написання функціоналу для гри, візуальної обробки проєкту. Застосування таких двигунів допоможе розробнику перенести свою гру на іншу платформу, без будь-яких великих змін в коді. Саме двигуни розв’язують руки розробникам і дають їм величезний функціонал, який можна використати при створенні гри. Плюси ігрових двигунів:

- Не виникає серйозних проблем при зміні платформи

- Використання різних плагінів та інструментів для пришвидшення процесу розробки
- Є можливість інтегрування в хмарні сервіси

Тож вибрати один найбільш підходящий двигун, це доволі відповідальна задача. Бо треба дивитися на інструментарій, швидкість роботи та зручність використання двигуна з великого списку існуючих.

Приклади ігрових двигунів:

- Unreal Engine. Один із найпопулярніших, відомих та зручних ігрових двигунів. Має великий список інструментів та здатність підключати до нього інші плагіни. Доволі багато великих студій користується саме ним. Також його цінять за те, що за його допомоги можна зробити дуже привабливу гру.
- Unity. Є одним із лідерів серед ігрових двигунів. Їм користуються як і великі компанії так і новачки, які хочуть зробити свою першу гру. Привітливий інтерфейс, кросплатформеність та функціонал виділяють його серед інших двигунів.
- CryEngine. Доволі відомих ігровий двигун, який приваблює своєю можливістю зробити гарну гру. За допомогою нього було створено ігри, про які знали дуже багато гравців по всьому світу.
- RPG Maker. Достатньо зручний інтерфейс, великий функціонал та гнучкість налаштувань робить його одним з улюблених двигунів серед початківців.
- Godot. Чудово підходить для створення 2D та 3D ігор. Безкоштовний у використанні та з відкритим кодом. Має аудиторію, яка допомагає покращувати роботу двигуна.

Так як вибір мови програмування прийшовся на C#, то дуже добрим вибором буде саме двигун Unity. За допомогою скриптиів цього ігрового двигуна можна розробляти майже всі елементи гри. Unity підтримує скрипти C# у відповідності до

двух основних підходів: традиційним або об'єктно-орієнтованим, що широко використовується.

Проаналізувавши список багатьох двигунів прийнято рішення використовувати ігровий двигун Unity.

2.6 Опис інструментарію ігрового двигуна

В основна частина інтерфейсу ігрового двигуна Unity – це вікна. Кожне вікно відповідає за ту чи іншу задачу. Перелік вікон:

1) Сцена (Scene)
2) Проєкт (Project)
3) Панель інструментів (Toolbar)
4) Консоль (Console)
5) Ієрархія (Hierarchy)
6) Оглядач (Inspector)
7) Ігрове вікно (Game)

Сцена містить в собі графічне оформлення кожного об’єкту, котрий буде туди поміщений. Це може бути куб, сфера, циліндр, площина, спрайт чи навіть елемент інтерфейсу. Сцена необхідна для того, щоб вистроїти рівень та побачити, як він буде виглядати безпосередньо у самій грі.

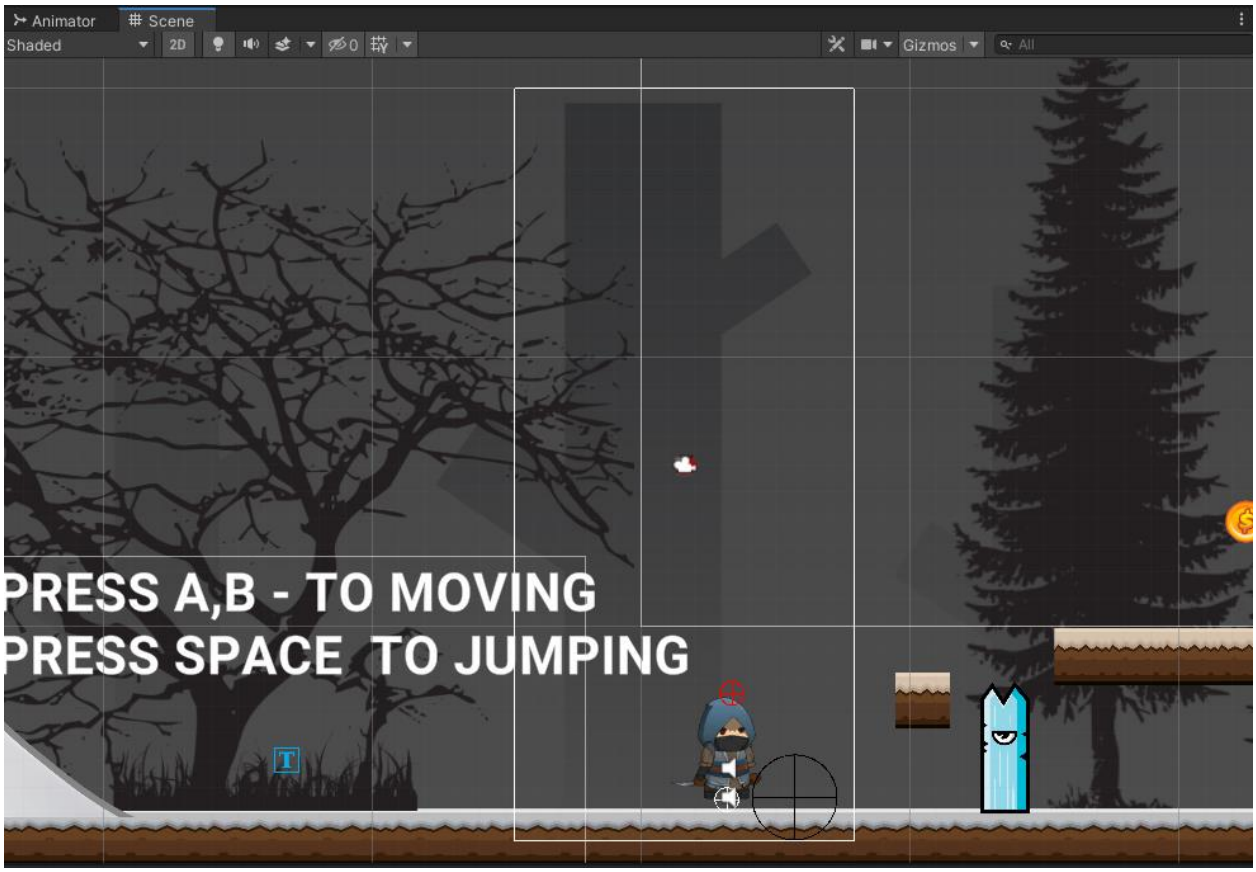

Рисунок 2.4 Сцена (Scene)

Ігрове вікно потрібне для того, щоб подивитися, як саме виглядає гра після того, як ви створили рівень чи інтерфейс. Ну і звісно, щоб зробити тести функціоналу, який ви створили.

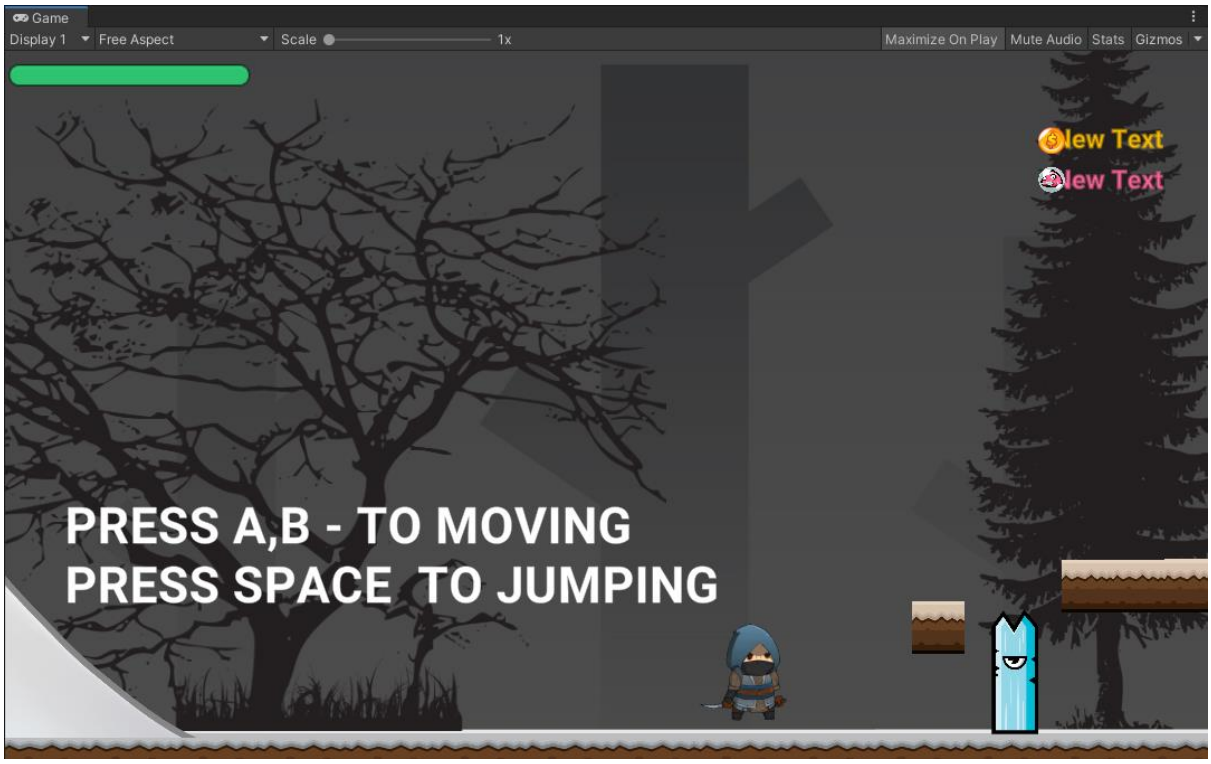

Рисунок 2.5 Ігрове вікно (Game)

Проєкт показує все що в ньому знаходиться. Файли з кодом, звуки, сцени, спрайти, анімації і т.д. Завдяки ньому можна швидко знайти необхідний елемент та переключитися на нього чи змінити.

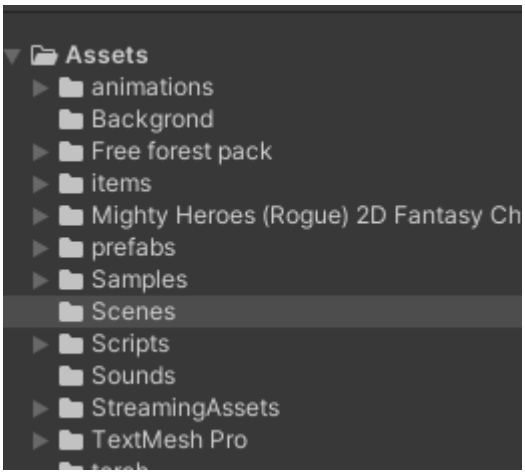

Рисунок 2.6 Проєкт (Project)

Панель інструментів один з найважливіших вікон в двигуні. З його допомогою розробник може змінювати та налаштовувати об’єкти на сцені так як

йому потрібно. Зробити об’єкт більше, перемістити його, змінити колір та повернути його на 180 градусів, це все можливо із-зі панелі інструментів.

Рисунок 2.7 Панель інструментів (Toolbar)

Завдяки консолі можна побачити результат певних дій на сцені. Консоль може показати помилку при її появі, вивести якесь повідомлення, якщо це прописано в коді гри чи попередити, що якийсь об’єкт не використовується та лише займає пам’ять.

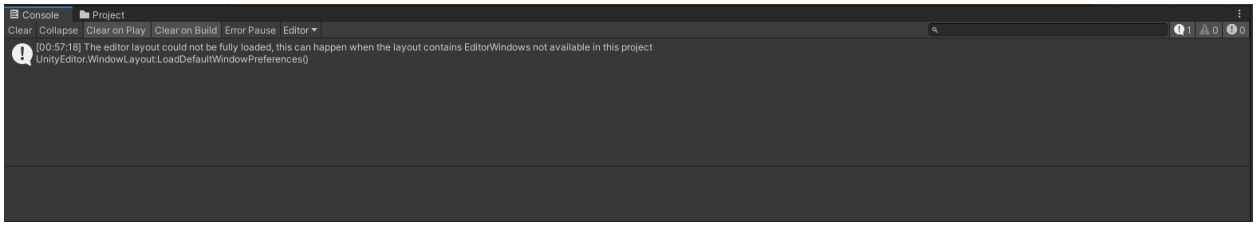

Рисунок 2.8 Консоль (Console)

Ієрархія дуже корисна річ для того, щоб побачити, що саме знаходиться на сцені. Кваліфіковані розробники тримають ієрархію в чіткому порядку. Це робиться для того, щоб знайти певний об’єкт, коли на сцені вже складно його знайти. Також вона дає змогу виключити або включити видимість певного об’єкту на сцені чи навіть настроїти слої інтерфейсу для коректного відображення.

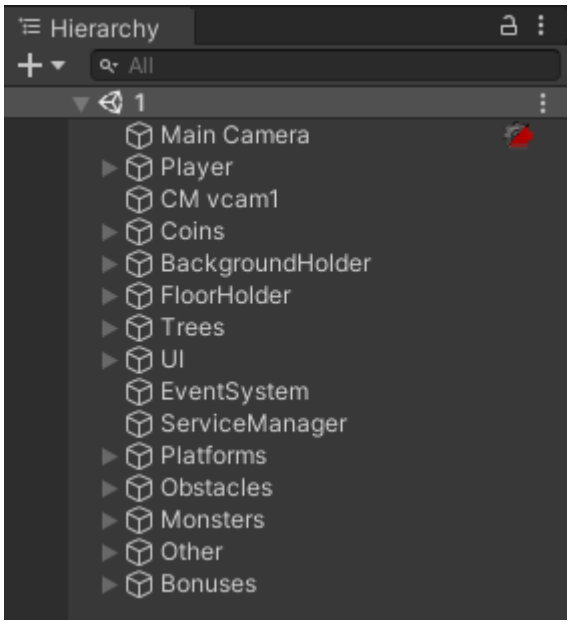

Рисунок 2.9 Ієрархія (Hierarchy)

Оглядач також являє собою одним із найважливіших частин ігрового двигуна. Він надає змогу чітко налаштувати об’єкт на сцені. В його функціонал входить: переміщення об’єкту, змінення його розміру, додавання певної характеристики, підключення скрипту та навіть відображення на сцені і у самій грі.

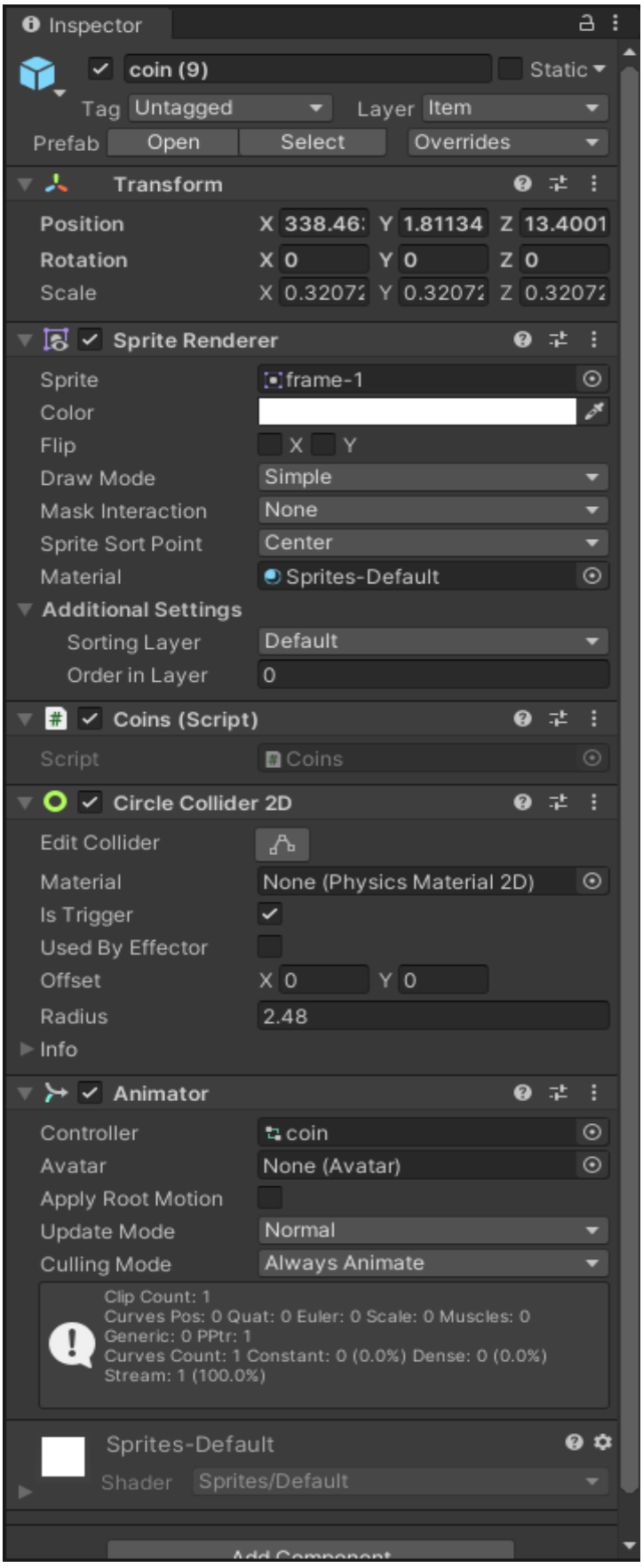

Рисунок 2.10 Оглядач (Inspector)

З англійської мови слово «скрипт» перекладається як сценарій, з чого можна зробити певні висновки. Це набір команд, тобто рядків коду, які виконують конкретне завдання. Для її виконання та створюються скрипти. Вони можуть бути дуже маленькими за обсягом і відповідати за запуск якихось простих служб операційної системи, так і об'ємними, порівнюючи змінні і виводячи результат на сайті.

Скрипт зберігається в текстовому файлі, тому при бажанні його можна легко переглянути і навіть змінити. Цей текстовий файл запускає ланцюжок виконання завдання, який і запрограмований у скрипті. Якщо всі рядки написані правильно та цільові об'єкти вдається знайти, завдання виконується успішно та скрипт спрацьовує.

Скрипти зараз активно інтегруються на сайтах, як приклад можна навести популярну скриптову мову – JavaScript. Однак спочатку вони працювали в операційних системах та виконувались за допомогою внутрішнього синтаксису командної оболонки.

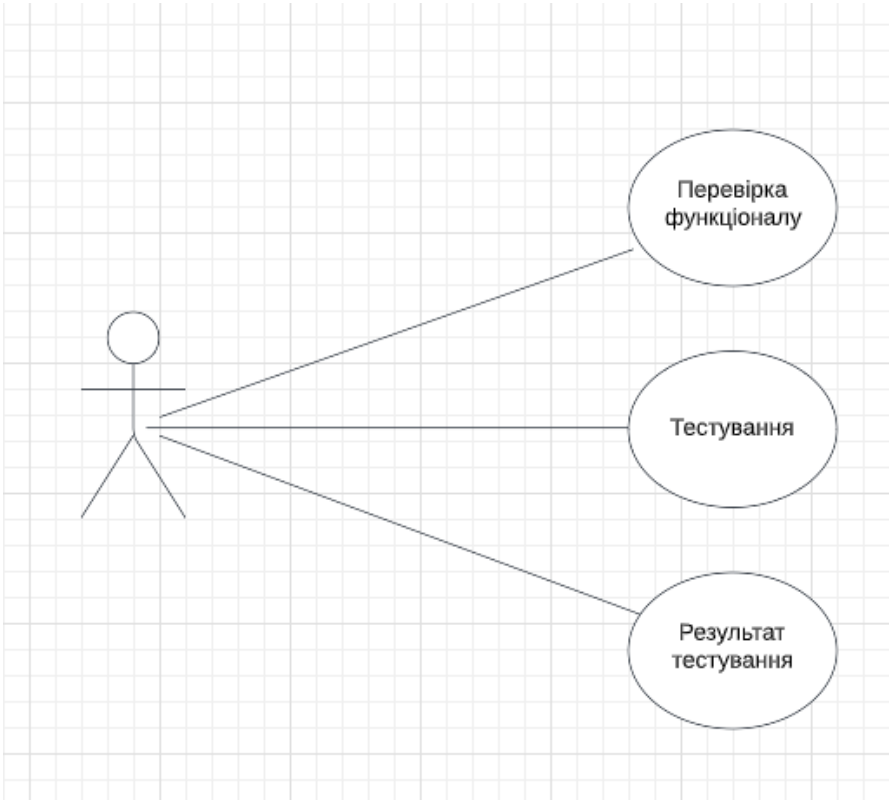

Рисунок 2.11 Діаграма використання

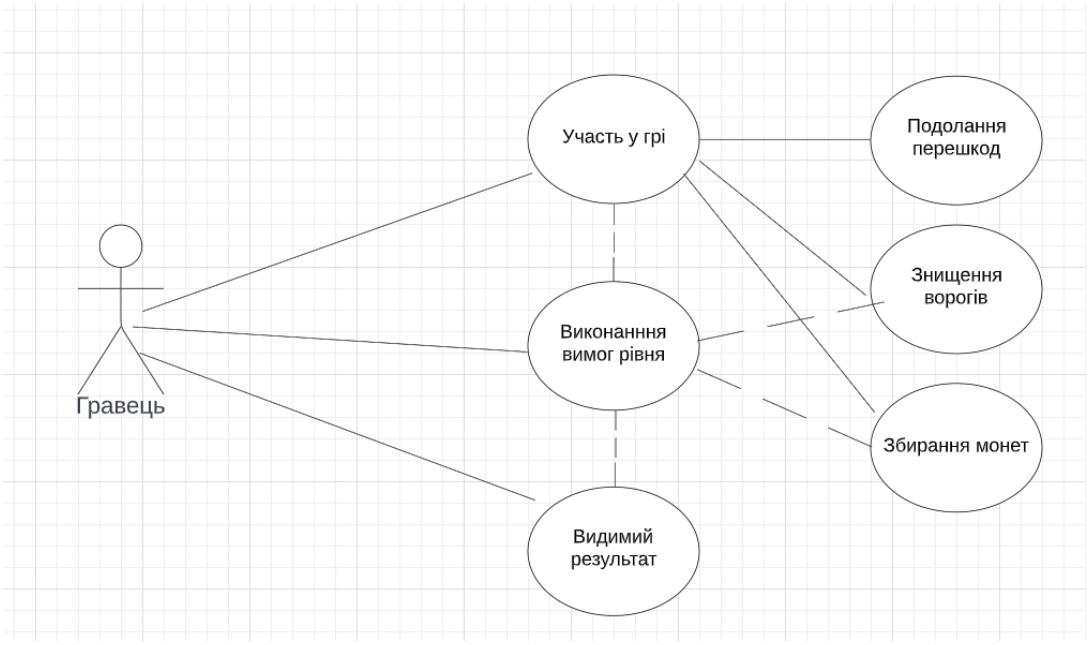

Рисунок 2.12 Діаграма прецедентів

2.3 Створення основної сюжетної ідеї

2.3.1 Сюжетна ідея

Головний герой гри «Cave» - це хлопець, який, подорожуючи світом і шукаючи скарбів, опинився посеред безлюдної зимової пустелі. Така ідея дає змогу окреслити дизайн гри, а також створити варіативність для створення лінійки подальших ігор, які будуть засновані на ідеї іншої сезонності і зміні ландшафту оточуючого середовища головного героя. Головний герой потерпає від лютого зимового холоду і намагається знайти вихід. На заваді головного героя стають зимові монстри, що охороняють скарби. Назва гри «Cave», що перекладається з англійської мови як «Печера» є втіленням символу порятунку, оскільки основну атмосферу гри символізує холод. Печера з вогнем та тріском гілок – символізуватиме можливість погрітися та перепочити для головного героя гри. Ця печера є основною ціллю рівня – дістатися живим до чергової печери, що підтримує життєздатність головного героя. Але печера покликана допомагати тільки найсміливішим і найбільш цілеспрямованим, тому створюється лічильник рівня. Лічильник демонструє вимоги до рівня, що символізують цілі головного героя,

після досягнення яких – печера стане активною. Якщо головний герой захоче скористатися печерою до моменту досягнення своїх цілей – вона буде не активна. Активність печери символізує вогонь, що зягоряється в ній після досягнення усіх цілей (виконання вимог рівна) і характерний тріскіт гілок у полум’ї багаття. Символ печери має закарбуватися в голові Юзера через назву гри, вигляд головного меню і створену атмосферу в цілому.

2.3.2 Унікальні дизайнерські рішення

Асети для створення гри «Cave» були взяті з переліку безкоштовних асетів Юніті. Також для підтримки атмосфери гри та створення у користувача відповідного настрою були створені унікальні спрайти та анімації Асети, що були створенні спеціально для гри «Cave» і є унікальними в своєму роді будуть продемонстровані нижче:

1. Куля з замороженим монстром. Цей об'єкт котиться і наносить урон головному герою.

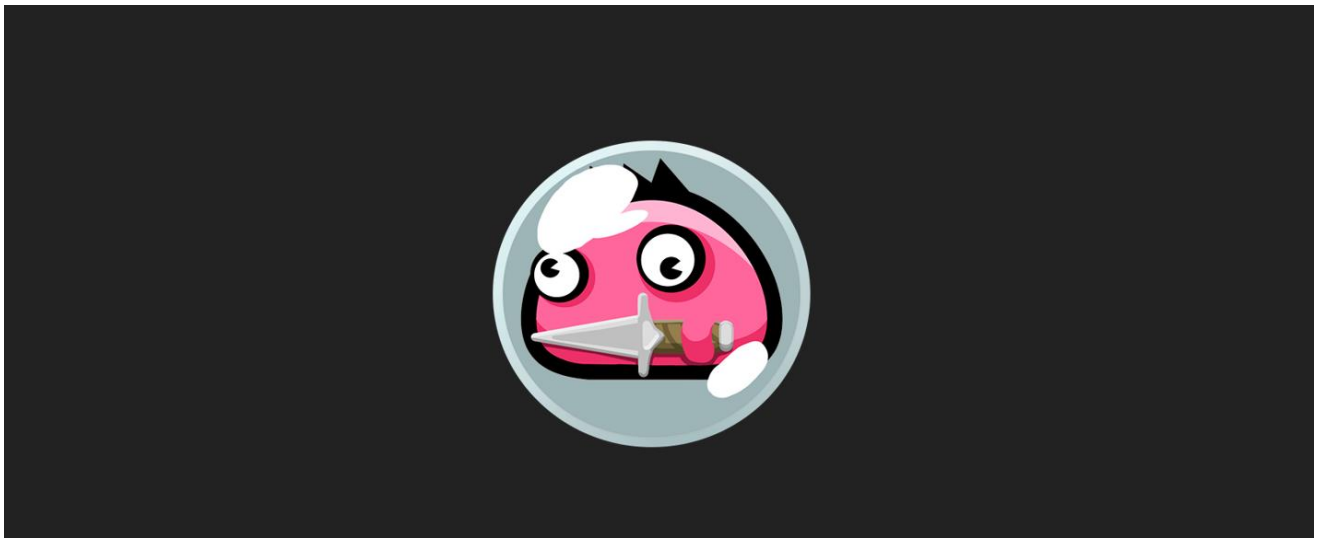

Рисунок 2.13 Куля з замороженим монстром

Під час ігрового процесу цей об’єкт виглядає наступним чином – рис 2.14:

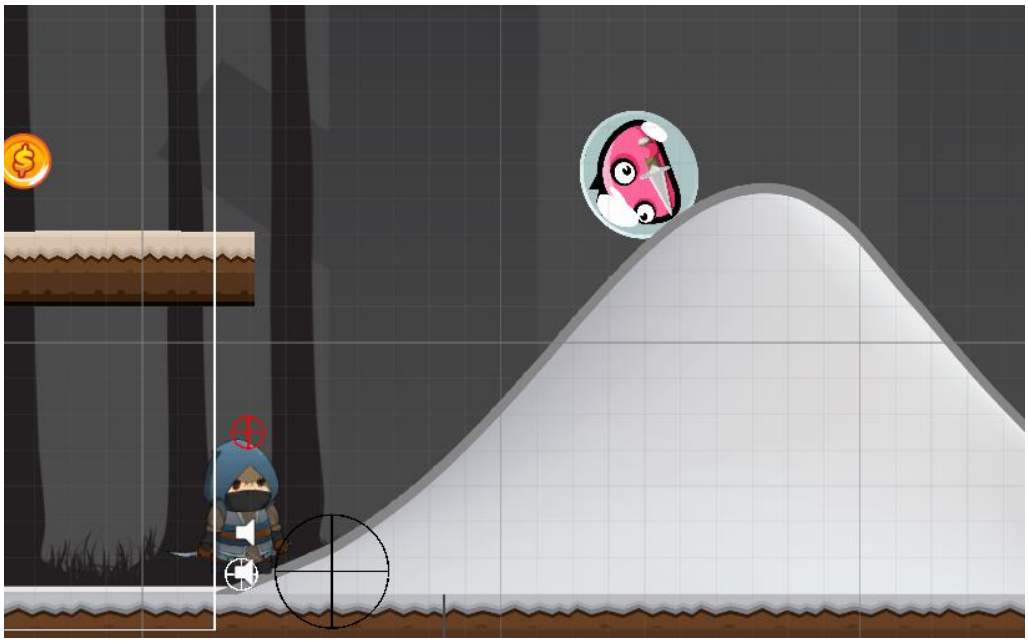

Рисунок 2.14 Куля з замороженим монстром в грі

2. Факел-чекпоінт. Має два стани. Статичний (Доки герой не дійшов до можливості зафіксувати свій прогрес на рівні) та Активний. Чекпоінт переходить в активний стан коли Плеєр підходить до нього, в цей момент загоряється полум’я і спрацьовує анімація. Користувач спостерігає палання вогню і чує характерний звук розгоряння вогню.

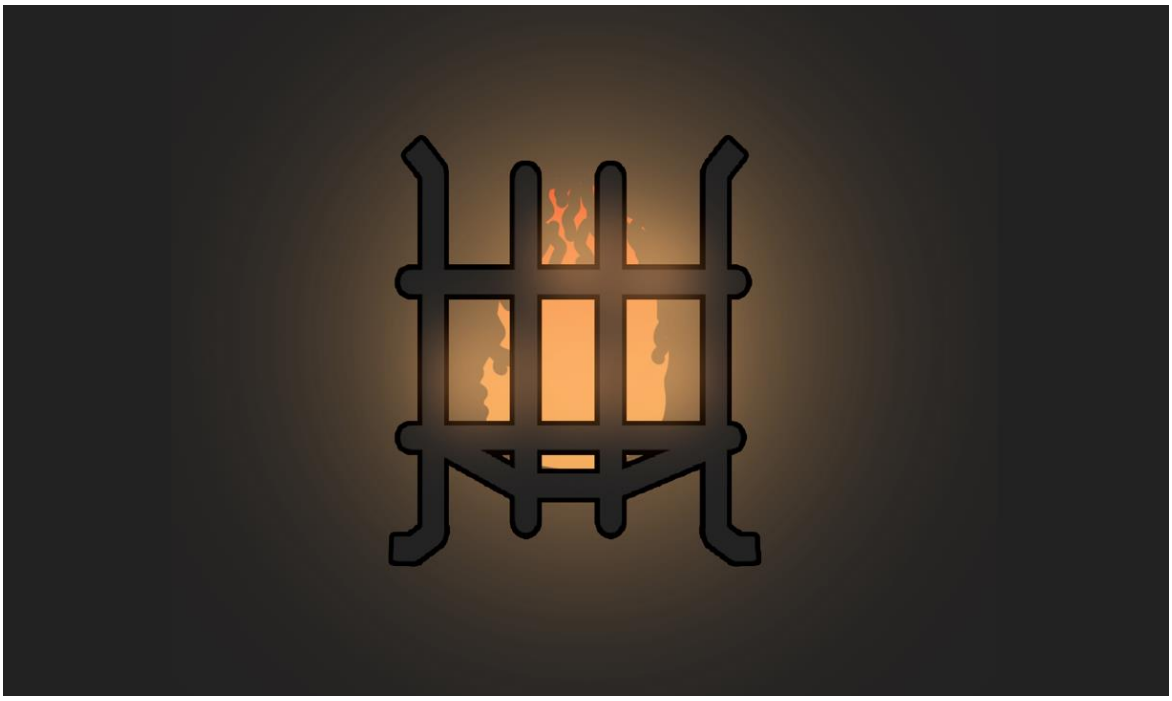

Рисунок 2.15 Факел-чекпоінт в активованому стані

3. Печера (перехід на наступний рівень). Печера має два стани. Статичний, доки не виконані умови переходу на наступний рівень, та активний. Після виконання всіх вимог в печері запалюється полум’я з характерною анімацією та чути тріскіт гілок в багатті.

Рисунок 2.16 Печера переходу на наступний рівень в статичному стані

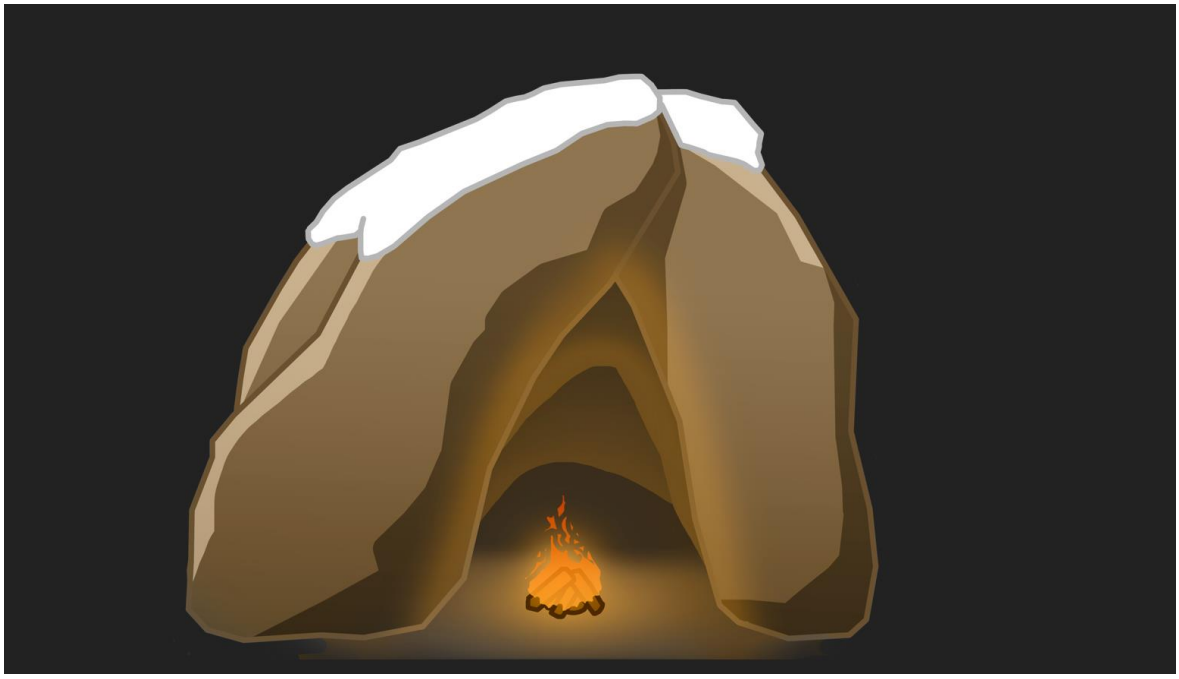

Рисунок 2.17 Печера переходу на наступний рівень в активному стані

3. ПРОЕКТУВАННЯ ОСНОВНИ МЕХАНІК ГРИ

3.1 Основні модулі

3.1.1 Компоненти Player

Проектуємо основні компоненти гри «Cave» в жанрі 2D платформер з використанням ігрового двигуна Unity. Для нашої гри знадобляться наступні компоненти:

1) Головний персонаж

Головний об’єкт за допомогою якого виповняються певні дії з іншими об’єктами. Цим об’єктом буде управляти гравець.

2) Ворог (ENEMY)

Об’єкт який необхідно знищити аби виконати вимоги та перейти на наступний рівень

3) Canvas

Об’єкт в якому містяться усі необхідні елементи інтерфейсу для гравця, а саме: шкала здоров’я, кількість знищених ворогів та кількість підібраних монет.

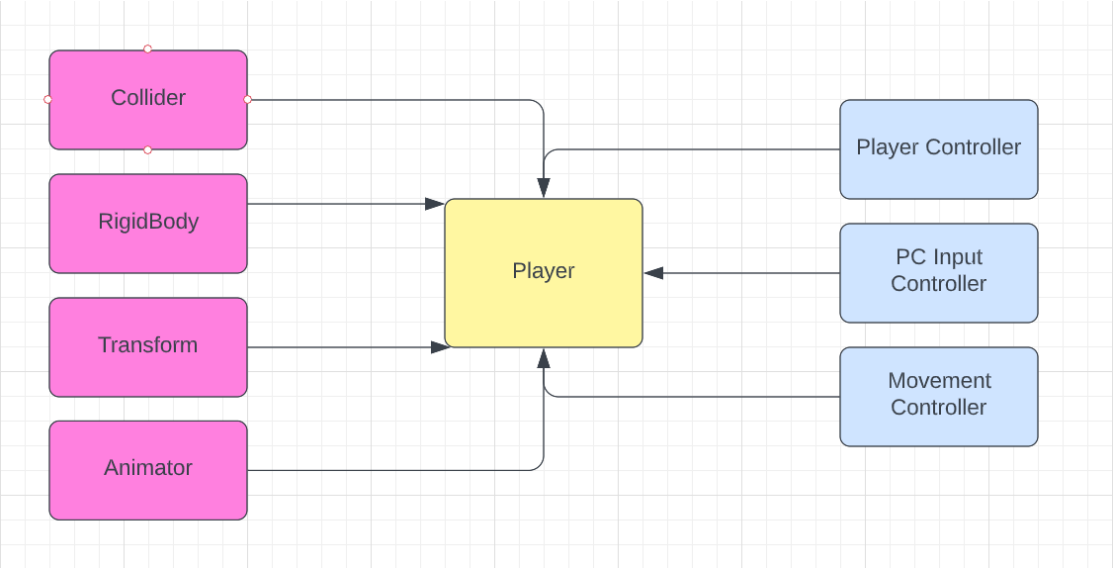

Рисунок 3.1 Компоненти об’єкта «Player»

Компонент Collider. Він буде вказувати зону взаємодії між нашим гравцем та іншими об’єктами на сцені. В нашому випадку коллайдер у вигляді капсули та сфери. Як примітивною, так і безпосередньої форми самого об'єкта. Існують 3 примітивів у коллайдера – Box Collider, Sphere Collider і Capsule Collider. Але якщо форма об'єкта не відповідає даним примітивам, можна використовувати Mesh Collider. Але вони використовують набагато більше навантаження на процесор, ніж примітивні типи. Тому використовувати їх потрібно економно, щоб підтримувати хорошу продуктивність. Однак хорошим правилом використання Mesh Colliders є застосування складових типів примітивних коллайдерів. Яким же чином у нас персонаж потрапляє в ігровий світ, як він пересувається, яким чином відбувається розвиток персонажа. Для цього відповімо спочатку на ці питання. Для цього важливо знати, яким чином працює Unity3D.

Рисунок 3.2 Collider на головному персонажі

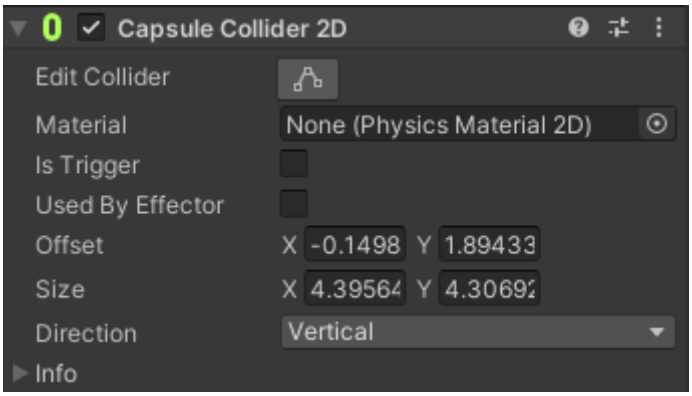

Рисунок 3.3 Характеристики компоненту Collider

Компонент RigidBody. Це той самий компонент, який надає нашому персонажу фізичний стан. Завдяки цьому наш Player зараз має вагу та силу тяжіння. Також можна обмежити пересування у просторі в будь-якій з осей координат.

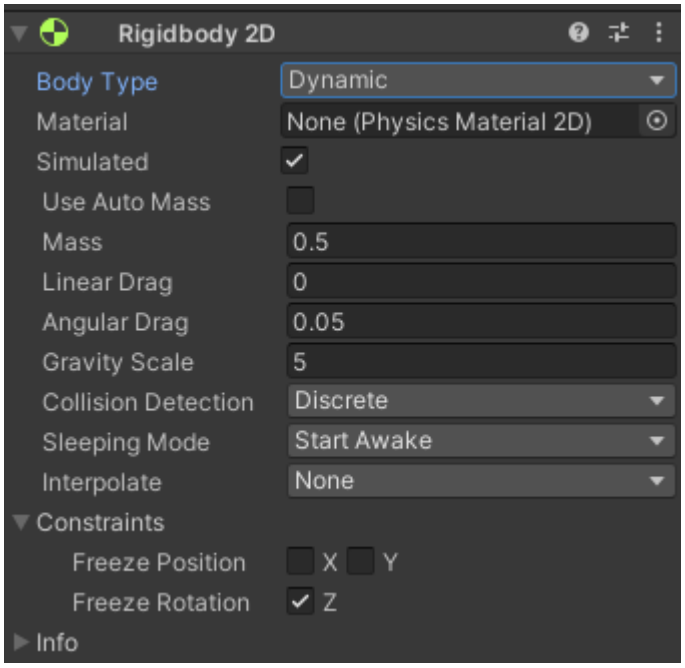

Рисунок 3.4 Характеристики компоненту RigidBody

Компонент Transform. Він відповідає за положення об’єкту на сцені, його розмір та поворот. Даний компонент є в кожному об’єкті, котрий знаходиться на сцені.

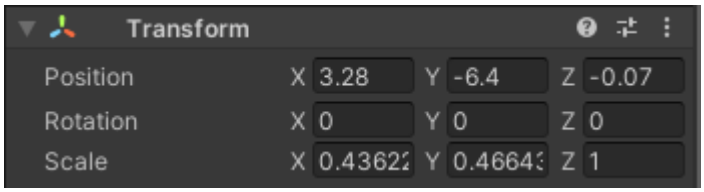

Рисунок 3.5 Характеристики компоненту Transform

Компонент Animator. Відповідно цей компонент допомагає об’єкту змінювати стани в залежності від дій персонажа. Це може бути біг, стрибок, постріл.

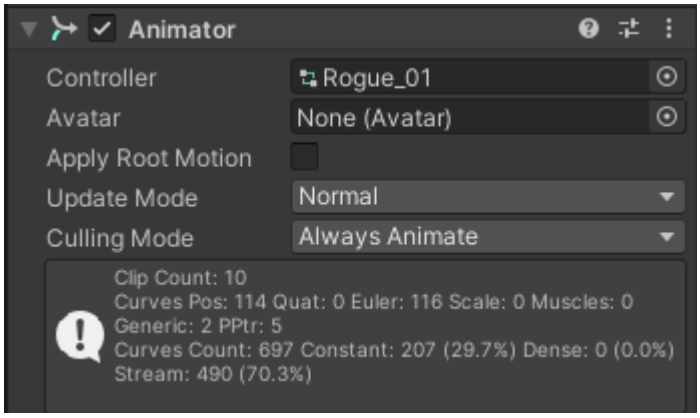

Рисунок 3.6 Характеристики компоненту Animator

Скрипт PlayerController. В цьому скрипті написані основні характеристики Player, такі як: кількість здоров’я, змога отримання пошкодження, змога відновлення здоров’я, позиція для чекпоінту та дії при програші.

```
Сообщение Unity | Ссылок: 0
void Start()
{
    _playerMovement = GetComponent<Movement_Controller>();
    _playerMovement.OnGetHurt += OnGetHurt;
    _HPslider.maxValue = _maxHP;
    _HPslider.value = _maxHP;
    _currentHP = _maxHP;
    _startPosition = transform.position;
    _serviceManager = ServiceManager.Instanse;

}


Ссылок: 6
public void TakeTamage(int damage, DamageType type = DamageType.Casual, Transform enemy = null)

{


    if (!_canBeDamaged)
        return;

    _currentHP -= damage;
    if (_currentHP <= 0)
    {
        OnDeath();
    }

    switch(type)
    {
        case DamageType.PowerStrike:
            _playerMovement.GetHurt(enemy.position);
            break;
    }

    _HPslider.value = _currentHP;


}
```

Рисунок 3.7 Демонстрація скрипту PlayerController

Скрипт PC Input Controller. Цей скрипт задає логіку для пересування. Тобто при натисканні на конкретну кнопку, Player , наприклад, буде рухатися вліво або підстрибне.

```
Сообщение Unity | Ссылок: 0
private void Start()
{
    _playerMovement = GetComponent<Movement_Controller>();
}
Сообщение Unity | Ссылок: 0
void Update()
{
    _move = Input.GetAxisRaw("Horizontal");
    if (Input.GetButtonUp("Jump"))
    {
        _jump = true;
    }

    _crawling = Input.GetKey(KeyCode.C);

    if (Input.GetKey(KeyCode.E))
        _playerMovement.StartCasting();

    if (Input.GetKey(KeyCode.Mouse0))
        _playerMovement.StartAttack();
    if (!IsPointerOverUI())
    {
        if (Input.GetButtonDown("Fire1"))
        {
            _strikeClickTime = DateTime.Now;
            _canAtack = true;
        }

    }


}
```

Рисунок 3.8 Демонстрація скрипту PC Input Controller

Скрипт MovementController. В цьому скрипті написана логіка, як саме буде пересуватися Player, з якою швидкістю, при яких умовах він зможе підстрибнути чи почати повзти. Також вказані стани дії для Animator та звуки.

```
ссылка: 1
public void Move(float move, bool jump, bool crawling)
{
    if (!_CanMove)
        return;

    #region Movement


    if (move != 0 && (_grounded || _AirControl))
        _playerRB.velocity = new Vector2(_speed * move, _playerRB.velocity.y);

    if (move > 0 && !_faceRight)
    {
        Flip();
    }
    else if (move < 0 && _faceRight)
    {
        Flip();
    }
    #endregion
```

Рисунок 3.9 Демонстрація скрипту MovementController

3.1.2 Компоненти Enemy

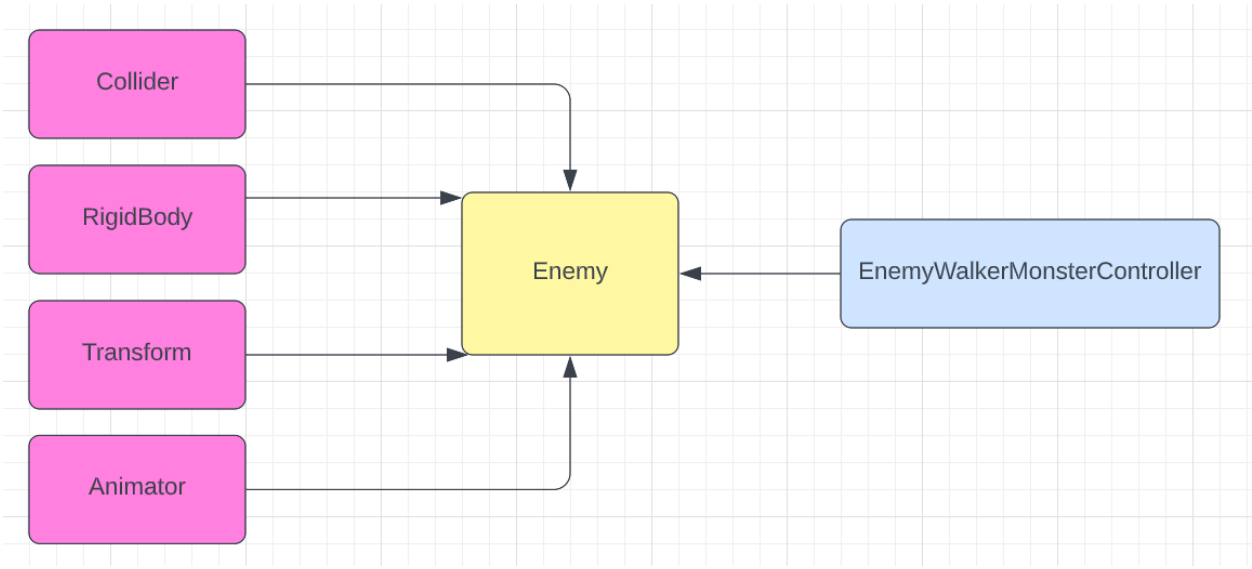

Рисунок 3.10 Компоненти об’єкта «Enemy»

Компонент Collider. Стандартний коллайдер як на рис. 3.3.

Компонент RigidBody. Додає фізичні властивості об’єкту на який був застосований цей компонент. Приклад рис. 3.4.

Компонент Transform. Стандартний компонент, який надає змогу змінювати розмір, кут нахилу та позицію в просторі об’єкта. Приклад рис. 3.5.

Компонент Animator. Компонент який дає можливість змінювати стани об’єкта. Приклад рис. 3.6.

Скрипт EnemyWalkerMonsterController. В цьому скрипті прописана логіка для того, щоб ворог зміг отримувати пошкодження.

```
Скрипт Unity | Ссылок: 0
public class Enemy_WalkingMonster_Controller : EnemiesControllerBase
{
    [SerializeField] private GameObject _projectilePrefab;
    [SerializeField] private Transform _shootPoint;
    [SerializeField] private float _arrowSpeed;
    Ссылок: 4
    public override void TakeDamage(int damage, DamageType type = DamageType.Casual, Transform palyer = null)
    {
        if (type != DamageType.Projectile)

            return;

        base.TakeDamage(damage, type, palyer);
    }

    Ссылок: 0
    public void offHurt()
    {
        _enemyAnimator.SetBool(EnemyState.Hurt.ToString(), false);
    }
}
```

Рисунок 3.11 Демонстрація скрипта EnemyWalkingMonsterController

3.2. Архітектура рівня

Щоб у гру були цікавіше грати і рівні просто не пробігали мимохіть, мною було прийнято рішення додати «виклик» гравцю. Виклик полягає в тому, що треба зібрати усі золоті монети на рівні та знищити усіх ворогів, в іншому випадку – на наступний рівень перейти неможливо. Логіка до цієї вимоги написана в скрипті ServiceManager. Якщо вимога виконана, тоді гравець може перейти на наступний рівень.

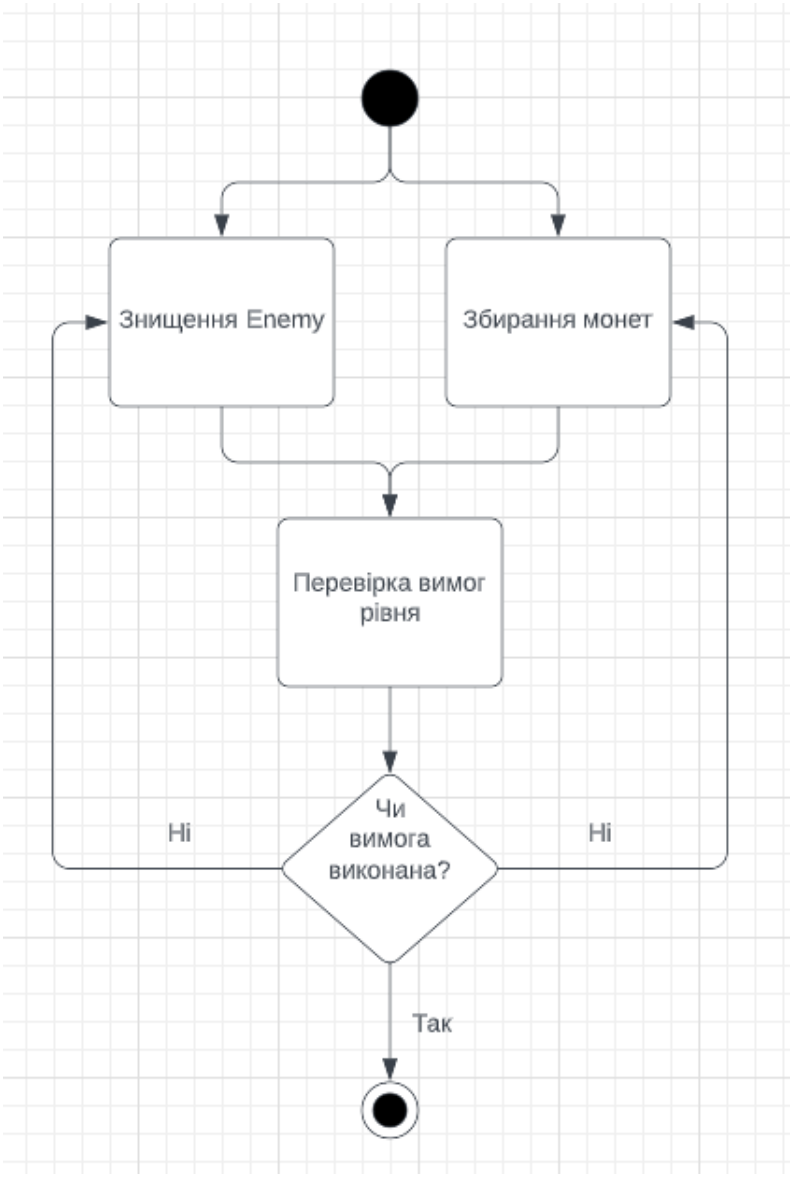

Рисунок 3.12 Вимога рівня для переходу на наступний рівень

```
public void Score ()
{
    _score++;
    _coinScore.text = _score + "/" + _maxCoins.ToString();
    EndLvlChecker();
}

ссылка: 1
public void EnemyScore()
{
    _enemyScore++;
    _enemyscore.text = _enemyScore.ToString() + "/" + _maxEnemies.ToString();
    EndLvlChecker();
}
```

Рисунок 3.13 Демонстрація скрипту ServiceManager

3.3. Ігрове меню

Меню гри зроблено в доволі простому стилі, без зайвих та відволікаючих елементів. Усі кнопки знаходяться по правій стороні екрану. В головному меню можна зайти до налаштувань, розпочати рівень, вибрати конкретний рівень із тих, що гравець вже пройшов та вийти з гри.

4. СТАДІЯ РЕАЛІЗАЦІЇ ГРИ «CAVE»

4.1. Створення головного меню

Головне меню гри наведено на рис. 4.1.

Рисунок 4.1 Меню гри «Cave»

Система меню написана в скрипті ServiceManager.

```
private void Start()
    {
        _lvEnder = LvEnder.Instanse;
        Time.timeScale = 1;

        if (SceneManager.GetActiveScene().buildIndex != 0)
        {
            PlayerPrefs.SetInt(GamePrefs.LastPlayedLvl.ToString(), SceneManager.GetActiveScene().buildIndex);
            PlayerPrefs.SetInt(GamePrefs.LvlPlayed.ToString() + SceneManager.GetActiveScene().buildIndex, 1);
        }
```

```
        _coinScore.text = "0/" + _maxCoins.ToString();
        _enemyscore.text = "0/" + _maxEnemies.ToString();
    }
    public void Restart()
    {
        ChangeLvl(SceneManager.GetActiveScene().buildIndex);
    }

    public void EndLevel()
    {
        ChangeLvl(SceneManager.GetActiveScene().buildIndex + 1);

    }

    public void ChangeLvl(int lvl)
    {
        SceneManager.LoadScene(lvl);
    }

    public void Quit()
    {
        Application.Quit();
        Debug.Log("Quit");
    }

    public void ResetProgres()
    {
        PlayerPrefs.DeleteAll();
}
```

4.2. Створення логіки управління

Переміщення гравця відбувається за допомогою наступного коду:

```
        _move = Input.GetAxisRaw("Horizontal");
        if (Input.GetButtonUp("Jump"))
        {
            _jump = true;
        }

        _crawling = Input.GetKey(KeyCode.C);

        if (Input.GetKey(KeyCode.E))
```

```
    _playerMovement.StartCasting();

if (Input.GetKey(KeyCode.Mouse0))
    _playerMovement.StartAttack();
if (!IsPointerOverUI())
{
    if (Input.GetButtonDown("Fire1"))
    {
        _strikeClickTime = DateTime.Now;
        _canAtack = true;
    }
```

4.3. Створення логіки для Enemy

Логіка для ворога записана у скрипті EnemiesControllerBase

```
private ServiceManager _serviceManager;
protected Rigidbody2D _enemyRb;
protected Animator _enemyAnimator;

[Header("Canavas")]
[SerializeField] GameObject _canvas;

[Header("HP")]
[SerializeField] protected int _maxHp;
[SerializeField] protected Slider _hpSlider;
protected int _currentHp;

[Header("StateChanges")]
[SerializeField] private float _maxStateTime;
[SerializeField] private float _minStateTime;
[SerializeField] private EnemyState[] _availableState;
protected EnemyState _currentState;
protected float _lastStateChange;
protected float _timeToNextChange;

[Header("Movement")]
[SerializeField] private float _speed;
[SerializeField] private float _range;
[SerializeField] private Transform _groundCheck;
[SerializeField] private LayerMask _whatIsGround;
protected Vector2 _startPoint;
protected bool _faceRight = true;

[Header("Damage dealer")]
[SerializeField] private DamageType _collisionDamageType;
[SerializeField] protected int _collisionDamage;
[SerializeField] protected float _collisionTimeDelay;
private float _lastDamageTime;
```

Рисунок 4.2 Демонстрація скрипту EnemiesControllerBase

В цьому скрипті є весь необхідний функціонал, щоб ворог міг переміщатися, знаходити головного героя, наносити атаку та отримувати пошкодження.

Після того як буде завантажений рівень, Unity зчитує скрипти з усіх об’єктів на сцені та дає їм функіонал прописаний в самому скрипті. Кожен раз, коли сцена буде оновлятися, Unity за допомогою методу Update() буде оновлювати дані по кожному об’єкту на сцені. Це дає змогу Unity зрозуміти, які характеристики на даний момент має певний об’єкт і коректно відобразити його на сцені. Також це допомагає необхідно для того, шоб побачити, наприклад, скільки здоров’я має Player на даний момент.

Для запуску готової гри необхідно зібрати все воєдино, а саме: сцени, скрипти, спрайти, звук, текстури. Щоб це зробити треба використати Build Settings у вікни File в ігровому двигуні Unity.

Після цього требя вибрати активні сцени, які будуть застосовуватися в кінцевій грі, вибрати платформу та натиснути Build.

4.4 Створення логіки для монет

Збирання монет – одна із головних задач на рівні. Якщо не зібрати усі монеті, які знаходяться на рівні, то гравець не зможе перейти на наступний рівень. Логіка для збирання монет:

```
void Start()
        {

          _serviceManager = ServiceManager.Instanse;

        }


        private void OnTriggerEnter2D(Collider2D info)
        {

          _serviceManager.Score();
          Destroy(gameObject);
```

```
        }

public void Score ()
        {
          _score++;
          _coinScore.text = _score + "/" + _maxCoins.ToString();
          EndLvlChecker();
                    }
```

4.5 Створення логіки для бонусів

Також на рівні будуть знаходитися різні бонусі, а саме відновлення здоров'я та прискорення. Ще на рівні можна буде знайти так звані точки збереження. Тобто, якщо гравець програє під час проходження рівня, він зможе почати не з початку рівня, а з точки збереження Це необхідно додати на рівень, щоб гравцю не було нудно проходити рівень, а також полегшити його проходження. Логіка для бонуса, що відновлює здоров'я:

```
[SerializeField] private int _speedPlus;



    private void OnTriggerEnter2D(Collider2D info)
     {
info.GetComponent<Movement_Controller>().SpeedMore(_speedPlus);

gameObject.SetActive(false);
      }
```

Логіка для бонуса, що прискорює гравця:

```
[SerializeField] private int _healValue;



    private void OnTriggerEnter2D(Collider2D info)
    {
info.GetComponent<Player_Controller>().RestoreHP(_healValue);
gameObject.SetActive(false);
      }
```

Логіка для точки збереження:

```
    public void Respown (GameObject player)
    {
if (_checkpoint)
{
   Player_Controller.Instanse.CheckpointHP();
   player.transform.position = _spawnPoint.transform.position;

   for (int i = 0; i < _items.Length; i++)
   {
      _items[i].SetActive(true);
   }
}

else
   Restart();
        }
```

Ці скрипти написані окремо один від одного, щоб додати їх до конкретного об’єкту на сцені та не було зайвих помилок.

4.6 Тестування кінцевої гри

Протестуємо нашу гру. Головним об’єктом є наш Player. Він повинен пройти весь рівень, переходячи через перешкоди, знищуючи ворогів та збираючи монети.

Рисунок 4.3 Головний об’єкт – Player

Рівень складається з різноманітних перешкод, ворогів та бонусів у вигляді монет та об’єктів у вигляді сердець, щоб відновити здоров’я.

Рисунок 4.4 Об’єкти на рівні

Щоб пройти на наступний рівень необхідно виконати умови рівня, а саме знищити усіх ворогів та зібрати усі монети.

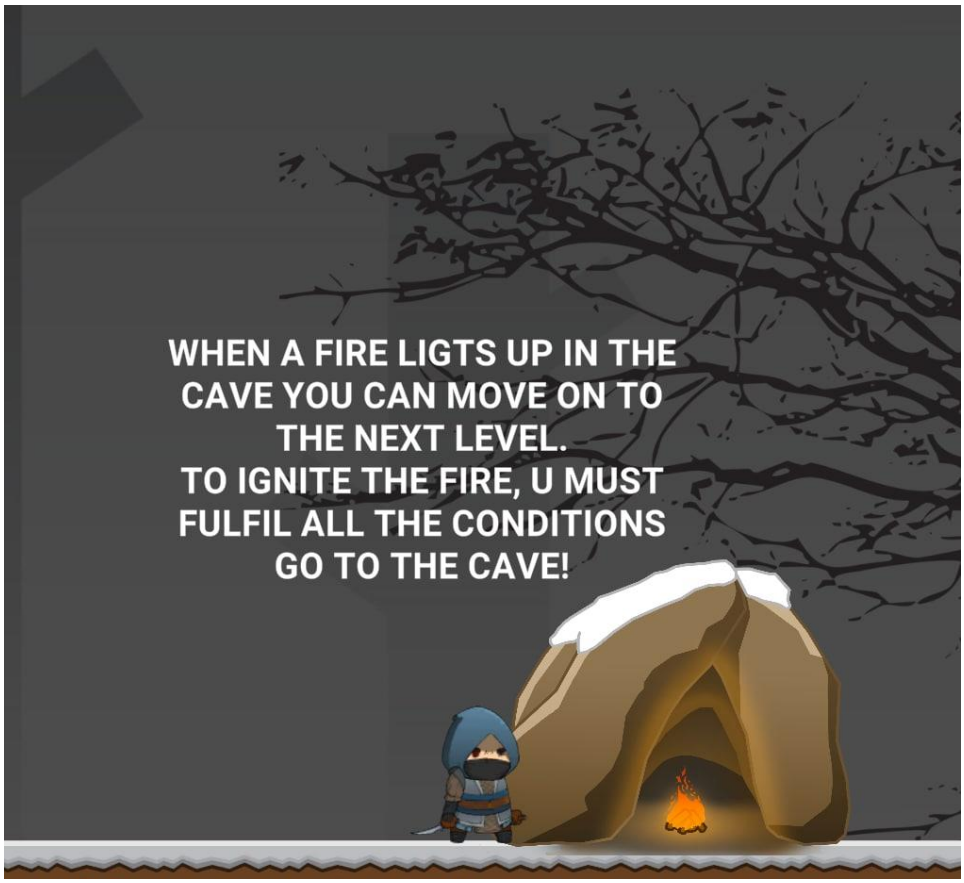

Рисунок 4.5 Перехід на наступний рівень

Під час тестування кінцевого варіанту гри помилок не виявлено. Все працює згідно вимог.

ВИСНОВКИ

В процесі роботи над даним проектом було виконано наступні завдання: 1. Проаналізувати предметну область

- Ознайомитися з видами графіки;
- Обрати жанр гри;
- Проаналізувати проблематику

2. Обрати засоби реалізації

3. Розробити та протестувати гру на ігровому двигуні Unity

- Побудувати структурні діаграми;
- Оглянути основні механіки гри;
- Розробити функціонал гри;
- Провести тестування

Проаналізувавши проблематику впливу відеоігор на дітей було створену гру, яка не займає багато часу на проходження, має різноманітні рівні та цікава. Графіка для гри була підібрана так, щоб око не втомлювалось та не було багато зайвих спец-ефектів. Кожен рівень в цій грі кидає виклик гравцю, що не дає гравцю можливості наскучити.

Ігровий рівень представляє собою лінію перешкод з різних об’єктів та ворогів. Вони розміщені вздовж всього рівня. Кожен об’єкт, це набор різних моделей, починаючи з головного персонажа, закінчуючи бонусами.

При виконанні роботи було застосовано ігровий двигун Unity, який добре себе показав. Двигун Unity доволі потужний, зрозумілий та простий в освоєнні, тому може підійти як для початківців так і для великих компаній.

Проведене тестування показало, що гра Cave у жанрі 2D платформер на двигуні Unity не має помилок та працює плавно та коректно.

ПЕРЕЛІК ПОСИЛАНЬ

1. Uchoose – ресурс доступу: https://uchoose.info/komp-yuterni-igry-ta-koryst-vid-nyh/
2. Cubic – ресурс доступу: https://cubiq.ru/luchshie-2d-shutery-na-pk/
3. Ulab – ресурс доступу: https://ulab.sumdu.edu.ua/uk/10-najkrashhih-igrovih-rushiiv
4. Funduk – ресурс доступу: https://funduk.ua/uk/technoblog/gaming-raznoe/chto-takoe-igrovoy-dvizhok/
5. Oracle NetSuite – ресурс доступу: https://www.netsuite.com/portal/resource/articles/accounting/asset.shtml
6. Heavy.Ai – ресурс доступу: https://www.heavy.ai/technical-glossary/graphical-user-interface
7. Microsoft – ресурс доступу: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
8. Web-proger – ресурс доступу: http://web.spt42.ru/index.php/chto-takoe-unity-3d
9. Gamicis – ресурс доступу: https://gamicus.fandom.com/wiki/2D_platform_video_games
10. LiveJournal – ресурс доступу: https://luckyea77.livejournal.com/2938502.html
11. DOU – ресурс доступу: https://dou.ua/lenta/articles/language-rating-jan-2020/
12. DevEducation – ресурс доступу: https://spb.deveducation.com/blog/luchshie-ide-dlya-c-razrabotchika/
13. OnlineFix – ресурс доступу: https://online-fix.me/games/rpg/16406-risk-of-rain-po-seti.html
14. UKessays – ресурс доступу: https://www.ukessays.com/essays/video-games/2d-and-3d-games.php

Додаток А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ЗВІТ
з переддипломної практики

Виконала студентка 4 курсу
групи ПД-44
Козлова Ю.С.
Керівник роботи
Дібрівний О.А

Київ – 2022

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Галузь розробки комп'ютерних ігор розвивається дуже швидко. І якщо раніше розробники у своїй більшості були чимось незрозумілим для основної маси людей, то зараз ця професія перетворилась у одну із найбільш бажаних.

- **Мета роботи**: засвоєння необхідних знань з основ програмування та формування практичних навичок щодо написання коду
- **Об’єкт дослідження:** створення гри на двигуні Unity.
- **Предмет дослідження:** мова програмування C# та двигун Unity, а також методи їх використання при розробці відеоігор.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

- **Visual Studio**
- **Unity 3D**
- **C#**

3

ВИСНОВКИ

Протягом практики ознайомилася з предметною областю дипломної роботи. Покращила знання про розробку ігор і взагалі індустрії. Визначилася із програмними продуктами та IT-засобами, які буду використовувати під час виконання дипломної роботи. Для розробки гри я обрала Unity. Вона безкоштовна, доволі зрозуміла та має багатий функціонал. Для написання функціоналу гри було обрано C#, тому що ця мова програмування дуже добре підходить під поставлену задачу та об’єднується з двигуном Unity. Як середовище розробки я обрала Microsoft Visual Studio. Це IDE найбільш пристосоване для зручного написання коду на мові програмування C#.

4