Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК /_____/
підпис
«___» _____ 2023 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»
на тему: «2D гра платформер на Unity»

Студент групи КН-41 Цьома І.С. _____
(підпис)

Керівник проекту Посвятовська О.Б. _____
(підпис)

Консультанти:

з техніко-економічного обґрунтування Меленчук Л.І. _____
(підпис)

Нормоконтролер Кульчинська Н.С. _____
(підпис)

Тернопіль – 2023
Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ

Завідувач відділення

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

підпис

«___» _____ 2022 р.

ЗАВДАННЯ

на дипломне проєктування

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»

студенту _____
(прізвище, ім’я та по-батькові студента)

1. Тема проєкту_____
_____
_____
_____

затверджена наказом по коледжу від “___”_____2022 р., №_____

2. Термін здачі студентом завершеного проєкту “___” _____2023 р.

3. Вихідні дані до проєкту _____
_____
_____
_____

4. Перелік питань, які повинні бути розроблені в проєкті:

а) основна частина _____
_____
_____
_____

б) техніко-економічне обгрунтування _____
_____
_____

5. Перелік графічного матеріалу _____
_____
_____

6. Консультанти проєкту: _____

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
з техніко-економічного	_____ (вчена ступень, звання П.І.Б.		

обгрунтування	_____ консультанта)		

КАЛЕНДАРНИЙ ПЛАН
дипломного проєктування

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1.	Вибір теми і ознайомлення з вимогами до дипломного проєкту.	20.10.22 р.	06.12.22 р.
2.	Аналіз типових рішень та написання відповідного розділу проєкту.	15.12.22 р.	23.01.23 р.
3.	Вивчення технологій реалізації та написання відповідного розділу проєкту.	23.01.23 р.	14.02.23 р.
4.	Розробка функіональних вимог до проєкту і робота над структурою програмного продукту. Написання відповідного розділу проєкту.	20.02.23 р.	28.02.23 р.
5.	Налаштування робочого середовища для реалізації та написання відповідного розділу проєкту.	12.03.23 р.	16.03.23 р.
6.	Проєктування програмного засобу і написання відповідного розділу проєкту.	21.03.23 р.	18.04.23 р.
7.	Реалізація і налаштування програмного засобу, а також написання відповідного розділу проєкту.	19.04.23 р.	06.05.23 р.
8.	Вдосконалення модулів.	15.05.23 р.	18.05.23 р.
9.	Тестування і налагодження програмного продукту, а також написання відповідного розділу проєкту.	25.05.23 р.	27.05.23 р.
10.	Аналіз економічних показників дипломного проєкту і створення відповідного розділу.	05.06.23 р.	10.06.23 р.
11.	Редагування та оформлення пояснювальної записки.	10.06.23 р.	14.06.23 р.
12.	Попередній захист дипломного проєкту та усунення помилок.	15.06.23 р.	
13.	Підготовка до захисту дипломного проєкту.	15.06.23 р.	26.06.23 р.
14.	Захист дипломного проєкту.	27.06.23 р.	27.06.23 р.

7. Дата видачі завдання “___” _____ 2022 р.

Керівник _____/ /

Завдання прийняв до виконання _____/ /

Реферат

Результатом виконання завдань дипломного проєктування буде повноцінна 2D гра платформер «Пригоди Пса Патрона». Гра заснована на використанні піксельної графіки та втілена у 2D стилі.

В ході написання дипломного проєкту буде проведено детальний аналіз різних середовищ розробки, зокрема ігрових движків, з метою вибору найбільш підходящого для створення гри. На основі проведеного аналізу буде обрано відповідний движок, що забезпечував потрібні можливості для реалізації задуманої гри.

В рамках проєкту будуть написані скрипти, що відповідають за функціональність гри. Ці скрипти повинні забезпечувати правильне функціонування головного героя та взаємодію з різними об'єктами в грі. Крім того, буде створено рівні та оточення з використанням наявних спрайтів та текстур.

Abstract

The result of completing the diploma design tasks will be a full-fledged 2D platformer game «Пригоди Пса Патрона». The game is based on the use of pixel graphics and embodied in 2D style.

In the course of writing the diploma project, a detailed analysis of various development environments, in particular game engines, will be carried out in order to choose the most suitable one for creating a game. Based on the analysis, a suitable engine will be selected that provides the necessary capabilities for the implementation of the planned game.

As part of the project, scripts responsible for the functionality of the game will be written. These scripts should ensure the correct functioning of the main character and interaction with various objects in the game. Additionally, levels and environments will be created using existing sprites and textures.

ЗМІСТ

ВСТУП

Розробка відеоігор стала важливою частиною індустрії розваг, і мільйони людей у всьому світі щодня проводять час за різними комп'ютерними іграми. Створення успішної відеоігри потребує підготовки сценарію гри, проведення аналітики, проектуання дизайну гри та її функціоналу, дослідження та вибір інструментарію створення, та проведення тестування після реалізації та проведення техніко-економічного аналізу.

Серед різноманіття сучасних комп'ютерних ігор 2D гри платформери займають визначену нішу серед інді-ігор. Вони створюються незалежно від фінансової підтримки, часто є маловартістними або безкоштовними, зазвичай мають невеликий розмір та поширюються як freeware.

Мета даного проекту полягає у тому, щоб застосувати набуті знання та навички за період навчання, шляхом розробки та створення багаторівневої 2D гри платформера, з використанням піксельної графіки.

Актуальність даного проекту полягає в необхідності популяризації україномовного сегменту індустрії розробки ігор. Важливим є використання світових тенденцій при розробці багаторівневого геймплея та включення різних ігрових механік для створення цікавого досвіду для гравців.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІСНУЮЧИХ РОЗРОБОК ІГОР 2D ПЛАТФОРМЕРІВ

1.1 Дослідження предметної області

Платформер - це жанр відеоігор, де головний герой пересувається по рівню, стрибаючи з платформи на платформу, і протистоїть різним ворогам та перешкодам. Цей жанр часто використовує 2D графіку, але також може бути в 3D. У платформерах зазвичай є декілька рівнів з різними складностями та перешкодами, які гравець повинен пройти, щоб завершити гру. Крім того, у платформерах часто є можливість збирати бонуси, що дозволяють гравцю підвищувати свої навички або отримувати додаткові життя. Цей жанр давно став одним з найпопулярніших у світі відеоігор, і він ще досить популярний і досить широко використовується в різних відеоігрових проєктах.

Історія платформерів досить довга і багатогранна. Перші платформери з'явилися в 80-х роках, коли комп'ютерні ігри тільки починали набирати популярність. Найбільш відомими представниками цієї ери є Super Mario Bros., Sonic the Hedgehog, та Castlevania.

У 90-х роках платформери продовжували свій розвиток, але з'явилися також і багатокористувацькі ігри, які почали займати лідируючі позиції в ігровій індустрії. Проте, платформери залишалися популярними серед гравців та розвивалися далі.

Сьогодні платформери можуть бути виконані в різних стилях та на різних платформах. Вони можуть бути 2D або 3D, з відкритим світом або лінійним сценарієм, містити елементи RPG, екшену та інших жанрів. Також платформери можуть бути доступні на різних платформах, таких як ПК, консолі, мобільні пристрої та інші.

У платформерах особливу увагу приділяють геймплею та контролю персонажа. Гравець має змогу керувати рухами свого героя та стрибати від однієї платформи до іншої, уникати перешкод та ворогів, збирати бонуси та

інші предмети. Крім того, платформери часто мають мальовану, або піксельну графіку та яскраву музику, що робить їх більш привабливими для гравців.

Наприклад предмети, що збирає гравець протягом гри, наділяють персонажа додатковими бонусами, які бувають як і вичерпними (на певний рівень чи на короткий проміжок часу), так і до повного завершення гри. Ці предмети зазвичай збираються простим дотиком персонажа і не потребують додаткових дій від гравця. Вороги у грі зазвичай представлені у вигляді так званих чисельних та різноманітних «монстрів», які мають примітивний штучний інтелект та намагаються завдати шкоди персонажу приблизившись максимально близько до нього. Також вони можуть не володіють штучним інтелектом зовсім, переміщатись по певній заданій траєкторії та здійснювати повторювані дії. У більшості випадків зіткнення з ворогом буде призводити до втрати очків життя або навіть смерті головного персонажа. Іноді вороги тікають, видають звуки або змінюють свій зовнішній вигляд при наближенні до гравця. У перших класичних платформерах вороги знешкоджувались звичайним стрибком на них, в той момент як сьогодні майже кожний головний герой платформера володіє якоюсь зброєю для знищення цих монстрів. Переможені монстри після знищення зазвичай зникають або просто провалюються за ігрові елементи.

Деякі платформери мають також елементи головоломок, що потребують від гравця не лише точності та швидкості, але й розумових здібностей для вирішення складних завдань. Наприклад, гравець може знайти спеціальні ключі або лічильники, які дозволяють розблокувати двері до нових рівнів або отримати бонусні рівні. Інші платформери можуть мати елементи RPG, такі як системи квестів, інвентарів і прокачування персонажа, що дозволяють гравцеві збільшувати свої можливості під час гри.

У загальному, платформери є дуже популярним жанром відеоігор, який має велику кількість різних варіацій та елементів, що робить кожну гру унікальною та цікавою для гравців різного рівня умінь та досвіду.

1.2 Аналіз існуючих розробок

Для аналізу було відібрано декілька платформерів, які займали топові позиції на онлайн сервісах Steam, Origin та Gog :

– Super Meat Boy;

– Broforce;

– Hollow Knight;

– Starbound;

– Deadlight.

1.2.1 Super Meat Boy

Super Meat Boy - це інді-гра, розроблена командою Team Meat і випущена у 2010 році для Microsoft Windows [6]. Гра стала дуже популярною, завдяки своїй геймплейній складності та ностальгійному візуалу, який нагадує 8-бітні ігри. Меню вибору рівня в грі Super Meat Boy поданно на рисунку 1.1.

Рисунок 1.1 – Меню вибору рівня в грі Super Meat Boy.

У грі ви керуєте головним героєм - кубиком м'яса на ім'я Meat Boy, який повинен пройти через багато рівнів, щоб врятувати свою кохану, Бінді, від злого ворога, Dr. Fetus. У кожному рівні Meat Boy повинен долати перешкоди, уникаючи пасток і небезпек, таких як лазери, гострі шипи, ріжучі пилки та інші.

У загальному, Super Meat Boy є дуже захоплюючою грою, що дозволяє гравцям відчути себе часткою класичних ігор та пройти багато цікавих та

складних рівнів. Гра є доступною на багатьох платформах, включаючи Xbox 360, PlayStation 4, Nintendo Switch та інші.

Геймплей Super Meat Boy дуже вимогливий та швидкий, і вимагає від гравця швидких реакцій та високої точності у керуванні. Одна з головних особливостей гри – це можливість миттєво повторювати рівні, щоб змагатися зі своїми друзями або покращувати свій власний рекорд.

До переваг даної гри можна віднести наступне:

– Складність: гра має дуже високий рівень складності, що робить її викликом для досвідчених гравців. Це забезпечує високий рівень задоволення від проходження гри та може стимулювати до покращення вмінь у грі.

– Геймплей: гра має дуже динамічний та рухливий геймплей, що робить її дуже захоплюючою та не дає нудьгувати гравцям.

– Графіка: гра має незвичайний та яскравий стиль графіки, що додає їй характеру та створює унікальну атмосферу.

– До недоліків можна віднести наступне:

– Складність: високий рівень складності може стати проблемою для новачків та менш досвідчених гравців, що може зменшити задоволення від гри та здатність до її проходження.

– Незручний контроль: деякі гравці можуть вважати, що контроль над головним героєм гри не дуже зручний або неінтуїтивний.

– Монотонність: після деякого часу гра може стати монотонною, особливо якщо гравець не може пройти рівень та повторює його декілька разів.

Узагалі, гра Super Meat Boy може бути дуже захоплюючою для тих, хто любить складні та динамічні інді-ігри. Проте, вона не підійде для всіх гравців через свій високий рівень складності та незручний контроль.

1.2.2 Broforce

Це інді-гра, з жанром бойовик і платформер [7], розроблена командою Free Lives і випущена у 2014 році для Microsoft Windows. Гра отримала популярність завдяки своїй веселій атмосфері, геймплейній складності та аркадному стилю.

У грі ви керуєте командою героїв, що називається Broforce, кожен з яких базується на класичних персонажах з фільмів та відеоігор, таких як Рембо, Термінатор, Індіана Джонс та інші. Кожен герой має свої унікальні здібності та спеціальну зброю.

Рисунок 1.2 – Битва з босом на одному із рівнів гри Broforce

Головна мета гри – це пройти через різні рівні, вбиваючи ворогів та знищуючи структури. Гра пропонує широку кількість різноманітних місій, де гравці повинні виконувати різні завдання, такі як знищення ворогів, збирання предметів та інше.

Гра має дуже швидкий темп та вимагає від гравців швидкість реакції та високої точності у керуванні. Одна з головних особливостей гри - це можливість знищувати все на своєму шляху, використовуючи велику кількість різноманітної зброї, такої як кулемети, гранати та інші.

До переваг даної гри можна віднести наступне:

– Геймплей: гра має дуже швидкий та наповнений екшеном геймплей, що робить її дуже захоплюючою та незабутньою. Гравці можуть вибирати з великої кількості різних персонажів, кожен з яких має свої унікальні здібності та зброю.

– Гумор: гра має веселий та іронічний стиль гумору, що робить її дуже приємною для гравців та додає до загального настрою гри.

– Графіка: гра має стильну та яскраву 8-бітну графіку, що додає їй характеру та створює унікальну атмосферу.

– До недоліків можна віднести наступне:

– Нестабільність: деякі гравці можуть стикнутися з проблемами стабільності гри, такими як затримки або збої.

– Коротка тривалість: гра має досить коротку тривалість та невелику кількість рівнів.

Узагалі, Broforce є дуже захоплюючою та веселою грою з високим рівнем екшену та гумору. Проте, деякі гравці можуть стикнутися з проблемами стабільності та короткою тривалістю гри, що може зменшити загальні враження від гри.

1.2.3 Hollow Knight

Hollow Knight - це інді-гра, платформер розроблено командою Team Cherry і випущена в 2017 році [8] для Microsoft Windows та macOS. Гра отримала високі оцінки критиків та широку популярність серед гравців завдяки своєму унікальному стилю, захоплюючій геймплейній механіці та вражаючій візуальній графіці.

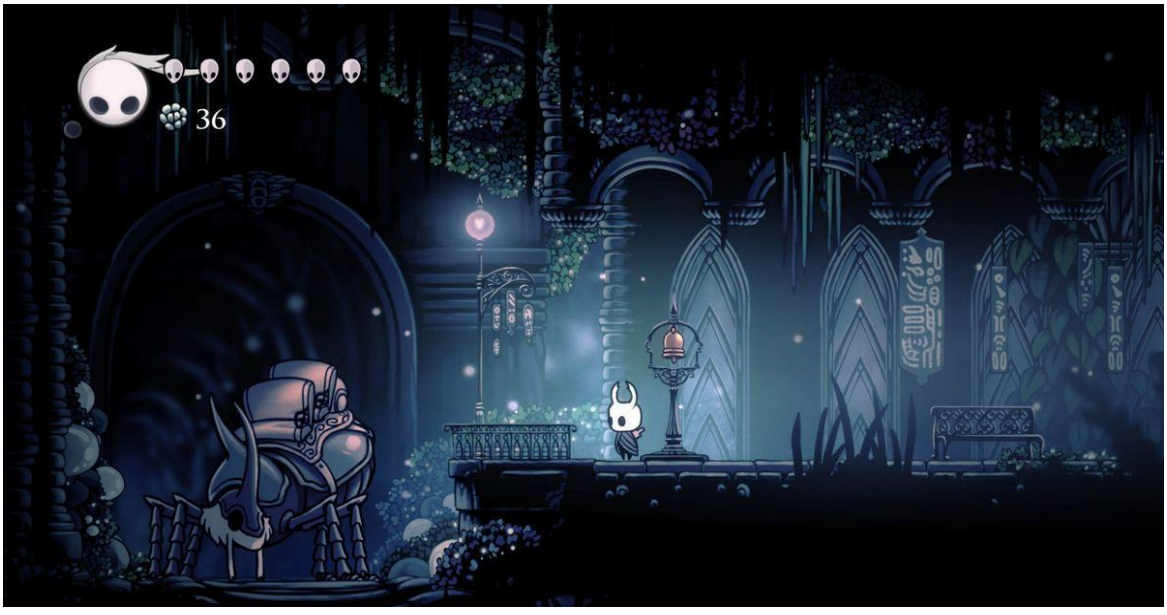

Рисунок 1.3 – Початкова локація у грі Hollow Knight

У грі ми керуємо маленьким безіменним жуком, якого усі називають просто «Лицар». Історія гри відбувається у давно покинутому королівстві комах Святогнізді (Hallownest). Гравець повинен досліджувати світ, зустрічати різноманітних персонажів та боротися з ворогами. Головна мета гри - це знайти та перемогти босів, які розташовані в різних частинах ігрового світу.

Гра має відкритий світ, де гравці можуть вільно переміщатися та досліджувати його. У світі гри є багато різних локацій, які мають свою унікальну атмосферу та графічний стиль. Локації пов'язані між собою системою підземних проходів та складних лабіринтів, що дозволяє гравцеві досліджувати світ наскрізь.

Hollow Knight має складну систему боїв та геймплейну механіку, яка вимагає від гравців високої точності та швидкості реакції. Гра містить багато різноманітних ворогів та босів, які мають свої унікальні характеристики та вимагають від гравців різних підходів до бою.

Hollow Knight також має захоплюючий саундтрек, який створює унікальну атмосферу гри та доповнює візуальну графіку. Гра є доступною на різних платформах, включаючи Nintendo Switch, PlayStation 4 та Xbox One.

До переваг даної гри можна віднести наступне:

– Геймплей: гра має глибокий та складний геймплей, який охоплює велику кількість різноманітних ворогів, головоломок та секретів. Ігровий світ є великим та складним, і дозволяє гравцям досліджувати та відкривати нові місця в своєму власному темпі.

– Графіка та атмосфера: гра має прекрасну, мистецьку графіку та незабутню атмосферу. Музика, звук та візуальний дизайн гри створюють неперевершену атмосферу, яка здатна затягнути гравців на години.

– Історія: гра має глибоку та захоплюючу історію, яка розкривається повільно через зустрічі з різними персонажами та дослідженням світу гри.

– До недоліків можна віднести наступне:

– Висока складність: гра має досить високий рівень складності, і може здатися деяким гравцям досить важкою і відлякати їх від проходження.

– Відсутність маркерів: гра не має достатньо маркерів або інших підказок для допомоги гравцям у навігації, що може збільшити час, необхідний для проходження та дослідження різних куточків ігрового світу.

– Довга тривалість: гра має досить довгу тривалість та складний сюжет, що може стати перешкодою для гравців, які не хочуть витрачати багато часу на проходження гри.

1.2.4 Starbound

Starbound – це комп'ютерна гра жанру пригодницької платформи [9], в якій гравець досліджує випадково згенеровані світи, знаходить різні ресурси та бореться з ворогами. Гра була розроблена студією Chucklefish і була випущена в 2016 році.

Рисунок 1.4 – Космічний корабель головного персонажа гри Starbound

У грі гравець грає за космічного мандрівника, який був вигнаний зі свого дому на кораблі. Гравець розпочинає свою подорож з одного з випадково генерованих світів. В подальшому йому потрібно буде досліджувати різні планети, знаходити корисні ресурси та боротися з ворогами.

У грі є безліч різних ресурсів, з яких можна створювати різні інструменти, зброю, броню, електронні пристрої та інші предмети. Крім того, гравець може будувати різні споруди та поселення, розвивати їх та

налагоджувати взаємодію з іншими персонажами, які можуть допомагати гравцеві в його подорожі.

У грі також є різні види ворогів, які можуть бути знайдені на різних планетах, а також багато різних босів, які з'являються в окремих місцях. Крім того, гравець може зустрічати різні раси, з якими може взаємодіяти, торгувати та навіть вступати в альянси.

Особливістю гри є величезна кількість різних елементів, які можна зібрати та поєднувати між собою, що дає гравцю безмежні можливості для творчого процесу. Гра має досить красиву та приємну піксельну графіку та добре пророблену атмосферу, яка допомагає гравцеві зануритися в віртуальний світ гри.

У грі Starbound є безліч різноманітних локацій, від сухих пустельних планет до вологих тропічних джунглів, заселених різноманітними істотами. Крім того, в грі є безліч різних секторів, які можна досліджувати, знаходити нові технології та ресурси, а також зустрічати нових персонажів та інші розваги.

Доступна можливість грати як в одиночному режимі, так і в мультиплеєрі з іншими гравцями. Гравці можуть відвідувати один одного на своїх планетах, торгувати та співпрацювати у різних справах.

В цілому, Starbound – це дуже цікава та розважальна гра з безліччю можливостей для дослідження віртуального світу та творчості.

До переваг даної гри можна віднести наступне:

– Безмежний світ: Starbound має величезний відкритий світ, що складається з безлічі планет, які можна досліджувати. Це дає гравцям можливість знайти безліч різноманітних ресурсів та предметів, а також насолоджуватися незабутнім геймплеєм.

– Будівництво: у грі Starbound є безліч матеріалів та ресурсів, які можна використовувати для будівництва різноманітних речей, від простих житлових приміщень до величезних космічних станцій.

– Різноманітність персонажі: гравці можуть вибирати з багатьох різних рас (фантастичних видів істот), кожен з яких має свої особливості та характеристики, що додає грі глибину та різноманітність.

– Графіка та музика: Starbound має дуже красиву графіку та незабутню музику, що робить гру ще більш насиченою та захоплюючою.

– До недоліків можна віднести наступне:

– Геймплей: Starbound може бути дещо складною грою для новачків, тому вам, можливо, знадобиться трохи часу, щоб зрозуміти, як все працює.

– Механіка бою: в бойовій системі гри Starbound є кілька недоліків, які можуть здатися гравцям незручними.

– Темп гри: хоча відкритий світ у грі дуже великий, дослідження можуть бути тривалим та повільним процесом, що може зменшувати бажання гравців грати в гру.

– Оптимізація: деякі гравці можуть зіткнутися з проблемами оптимізації у грі, особливо якщо вони грають на менш потужному комп'ютері.

1.2.5 Deadlight

Deadlight - це 2D платформер з елементами хоррору [10], що розроблений компанією Tequila Works та виданий Microsoft Studios у 2012 році для платформ Xbox 360, PC та PlayStation 4.

Гравець у грі виступає у ролі Рендолфа Уейна, який є одним із небагатьох вцілілих після зомбі-апокаліпсису. Він намагається знайти свою родину та вижити, пройшовши через різні перешкоди та ворогів.

Події відбуваються в Сіетлі 1986 року, де Рендолф вимушений пройти через забруднені вулиці, закинуті будівлі та небезпечні місця, щоб дістатися до родини.

Рисунок 1.5 – Знімок екрану з гри Deadlight під час проходження місії “Лігво”

Гравець повинен пройти через різні перешкоди, уникати ворогів та збирати різні ресурси, щоб продовжувати свою подорож.

Головна особливість гри - це елементи хоррору, які додають особливу атмосферу грі, а також елементи головоломок, що не дозволяють гравцю просто пройти рівень.

Гра має цікавий стиль малюнку, з використанням затемнених та зламаних тонів, що надає грі постапокаліптичний настрій. Геймплей гри також дуже добре продуманий, з різноманітними моментами дії та неочікуваними поворотами сюжету.

Загалом, Deadlight - це цікава гра для тих, хто любить платформери з хорошою сюжетною лінією та елементами хоррору.

Ця гра зможе зацікавити як досвідчених гравців, так і новачків, які шукають щось нове та цікаве.

До переваг даної гри можна віднести наступне:

– Атмосфера: гра має дуже насичену атмосферу та створює відчуття постійної загрози. Музика, звукові ефекти та зображення зомбі допомагають створити цей настрій.

– Графіка: у грі чудова графіка та дизайн рівнів. Вони розроблені дуже детально та мають велику кількість складних головоломок та пасток, що додає грі глибину та різноманітність.

– Історія: гра має цікаву та захоплюючу історію головного героя, який є харизматичним та цікавим персонажем.

– До недоліків можна віднести наступне:

– Тривалість гри: гра може здатися дуже короткою для деяких гравців. Хоча є багато складних рівнів, загальна тривалість гри менша, ніж у багатьох інших платформерів.

– Керування: процес керування персонажем може бути незручним на деяких рівнях, особливо на рівнях, які потребують точності та швидкості.

– Нестабільність: гра містить деякі помилки та баги, які можуть спричинити крах гри та втрату прогресу.

– Різноманітність: гра має обмежену кількість ворогів та видів зброї, що може зменшити різноманітність геймплею.

У проаналізованих прикладів переважає: простий стиль керування головними персонажами, піксельна графіка та яскравий дизайн. Це має бути врахованно при створені 2D гри платформера.

2 ПРОЕКТУВАННЯ 2D ГРИ ПЛАТФОРМЕРА

2.1 Опис сценарію 2D гри платформера «Пригоди Пса Патрона»

Головний герой гри ославлений Пес Патрон, який зображений на рисунку 2.1. Патрон є першою собакою в історії, що має титул «Пес доброї волі» від UNICEF. Джек-рассел-тер'єр став популярним у 2022-му як помічник саперів у лавах ДСНС. У травні минулого року Патрон отримав медаль «За віддану службу» від президента України. Загальна кількість його підписників у соцмережах сягає мільйона.

Рисунок 2.1 – Пес Патрон

Авторські права на зображення Пса Патрона належать компанії «Патрон не спить» [11], яка у серпні 2022-го. була створена Гендиректоркою Юлією Гребеньовською.

У даній грі не порушується авторське право, оскільки передбачається створення власних зображень , або використання готових матеріалів з вільного доступу.

За задумом гри головний герой пес Патрон відправляється на пошуки головного скарбу, підчас пошуку якого стикається з перешкодами: пасивними у вигляді різноманітних пасток та активними у вигляді ворожодіючих персонажів. В процесі гри передбачено нарахування балів за виконання поточних завдань.

За сценарієм існує можливість проходження 3 рівнів від простішого до складнішого. Скарб, що є цілю, знаходиться в кінцевій локації останнього рівня. Проте, передбачена система ушкоджень головного героя при невдалому виконані завдань, що накопичуються та можуть призвести до завершення гри, після чого гра може розпочинатися з початку поточного рівня.

Час виконання завдань не обмежується, що сприяє ненапруженому стилю гри.

2.2Функціонал головного меню

Основний алгоритм 2D гри платформера з головним меню та 3 рівнями можна описати наступним чином:

– Запуск програми:

– Вивід головного меню з варіантами: "Почати гру", "Вибрати рівень" та "Вийти".

– Очікування вибору користувача.

– Вибір "Почати гру":

– Запуск першого рівня гри.

– Запуск основного циклу гри, який оновлює стан гри та відображає його на екрані.

– Обробка вводу користувача для керування головним персонажем.

– Перевірка зіткнень головного персонажа з іншими об'єктами в грі (платформи, вороги, предмети тощо).

– Перевірка умов для завершення рівня (наприклад, досягнення виходу або вбивство всіх ворогів).

– Якщо рівень завершено, перехід до наступного рівня або повернення до головного меню.

– Вибір "Вибрати рівень":

– Вивід списку доступних рівнів, наприклад, "Рівень 1", "Рівень 2", "Рівень 3".

– Очікування вибору користувача.

– Вибір конкретного рівня:

– Запуск обраного рівня гри.

– Аналогічно пункту 2, запуск основного циклу гри та його обробка.

– Вибір "Вийти":

– Завершення програми.

Цей загальний алгоритм надає можливість користувачеві вибрати режим гри (почати гру або вибрати рівень) та виходити з програми. Рівні доступні як у головному меню, так і в режимі "Вибрати рівень". Крім того, після завершення кожного рівня є можливість переходу до наступного рівня або повернення до головного меню. На рисунку 2.2 зображений загальний алгоритм головного меню 2D гри платформера «Пригоди Пса Патрона».

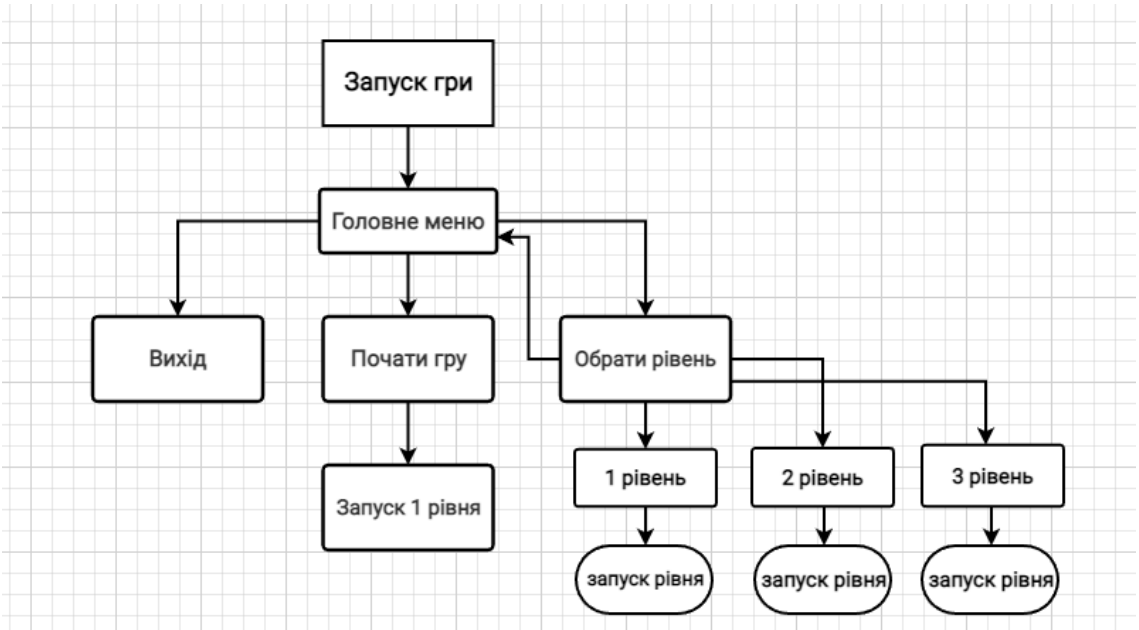

Рисунок 2.2 – Загальний алгоритм роботи меню 2D гри платформера «Пригоди Пса Патрона»

2.3 Проектування інтерфейсу користувача

Проектування інтерфейсу користувача (UI) для 2D гри платформера включає створення елементів, які взаємодіють з гравцем і полегшують його навігацію та взаємодію з грою.

На рисунку 2.3 зображений спроектований макет вікна головного меню 2D гри платформера «Пригоди Пса Патрона».

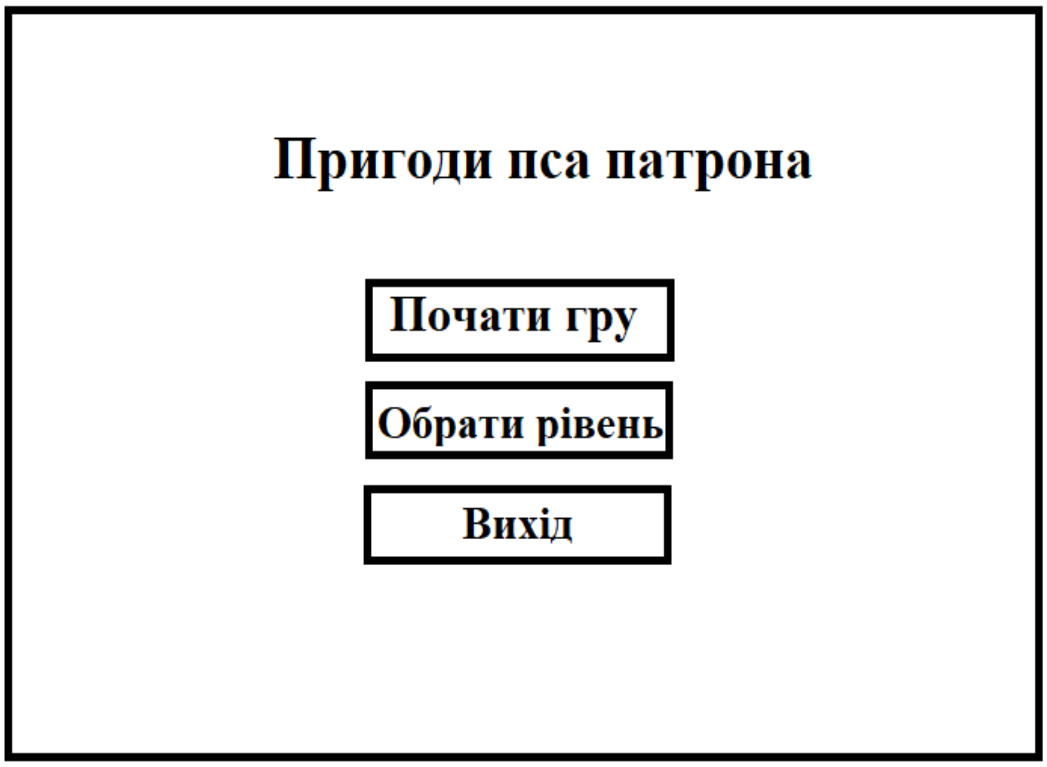

Рисунок 2.3 – Макет головного меню 2D гри платформера «Пригоди пса Патрона»

Після втрати усіх життів, буде з'являтися екран, який буде сповіщати про кінець гри. Нижче на рисунку 2.4 зображено його макет.

На ньому можна буде обрати три варіанти подальших дій: Перезапуск, вихід до головного меню або вихід з гри.

Рисунок 2.4 – Макет меню вікна “Кінець гри” 2D гри платформера «Пригоди пса Патрона»

Після вибору кнопки «Перезапуск» відбувається запуск поточного рівня з початку.

Вибір кнопки «Меню» перенаправляє користувача на вікно головного меню.

Кнопка «Вихід» дає можливість завершити гру та здійснити вихід з програми.

2.4 Алгоритм дій користувача 2D гри платформера «Пригоди Пса Патрона»

При створені гри, 2D гри платформера «Пригоди Пса Патрона» будуть взяті до уваги наступні умови:

Запуск рівня:

– Завантаження рівня зі збереженої структури (наприклад, з файлу або бази даних).

– Встановлення початкової позиції головного персонажа.

– Ініціалізація рахунку балів, кількості життів та інших необхідних змінних.

Геймплей:

– Обробка вводу користувача для керування головним персонажем (рух вліво/вправо, стрибок).

– Перевірка колізій та взаємодії персонажа з об'єктами на рівні (платформи, вишеньки, вороги).

Якщо персонаж зіштовхується з ворогом:

– Кількість життів зменшується на 1.

– Перевірка, скільки залишилося життів.

Якщо немає життів:

– Зупинка гри.

– Виведення повідомлення про поразку.

– Перехід до пункту 5.

Якщо є ще життя:

– Перенесення персонажа на початкову позицію рівня.

Якщо персонаж стрибає на ворога:

– Знищення ворога.

– Додавання балів до рахунку.

Перевірка, чи персонаж досяг кінця рівня:

Якщо так:

– Перехід на наступний рівень.

– Виведення повідомлення про перемогу.

– Перехід до пункту 5.

Якщо ні:

– Продовження гри.

Взаємодія з вишеньками:

– Перевірка колізій персонажа з вишеньками.

– Якщо персонаж підбирає вишеньку:

– Знищення вишеньки.

– Додавання балів до рахунку.

Завершення рівня:

– Зупинка гри.

– Збереження рахунку та інших потрібних даних (наприклад, прогресу рівнів, отриманих балів) для наступних рівнів або локального збереження.

– Перехід на наступний рівень

– Відображення опційного меню з вибором наступного кроку: "Продовжити" або "Почати з початку".

Продовження або початок з початку:

– Залежно від вибору користувача:

– Продовження гри з наступного рівня або з початку поточного рівня (відповідно до збережених даних).

– Повторний запуск гри з першого рівня.

– Повернення до головного меню або вихід з гри.

Нижче на рисунку 2.5 зображено UML діаграму варіантів використання, яка демонструє ігрові можливості головного героя даної гри.

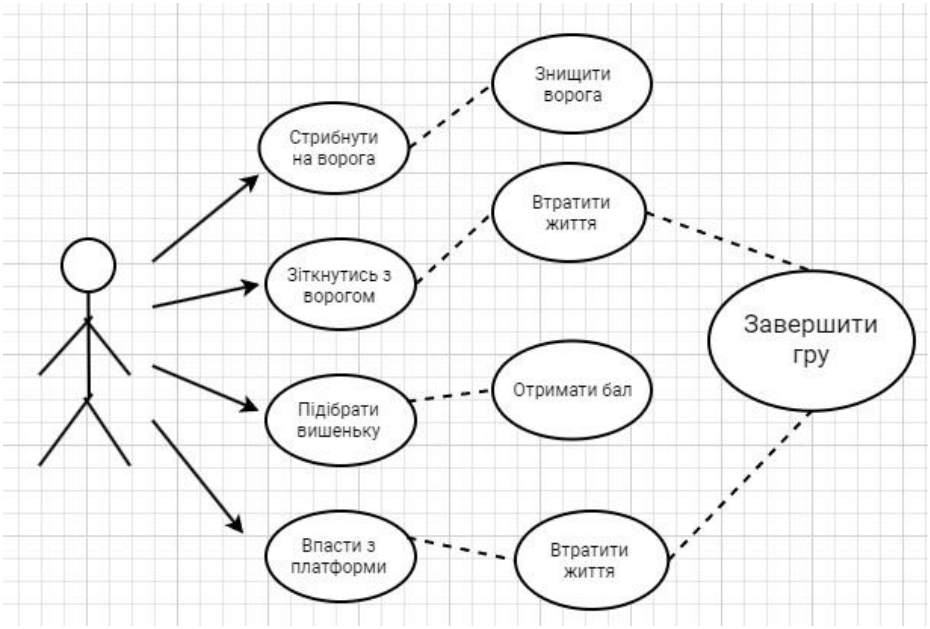

Рисунок 2.5 – UML діаграму варіантів використання головного героя 2D гри «Пригоди Пса Патрона»

За допомогою вищенаведеної UML діаграми змодельовані основні процеси гри.

2.5 Аналіз середовищ розробки для створення 2D гри платформера

Ігровий рушій (game engine) - це програмне забезпечення, яке використовується для створення відеоігор.

Він надає розробникам ігор набір інструментів та бібліотек для створення графіки, фізики, звуку, штучного інтелекту та інших елементів, що складають відеоігру.

Ігровий рушій може бути використаний для створення різних жанрів ігор, від екшн до стратегій та рольових ігор.

Для аналізу було обрано декілька популярних ігрових рушіїв – Unity 2022, Unreal Engine, та Godot Engine.

2.5.1 Unity 2022

Unity Engine - це один з найбільш популярних ігрових рушіїв у світі, що використовується для створення відеоігор для різних платформ, таких як ПК, консолі, мобільні пристрої та віртуальна реальність.

Unity був розроблений компанією Unity Technologies, заснованою в Данії у 2004 році. Спочатку цей ігровий рушій був розроблений як інструмент для створення ігор на ПК, проте згодом була додана підтримка для інших платформ.

На даний момент Unity 2022 підтримує розробку ігор для наступних платформ:

- ПК (Windows, MacOS, Linux)
- Консолі (PlayStation, Xbox, Nintendo Switch)
- Мобільні пристрої (iOS, Android)

– Віртуальна та доповнена реальність (Oculus, HTC Vive, Microsoft HoloLens, Google Cardboard та інші платформи віртуальної та доповненої реальності)

Завдяки своїй мультиплатформовості, є одним з найбільш популярних ігрових рушіїв у світі, що дозволяє розробникам створювати ігри для широкої аудиторії та різних платформ.

– Для роботи з Unity 2022 на ПК необхідні наступні системні вимоги:
– Операційна система: Windows 7 SP1+, 8, 10, або macOS 10.15+
– Процесор: Intel Core i3 2,5 ГГц або еквівалентний
– Оперативна пам'ять: 4 ГБ
– Відеокарта: сумісна з DirectX 11 або OpenGL 4.4
– Місце на жорсткому диску: 30 ГБ вільного місця
– Unity має досить простий та зрозумілий інтерфейс. (рисунок 2.6)

Рисунок 2.6 – Інтерфейс Unity 2022

Також за допомогою Unity можна розробляти різноманітні інтерактивні додатки та програми, які не є іграми. Ось декілька прикладів того, що можна зробити за допомогою Unity окрім ігор:

– Віртуальні тури - Unity можна використовувати для створення віртуальних турів по музеях, виставкових залах, архітектурних спорудах та інших місцях.

– Тренажери та симулятори - Unity можна використовувати для створення тренажерів та симуляторів, наприклад, для навчання пілотів, медичних працівників, водіїв та інших професій.

– Візуалізація даних - Unity можна використовувати для створення інтерактивних візуалізацій даних, таких як графіки, діаграми та інші візуалізації.

– Інтерактивні книги та комікси - Unity можна використовувати для створення інтерактивних книг та коміксів з анімаційними ефектами та інтерактивними елементами.

– Відео та аудіо плеєри - Unity можна використовувати для створення відео та аудіо плеєрів з різноманітними функціями та можливостями.

– Додатки для віртуальної та доповненої реальності - Unity можна використовувати для створення додатків для віртуальної та доповненої реальності, таких як віртуальні майданчики для спортивних заходів, навчальні програми та інші.

– Загалом, за допомогою Unity можна створювати різноманітні інтерактивні додатки, які не обов'язково пов'язані з ігровою індустрією.

Також у Unity є свій цифровий магазин Asset Store, у якому знаходиться велика кількість матеріалів для розробки: 2D та 3D моделі, спрайти, фонові рисунки, скрипти, різні доповнення та розширення до клієнту Unity, тощо.

Даний рушій надає різні тарифні плани для різних типів користувачів та компаній залежно від їх потреб. Ось детальна інформація про доступні тарифні плани:

– Unity Personal (безкоштовна) - безкоштовна версія, призначена для некомерційного використання. Вона містить базові функції та інструменти для розробки 2D та 3D ігор, та може бути використана для розробки ігор на платформи Windows, Mac та Linux.

– Unity Plus ($40 на місяць або $399 на рік) - цей план призначений для малих та середніх комерційних проектів з обсягом до $200,000 річного доходу. Він містить усі функції та інструменти, які є у Unity Personal, а також додаткові функції, такі як Unity Teams Basic, підтримка для мобільних платформ Android та iOS, а також доступ до додаткових ресурсів та підтримки від команди Unity.

– Unity Pro ($150 на місяць або $1500 на рік) - цей план призначений для серйозних комерційних проектів та має значно більше можливостей, ніж попередні два. Він містить усі функції та інструменти, які є у Unity Plus, а також доступ до додаткових функцій, таких як Unity Teams Advanced, підтримка для великих проектів та додаткові можливості для розробки додатків віртуальної та доповненої реальності. Цей план має місячну підписку.

Також є Unity Enterprise та Unity Education. Перший призначений для великих компаній з обсягом доходу понад $200,000 на рік, та має індивідуальну ціну залежно від потреб, а наступний Unity Education - це спеціальний безкоштовний план для навчальних закладів, який дозволяє використовувати движок для навчання студентів.

Програмний код у Unity реалізується за допомогою Microsoft Visual Studio, яка має можливість інтеграції додаткового функціоналу для Unity. (рисунок 2.7)

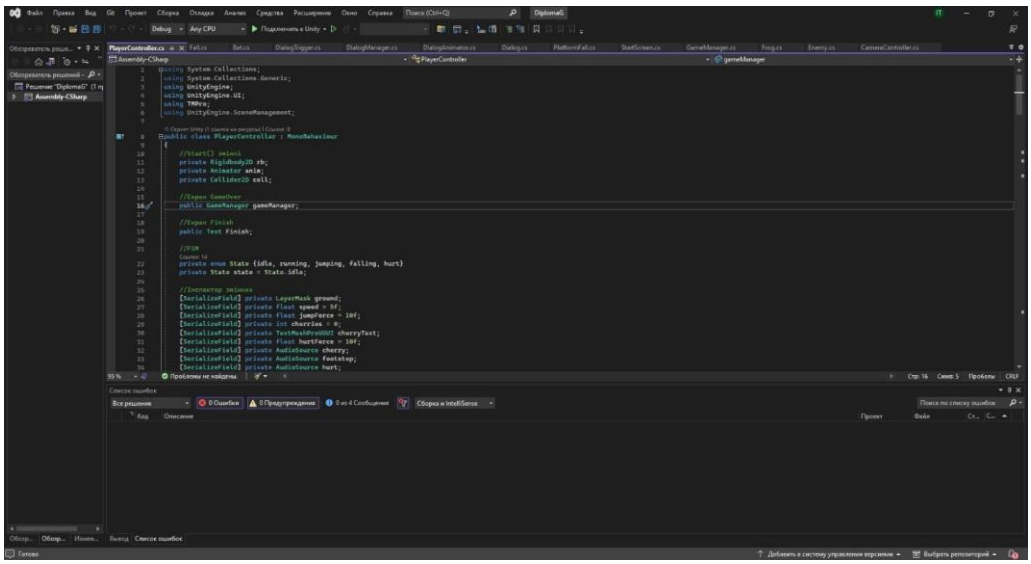

Рисунок 2.7 – Інтерфейс Microsoft Visual Studio 2022

Microsoft Visual Studio - це інтегроване середовище розробки (IDE) від компанії Microsoft, призначене для створення програмного забезпечення на різних платформах, включаючи Windows, Linux, iOS, Android та інші. Visual Studio містить різноманітні інструменти для розробки, відладки та тестування програмного забезпечення, включаючи редактор коду, інструменти відлагодження, підтримку систем контролю версій, аналізатори коду та інші. Підтримує різноманітні мови програмування, такі як C++, C#, Visual Basic, JavaScript та інші, та має можливості для розробки різних типів додатків, від настільних програм до веб-сайтів та мобільних додатків. Він також підтримує різні платформи для розробки, включаючи .NET, .NET Core та Xamarin.

Microsoft Visual Studio є одним з найпопулярніших інструментів розробки програмного забезпечення та використовується розробниками по всьому світу для створення різноманітних додатків та програм.

2.5.2 Unreal Engine 5

Unreal Engine 5 - це остання версія ігрового движка, розроблена компанією Epic Games. Перша версія Unreal Engine була випущена у 1998 році для гри Unreal, яка стала досить популярною в індустрії відеоігор (рисунок 2.8). В наступні роки Unreal Engine був використаний для створення таких відомих ігор, як Gears of War, BioShock, Mass Effect, Batman: Arkham Asylum та інших.

Рисунок 2.8 – Знімок екрану з гри Unreal

Unreal Engine підтримує широкий спектр платформ - Windows, MacOS та Linux, Консолі нового покоління, такі як PlayStation 5 та Xbox Series X/S, Консолі попереднього покоління, такі як PlayStation 4 та Xbox One, Мобільні пристрої, такі як iPhone та Android-пристрої, Віртуальна реальність (VR) та доповнена реальність (AR), Oculus та HTC Vive. Більше того, Unreal Engine 5 підтримує ряд інших платформ та сервісів, таких як Nintendo Switch, Google Stadia, Amazon Web Services, Microsoft Azure та інші. Це дозволяє розробникам створювати ігри для широкого спектру платформ та сервісів з використанням одного і того ж ігрового движка.

Також для комфортної роботи з редактором Unreal Engine персональний комп’ютер має задовольняти мінімальні системні вимоги:

Мінімальні вимоги:

– Операційна система: Windows 10 64-бітна версія
– Процесор: Quad-core Intel or AMD, 2.5 GHz or faster
– Оперативна пам'ять: 8 GB RAM
– Відеокарта: DirectX 11 or DirectX 12 compatible graphics card
– Місце на диску: 200 GB free disk space
– Роздільна здатність екрану: 1280 x 720

У движка зручний та простий інтерфейс, що вдосконалюється з кожною новою версією. (рисунок 2.9)

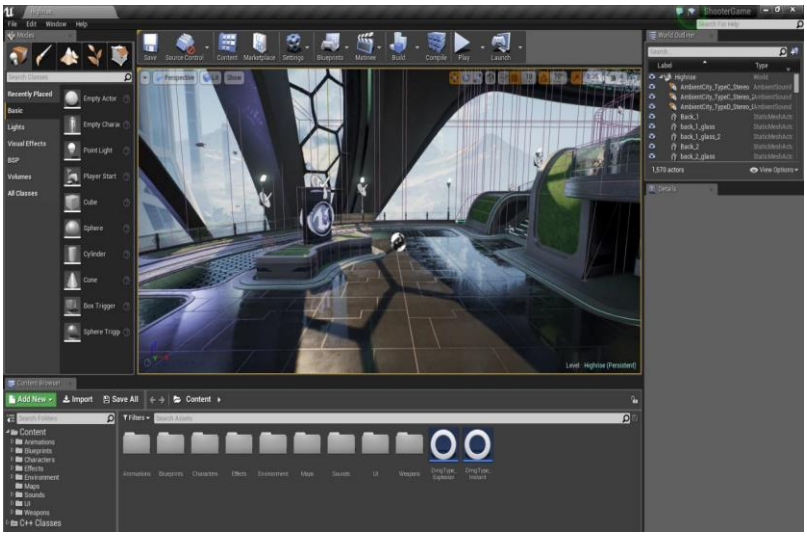

Рисунок 2.9 – Інтерфейс Unreal Engine 5

Unreal Engine 5 був офіційно анонсований у травні 2020 року. Одним із головних плюсів Unreal Engine 5 є підтримка нової технології Nanite, яка дозволяє реалістично відображати велику кількість полігонів та текстур без значного зниження продуктивності. Крім того, Unreal Engine 5 включає в себе ряд інших нововведень, таких як покращена система освітлення Lumen, покращені інструменти анімації та фізики, покращені інструменти редагування країв, нові можливості для роботи з зовнішніми сервісами та інші.

Unreal Engine 5 підтримує кілька мов програмування. Основними мовами програмування, які використовуються в Unreal Engine 5, є:

– C++ - це основна мова програмування в Unreal Engine. Вона використовується для розробки ядра гри, фізики, штучного інтелекту та багатьох інших аспектів.

– Blueprint - це візуальна мова програмування, що дозволяє створювати скрипти без написання коду. Blueprint дуже популярна серед розробників ігор, оскільки дозволяє швидко створювати прототипи ігор.

Крім того, Unreal Engine 5 також підтримує інші мови програмування, такі як Python та Java, але вони зазвичай використовуються для додаткових функцій, таких як автоматизація процесу розробки або з'єднання з іншими сервісами.

Щодо вартості використання Unreal Engine 5, то движок є безкоштовним для використання та розробки ігор. Однак, Epic Games вимагає 5% від прибутку від продажу гри, яка була створена з використанням Unreal Engine 5. Це означає, що ви можете використовувати Unreal Engine 5 для розробки своєї гри, але якщо ви заробите гроші на продажі гри, ви повинні буде сплатити Epic Games 5% від цього прибутку.

Універсальність та потужність Unreal Engine 5 робить його одним з найбільш популярних ігрових движків на ринку, що дозволяє розробникам створювати якісні ігри для різних платформ, включаючи ПК, консолі та мобільні пристрої. Звичайно Unreal Engine 5 більш підходить до глобальних

розробок ігор у стилі 3D, ніж 2D, але все ж таки має широкий вибір платформ для готових проектів та розробки нових.

2.5.3 Godot Engine

Godot Engine - це безкоштовний, відкритий ігровий движок з візуальним редактором, який дозволяє розробляти 2D та 3D ігри для різних платформ, таких як Windows, Linux, macOS, Android, iOS, HTML5, PlayStation, Xbox та Nintendo Switch.

Ось системні вимоги для запуску Godot Engine на комп'ютері:

Мінімальні вимоги:

– Операційна система: Windows 7 або новішої / Linux-дистрибутив з ядром версії 3.14 або новішої / macOS 10.12 Sierra або новішої

– Процесор: з тактовою частотою 1 ГГц

– Оперативна пам'ять: 512 МБ

– Відеокарта: Графічна карта з підтримкою OpenGL ES 3.0

– Роздільна здатність екрану: 1024 x 768

– Рекомендовані вимоги:

– Операційна система: Windows 10 або новішої / Linux-дистрибутив з ядром версії 3.14 або новішої / macOS 10.15 Catalina або новішої

– Процесор: з тактовою частотою 3 ГГц або вище

– Оперативна пам'ять: 4 ГБ або більше

– Відеокарта: Графічна карта з підтримкою OpenGL ES 3.0 або OpenGL 3.3 або новішої

– Роздільна здатність екрану: 1920 x 1080

Важливо також звернути увагу, що конкретні вимоги можуть залежати від розміру та складності проекту, який ви хочете розробити в Godot Engine (рисунок 2.10).

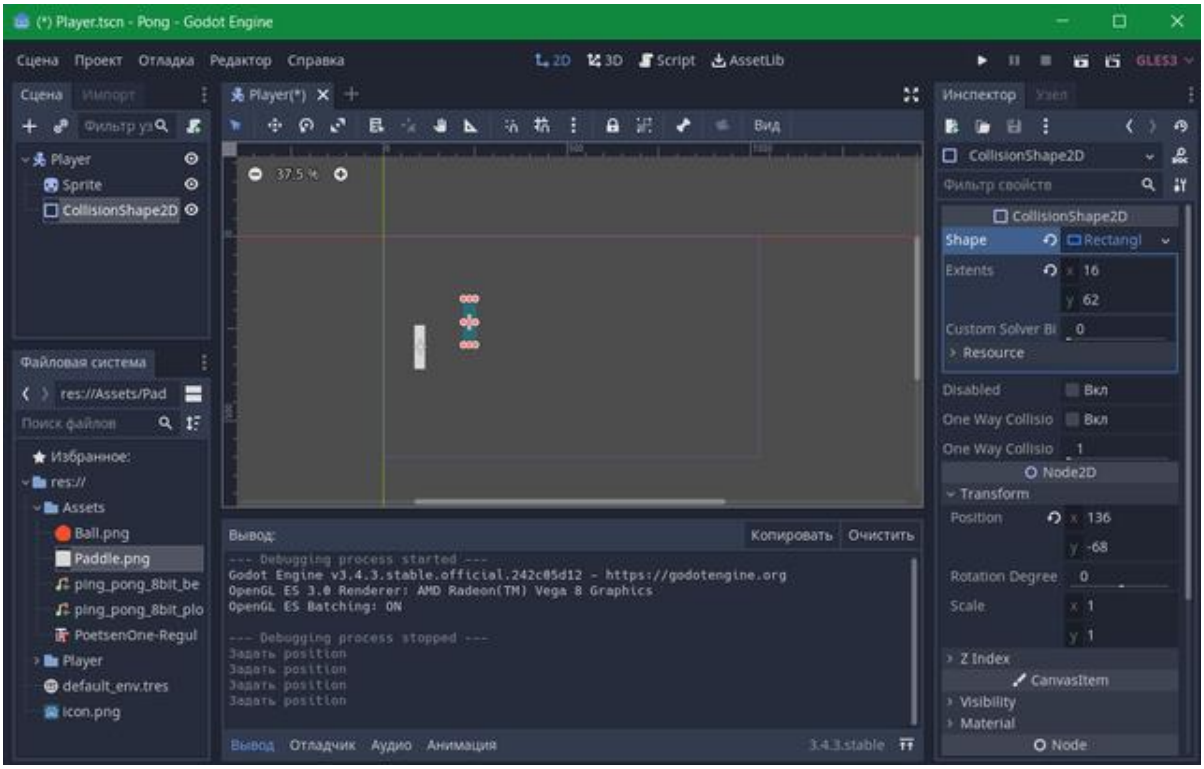

Рисунок 2.10 – Інтерфейс Godot Engine

Основна мета Godot Engine - зробити розробку ігор доступною для всіх, незалежно від рівня їх знань і досвіду. Це досягається завдяки простому інтерфейсу, візуальному редактору, повноцінній підтримці скріптів (мовами програмування GDScript, C #, C ++ та VisualScript) та великій базі знань, яка є безкоштовною і доступною для всіх користувачів.

Незважаючи на те, що Godot Engine з'явився на ринку досить недавно, він вже набув значної популярності серед розробників. Це пояснюється тим, що движок має безліч переваг, таких як:

– Відкритий код та безкоштовність

– Підтримка розробки ігор на різних платформах

– Широкий вибір мов програмування

– Візуальний редактор для розробки ігор без кодування

– Швидкість та ефективність роботи завдяки низькому рівню абстракції та оптимізованому движку рендерингу

– Велика кількість готових модулів та плагінів для розширення можливостей движка

– Активна спільнота, яка постійно розвиває движок та допомагає його користувачам.

Godot Engine – це чудовий вибір для початківців, а також для досвідчених розробників, які шукають потужний та ефективний інструмент для розробки ігор.

2.6 Формалізація вимог до створення 2D гри платформера «Пригоди Пса Патрона».

Відповідно до вище поданого сценарію 2D гра «Пригоди Пса Патрона» оптимально реалізувати як 2D гри платформер у піксельному стилі, який складається із трьох рівнів. Кожен рівень має свій власний дизайн, що ускладнює проходження гри, коли гравець переходить на наступний рівень.

У грі «Пригоди Пса Патрона» користувач керує персонажем під назвою Пес Патрон. Потрібно пройти декілька рівнів уникаючи падінь та ворогів, які будуть зустрічатися підчас гри У персонажа є певна кількість життів, в результаті втрати яких герой розпочинає гру з початку поточного рівня. Керувати героєм можна за допомогою клавіш напрямку (вгору, вниз, вліво та вправо), а також клавішею пробіл для стрибків.

Перший рівень гри має найлегшу складністю, адже має досить неважку карту в порівнянні з наступними рівнями. Із зростанням рівнів, збільшується і сама складність процесу гри. Другий рівень є помірно складним, в той момент як третій вважається найскладнішим рівнем.

Інтерфейс користувача повинен бути простим та інтуїтивно зрозумілим для гравця.

3 РЕАЛІЗАЦІЯ 2D ГРИ ПЛАТФОРМЕРА «ПРИГОДИ ПСА ПАТРОНА»

3.1 Інструментарій для створення 2D гри платформера «Пригоди Пса Патрона»

Для гри потрібно створити три різні ігрові сцени. Кожна сцена має ігрового персонажа, елементи декору у вигляді фону місцевості, ворогів та можливості зупинити гру, тобто поставити рівень на паузу, або вийти в головне меню. Управління персонажем здійснюється за допомогою клавіатури. Ми будемо застосовувати функцію отримання стану клавіш та користуватися константами для потрібних клавіш [3].

Для створення обрано ігровий рушій: Unity Engine тому, що він повністю задовольняє задані вимоги.Рушій зображено на рисунку 3.1

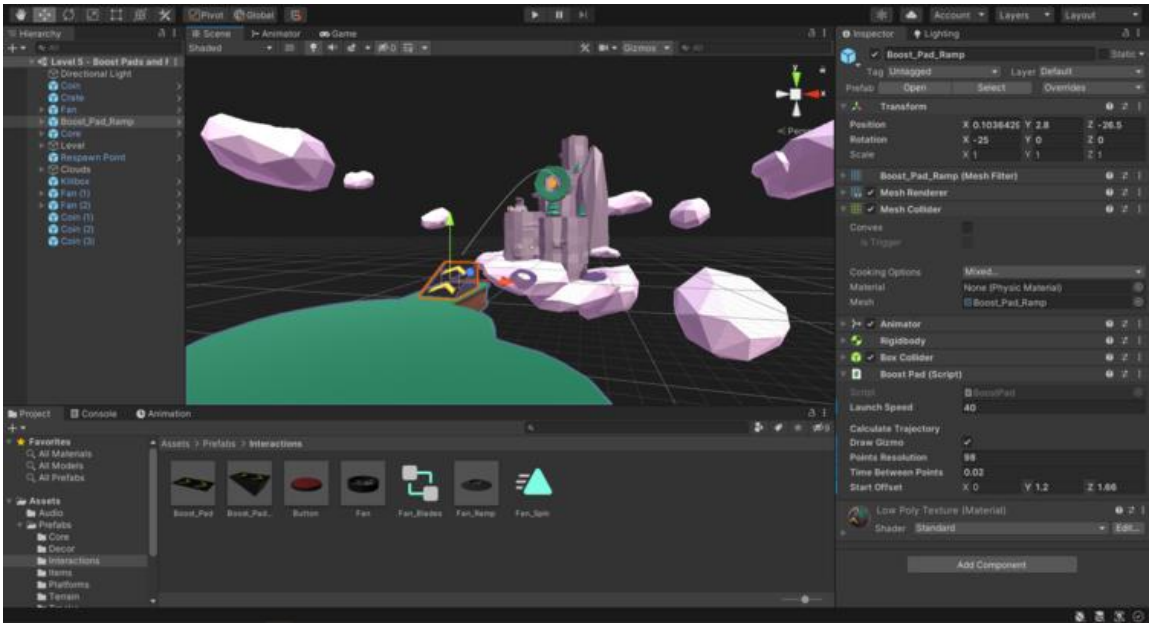

Рисунок 3.1 – Інтерфейс робочого вікна рушія Unity

Unity - це інтегроване середовище розробки (IDE), що використовується для розробки ігор та інтерактивних додатків. Вона надає засоби для створення 2D та 3D графіки, фізики, анімації, штучного інтелекту, мережевої гри, звуку та інших функцій.

Unity Engine має ряд інструментів, що дозволяють розробникам створювати різноманітні ігрові об'єкти та рівні, налаштовувати їх поведінку та

взаємодію між ними. Двигун також має вбудовану систему фізики, яка дозволяє розробникам створювати реалістичні фізичні ефекти, такі як зіткнення та руйнування.

Однією з головних переваг рушія є те, що вона підтримує різноманітні платформи, такі як Windows, Mac, Linux, Android, iOS, Xbox, PlayStation та інші. Це дозволяє розробникам створювати ігри та додатки для різних пристроїв та платформ.

Крім того, програма має велику спільноту розробників та різноманітні ресурси, які допомагають новачкам швидко засвоїти цю програму та розпочати розробку своїх ігор та додатків.

Для створення гри використовується мова програмування C#, яка є об'єктно-орієнтованою та має великий спектр можливостей. Основною перевагою є простота мови, та використання в поєднанні з доступністю і великим спектром різноманітних можливостей.

У загальному, Unity є потужним інструментом для розробки ігор та інтерактивних додатків, який надає розробникам засоби для створення реалістичних та захоплюючих проектів для різних платформ та пристроїв.

3.2 Розробка 2D гри платформера «Пригоди Пса Патрона» за допомогою рушія Unity

На початку необхідно створити новий проект платформера в Unity з використанням 2D шаблону, для цього слід виконати наступні кроки:

– У головному вікні Unity обираю опцію "New".

– Відкриється вікно "New Unity Project".

– В "New Unity Project" обираю вкладку "2D".

– У списку 2D шаблонів необхідно обрати шаблон, який найбільше відповідає потребам платформера.

– Зазвичай серед шаблонів є "2D Platformer", "2D Side-Scrolling" або схожі варіанти. У моєму випадку я обираю "2D Platformer".

– Обираю папку, де буде зберігатися проект.

– Вводжу назву проекту.

– Натискаю кнопку "Create" або "ОК", щоб створити новий проект.

– Unity створить папку проекту та запустить його.

Створення нового проеку платформера в Unity з використанням 2D шаблону подано на рисунку 3.2.

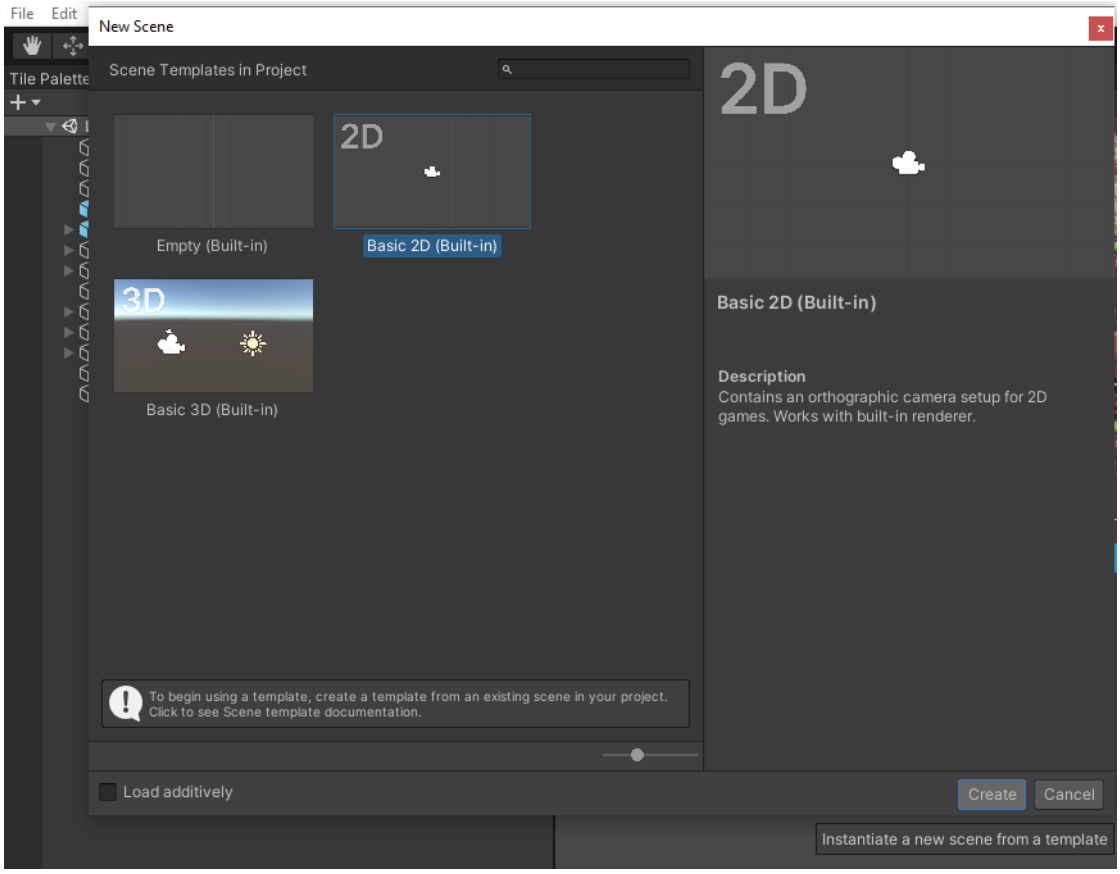

Рисунок 3.2 – Створення нового проєкту в Unity

Після створення проекту відбувається перенаправлення до головного робочого простору Unity, де є можливість розробляти гру 2D платформер, додавати об'єкти, рівні, налаштовувати логіку гри та багато іншого.

Після вибору відповідного 2D шаблону проєкту, отримується початковий набір компонентів, налаштувань та ресурсів, що спрощують процес створення платформера. Зазвичай ці шаблони містять основну фізичну модель, обробку введення, обробку колізій, рух персонажа та інші компоненти, які можна використовувати для створення гри. Наступним кроком є додавання власних графічних та аудіо ресурсів, щоб створити унікальний платформер.

3.2.1 Розробка рівня

Створення рівня для 2D гри платформера в Unity та розміщення платформ та об'єктів на сцені виконується наступним чином:

В Unity потрібно відкрити вкладку "Scene" та натиснути кнопку "New".

З'явиться нова порожня сцена (рисунок 3.3), де можна створювати свій рівень гри.

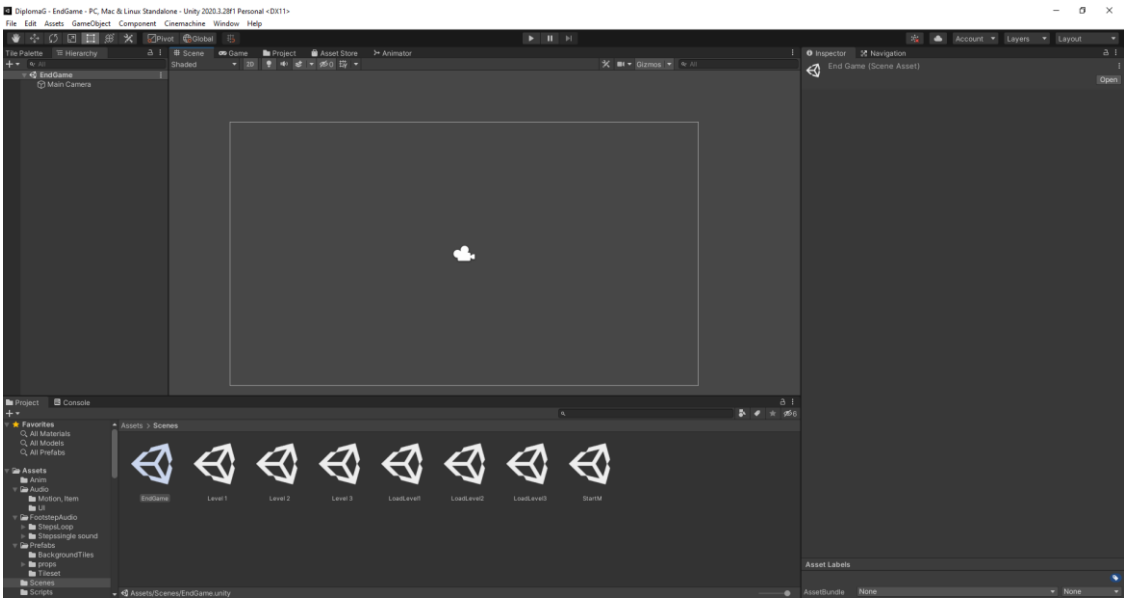

Рисунок 3.3 – Нова сцена Unity

Наступним кроком буде розміщення платформ та об'єктів. На панелі "Hierarchy" обирається об'єкт, який буде виступати як платформа (наприклад, порожній об'єкт або спрайт платформи).Потрібно перетягнути його на сцену в редакторі Unity, після чого об'єкт з'являється на сцені.

Цей процес виконується для всіх платформ та об'єктів, які потрібно розмістити на рівні.

3.2.2 Робота з піксельною графікою для реалізації 2D гри платформера «Пригоди Пса Патрона»

Під час створення гри у програмі Unity, робота з графікою включає розробку та використання таких елементів, як тайли (Tiles), спрайти (Sprites), текстури (Textures) та інші. Ось що ці елементи означають:

Тайл (Tile) є невеликим графічним елементом, який представляє собою частину фону, платформи або об'єкта на рівні гри.

Тайли зазвичай мають квадратну форму та можуть бути розміщені поруч або один на одного, утворюючи поверхню або об'єкти на сцені.В Unity тайли часто використовуються для створення поверхонь рівня, таких як підлоги, стіни або інші статичні об'єкти.

Спрайт (Sprite) є графічним зображенням або анімованим зображенням, яке може представляти персонажів, об'єкти або інші елементи гри.

В Unity спрайти використовуються для візуального відображення об'єктів на сцені.Спрайти можуть мати різні форми, розміри та анімацію. Вони можуть бути статичними або рухатися, змінювати своє положення, форму або інші властивості.

Текстура (Texture) є графічним зображенням, яке застосовується на поверхні об'єкта, щоб надати йому вигляд і колір.

В Unity текстури використовуються для прикраси, фарбування та візуалізації об'єктів.Текстури можуть бути створені з графічних файлів, таких як PNG або JPEG, або згенеровані в програмах для створення графіки.

Під час роботи з графікою в Unity створюється та редагується тайли, спрайти та текстури за допомогою різноманітних інструментів. Unity надає можливості для імпорту графічних ресурсів, редагування їх власною вбудованою графічною програмою або використання зовнішніх редакторів графіки, таких як Photoshop, Aseprite або GIMP, імпортування їх у проект та використання у грі.

Робота з графікою включає також анімацію спрайтів, налаштування масштабування, повороту, зміни кольорів та інших властивостей графічних елементів для досягнення бажаного візуального ефекту у грі.

Загалом, робота з графікою в Unity надає можливість створити візуально привабливий та цікавий світ гри, використовуючи тайли, спрайти та текстури, а також редагуючи їх властивості та анімацію, щоб додати реалізму та виразності 2D грі.

3.2.3 Створення спрайту головного героя 2D гри платформера «Пригоди Пса Патрона»

Щоб створити спрайт персонажа, буде використовуватись програма Aseprite (рисунок 3.4).

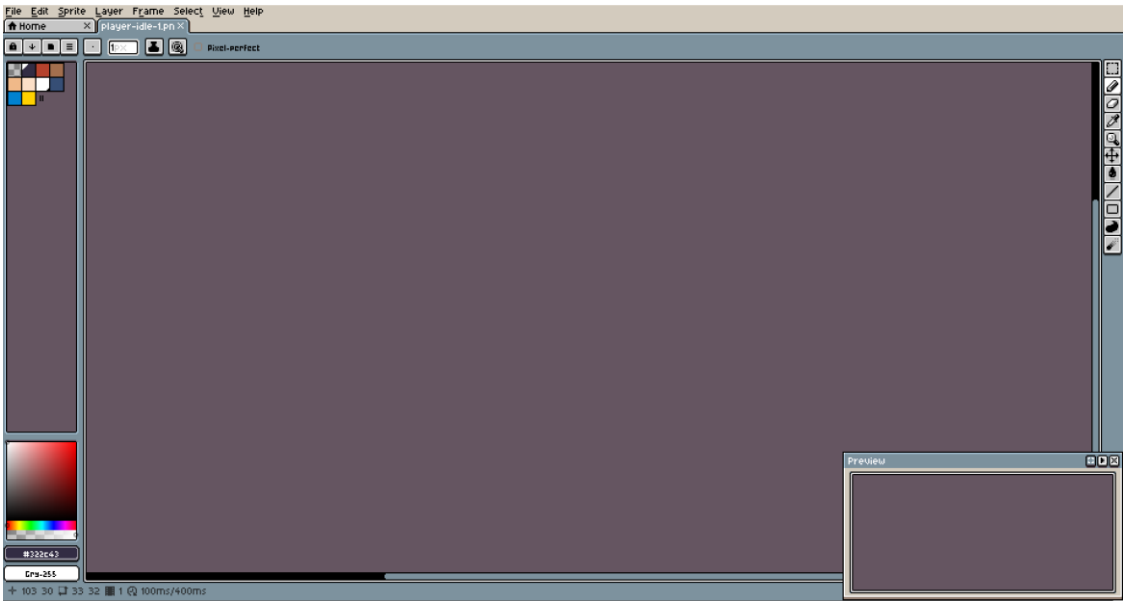

Рисунок 3.4 - Інтерфейс програми Aseprite

Aseprite – це популярний редактор піксельної графіки, який широко використовується для створення спрайтів для ігор, включаючи Unity. Ось кроки для створення спрайту в програмі Aseprite:

Після відкриття редактора Aseprite та створення нового файлу проєкту потрібно виконати наступну послідовність дій:

– Клацнути "File" у верхньому меню та вибрати "New" або використати комбінацію клавіш Ctrl+N.

– Задати розміри файлу, що відповідають розмірам спрайту в Unity.

– Обрати кількість кадрів, для створеня анімації для спрайту.

– Здійснення редагування спрайту відбувається за допомогою використання наступних інструментів:

– інструменти Aseprite для малювання або редагування спрайту;

– пензлики, кольори та інші інструменти, щоб створити бажану графіку спрайту;

– шари для розділення різних елементів спрайту або для створення окремих кадрів анімації.

– Коли спрайт буде готовий можна переходити до його збереження, та імпорту. Збереження спрайту відбувається за допомогою наступних дій:

– Клацнути "File" у верхньому меню та обрати "Save As" або використати комбінацію клавіш Ctrl+Shift+S.

– Обрати формат файлу для збереження спрайту (рекомендований формат PNG або GIF для піксельної графіки).

– Вказати місце збереження файлу та задати йому назву.

– Імпорт спрайту в Unity відбувається таким чином:

– В Unity відкрити проект і перейти до вкладки "Project".

– Розгорнути папку, в яку потрібно імпортувати спрайт.

– Перетягти файл спрайту з вікна файлового менеджера до вікна проекту Unity.

– Відкрити вкладку "Inspector" для вибраного спрайту та налаштувати параметри імпорту (наприклад, розмір, фільтрація, формат).

Тепер можна використовувати імпортований спрайт в Unity, розміщати його на сцені, створювати анімації, прив'язувати до об'єктів і використовувати для створення візуальних ефектів у вашій грі. Вигляд готового спрайту в Unity зображено на рисунку 3.5.

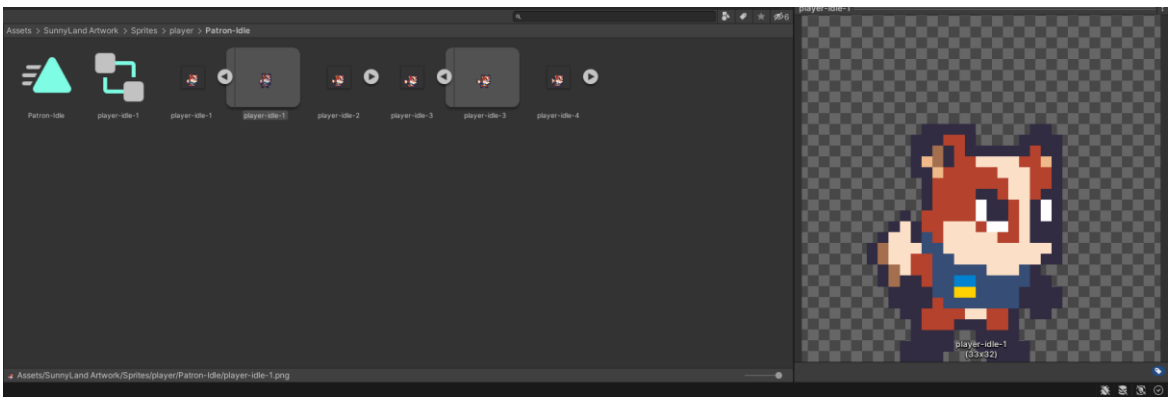

Рисунок 3.5 – Готовий спрайт головного персонажа в Unity

3.2.4 Імпорт та налаштування спрайтів ігрових текстур

Для роботи з спрайтами ігрових текстур використовується інструмент Unity під назвою Tile Palette. Tile Palette - це інструмент в Unity, який дозволяє легко створювати та редагувати тайли для використання на різних рівнях гри. Також, в Tile Palette можна імпортувати готові спрайти з бібліотеки Unity (рис. 3.6).

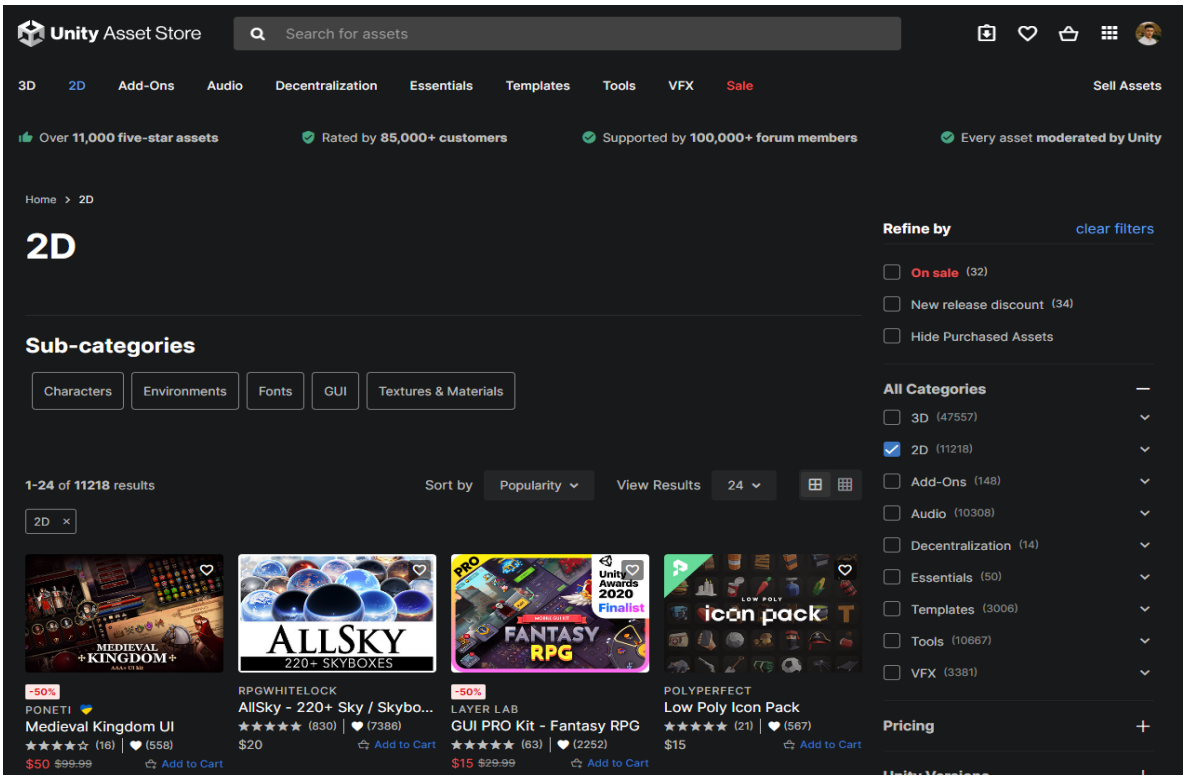

Рисунок 3.6 – Магазин матеріалів в Unity

Нижче описані кроки для роботи з Tile Palette та імпорту спрайтів з бібліотеки Unity:

– Відкрити вкладку "Window" у верхньому меню Unity та обрати "2D" - "Tile Palette". Це відкриє панель Tile Palette.

– Додавання спрайтів з бібліотеки Unity:

– У панелі Tile Palette клацнути на "+" внизу та обрати "Add Tile Palette".

– Відкриється діалогове вікно. Де буде обрано бібліотеку спрайтів, з якої будуть імпортуватися спрайти.

– Обераються спрайти які потрібно додати до Tile Palette, після чого натискається "Add Selected". Вони з'являться у панелі Tile Palette готовими до використання.

– Використання спрайтів з Tile Palette здійснюється наступним чином:

– Перетягуються спрайти з Tile Palette на сцену або на Tilemap (прошарок з тайлами) для розміщення їх на рівні гри.

– Використовується інструменти редагування тайлів, щоб створювати ландшафти, платформи та інші елементи гри.

– Можна об'єднувати тайли, міняти їх розмір, поворот або додавати анімацію, використовуючи функції Tile Palette.

– Редагування Tile Palette реалізується в один з наступних способів.У панелі Tile Palette можна змінювати порядок та організацію спрайтів, створювати нові групи тайлів та категорії для зручного управління, або також можна змінювати властивості тайлів, такі як колізія, теги, тайлові об'єкти тощо.

Після виконання вище наведених кроків буде отримано готову до роботи панель Tile Palette, яка тепер має такий вигляд (рисунок 3.7).

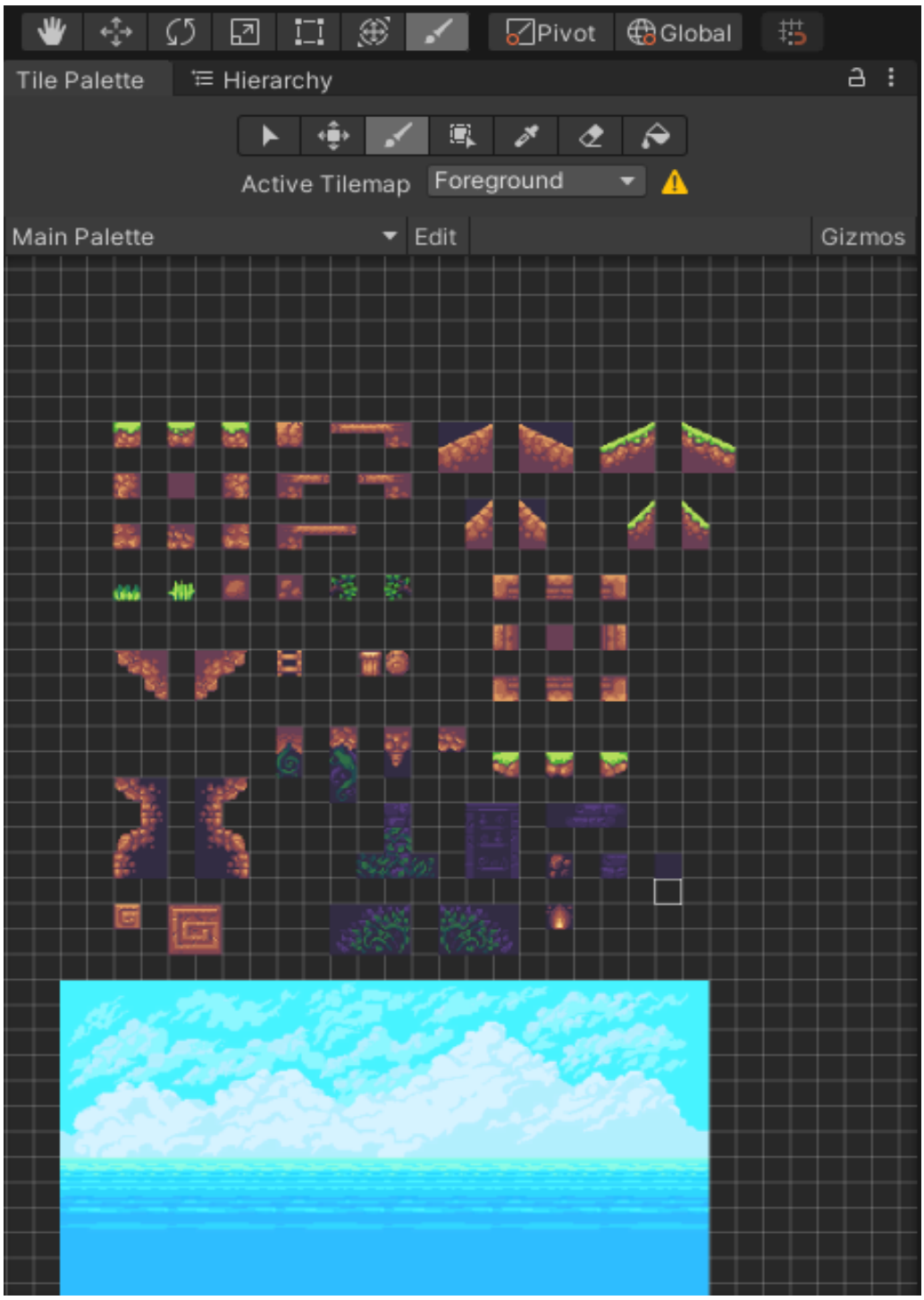

Рисунок 3.7 – Готова до роботи панель Tile Palette

На створеній панелі з допомогою відповідних інструментів відбувається реалізація передбачених рівнів.

3.2.5 Використання спрайтів для другорядних персонажів

У грі “Пригоди пса патрона” було додано персонажів, яких головний герой буде зустрічати на своєму шляху. Для різноманітності було додано декілька варіантів цих другорядних персонажів. Нижче наведено приклад їх реалізації у грі.

Для цього було використано готовий набір спрайтів з бібліотеки Unity. Після підключення готового набору до проекту у вікні “Project” з'являться папки з готовими спрайтами, які тепер потрібно налаштувати та добавити в сам проект.

На рисунку 3.8 зображено одного з персонажів Жабу, яка виконана в стилі піксель арту.

Рисунок 3.8 – Спрайт персонажа «Жаба»

Підчас створення персонажа додаються і попередньо налаштовуються маркери маршруту персонажа

3.2.6 Використання спрайту для створення об’єкта вишеньки

Спрайти вишеньки наявні в готовому пакеті “SunnyLand”. На рисунку 3.9 зображено її спрайт.

Рисунок 3.9 – Спрайт предмета «Вишенька»

В процесі гри можна буде підбирати вишеньки для отримання балів.

3.2.7 Налаштування колайдерів об’єктів

В програмі Unity фізика об'єктів може бути надана за допомогою різних компонентів, які надають колайдери об'єктам. Колайдери використовуються для забезпечення правильної колізійної детекції та реакції на фізичні взаємодії, такі як зіткнення, стикування, стрибання тощо.

Нижче наведено кілька основних компонентів, які додають колайдери на об'єкти:

- Box Collider (Прямокутний колайдер): Цей компонент додає прямокутний колайдер до об'єкта. Він використовується для створення колізійних областей у формі прямокутників, які можуть взаємодіяти з іншими об'єктами.
- Circle Collider (Коловий колайдер): Цей компонент додає коловий колайдер до об'єкта. Він використовується для створення колізійних областей у формі кола або кола з обмеженою областю.
- Rigidbody (Твердість): Компонент Rigidbody додає фізичні властивості до об'єкта, такі як маса, швидкість, сила, гравітація тощо. Він

дозволяє об'єктам реагувати на фізичні сили, такі як гравітація та впливи зіткнень.

– Collider 2D (2D колайдер): Цей компонент додає колайдер для об'єктів у 2D просторі. Він може бути використаний разом з Box Collider 2D, Circle Collider 2D та іншими компонентами для створення колізійних областей.

Додавання цих компонентів до платформ, об'єктів та персонажів дозволяє встановити фізичні властивості та поведінку для кожного об'єкта у грі. Наприклад, колайдери можуть використовуватись для запобігання проникненню об'єктів один через одного, для забезпечення зіткнень з перешкодами, а також для взаємодії з іншими об'єктами у грі. Компонент Rigidbody дозволяє об'єктам рухатись та реагувати на фізичні сили, що створює більш реалістичну фізичну модель у грі.

3.2.8 Реалізація рухів головного персонажа

Для роботи над скриптами, а сами скриптом для руху головного персонажа необхідно задати такі змінні:

- `private Rigidbody2D rb;`
- `private float speed;`
- `private float jumpForce;`

`private Rigidbody2D rb` – це приватна змінна типу Rigidbody2D, яка використовується для посилання на компонент Rigidbody2D персонажа. Rigidbody2D відповідає за фізичну поведінку об'єкта, включаючи рух та взаємодію з іншими об'єктами у сцені. Ця змінна дозволяє нам звертатись до різних методів та властивостей Rigidbody2D, щоб керувати рухом та фізикою персонажа.

`private float speed` – це приватна змінна типу float, яка визначає швидкість руху персонажа. Значення цієї змінної використовується для обчислення вектора швидкості персонажа при руху вліво або вправо. Змінюючи значення цієї змінної, ви можете контролювати швидкість руху персонажа.

`private float jumpForce` – це приватна змінна типу float, яка визначає силу стрибка персонажа. Значення цієї змінної використовується для додавання

сили до персонажа в напрямку висоти при стрибку. Змінюючи значення цієї змінної, ви можете контролювати висоту та силу стрибка персонажа.

Ці змінні використовуються для збереження значень, які впливають на рух та фізичну поведінку персонажа. Їх значення можна налаштувати в коді, щоб змінити характеристики руху та стрибка персонажа відповідно до ваших потреб.

Далі потрібно написати код який відповідає за рух персонажа вліво та вправо в залежності від вхідних команд гравця:

```
float hDirection = Input.GetAxis("Horizontal"):
```

Цей рядок коду зчитує значення вхідної осі "Horizontal". Воно вказує на напрямок руху персонажа. Значення зберігається в змінній hDirection:

```
if (hDirection < 0) { ... }:
```

Ця умова перевіряє, чи значення hDirection менше 0, що вказує на рух персонажа вліво. Якщо умова виконується, виконуються наступні дії:

```
rb.velocity = new Vector2(-speed, rb.velocity.y):
```

Встановлюється горизонтальна швидкість персонажа у від'ємному напрямку -speed, що призводить до руху вліво. Значення rb.velocity.y залишається без змін, що дозволяє персонажеві продовжувати вертикальний рух (якщо він був у русі):

```
transform.localScale = new Vector2(-1, 1):
```

Змінюється масштаб об'єкта (transform.localScale), встановлюючи його від'ємне значення по осі X (-1). Це змінює напрямок зображення персонажа, роблячи його оберненим вліво:

```
else if (hDirection > 0) { ... }:
```

Ця умова перевіряє, чи значення hDirection більше 0, що вказує на рух персонажа вправо. Якщо умова виконується, виконуються наступні дії:

```
rb.velocity = new Vector2(speed, rb.velocity.y):
```

Встановлюється горизонтальна швидкість персонажа у позитивному напрямку speed, що призводить до руху вправо. Значення rb.velocity.y залишається без змін, що дозволяє персонажеві продовжувати вертикальний рух (якщо він був у русі):

`transform.localScale = new Vector2(1, 1):`

Змінюється масштаб об'єкта (transform.localScale), встановлюючи його значення 1 по обох осях. Це повертає зображення персонажа в його початковий напрямок, роблячи його оберненим вправо.

Цей код забезпечує рух персонажа вліво або вправо з використанням горизонтальної швидкості (rb.velocity) та змінює напрямок зображення персонажа за допомогою масштабу (transform.localScale). Залежно від значення hDirection, встановлюється відповідний рух та напрямок персонажа.

Тепер персонаж може переміщатись в ліво та право, необхідно також додати можливість стрибків, для цього використовуємо такі команди:

```
if (Input.GetButtonDown("Jump") &&
coll.IsTouchingLayers(ground)):
```

Ця умова перевіряє, чи гравець натиснув кнопку стрибка (використовуючи визначену кнопку "Jump") і чи персонаж торкається платформи (ground). Якщо обидва умови виконуються, виконуються наступні дії:

`Jump()`: Викликається метод Jump(), який здійснює фізичний стрибок персонажа.

`jump.Play()`: Відтворює звуковий ефект стрибка (якщо такий є).

`private void Jump()`: Цей метод встановлює вертикальну швидкість персонажа, щоб забезпечити йому ефект стрибка.

`rb.velocity = new Vector2(rb.velocity.x, jumpForce):`

Встановлюється значення вертикальної швидкості (rb.velocity.y) персонажа на величину jumpForce. Це дає йому висоту стрибка.

`state = State.jumping:`

Встановлюється стан персонажа на "jumping" (стрибок). Це може використовуватись для управління анімацією або іншими діями, пов'язаними зі стрибком персонажа.

Отже, ці команди перевіряють, чи гравець натиснув кнопку стрибка та чи персонаж торкається платформи. Якщо обидві умови виконуються,

викликається метод Jump(), який встановлює вертикальну швидкість персонажа для здійснення стрибка, а також встановлює стан персонажа на "jumping".

3.2.9 Знищення негативних другорядних персонажів

Знищення негативних персонажів в грі передбачена за допомогою стрибка на них. Він є популярним та ефективним способом в комп'ютерних іграх, який додаватиме цікавість та виклик геймплею. Ось кілька причин, чому було обрано саме цей варіант:

– Інтерактивність: Знищення ворогів за допомогою стрибка на них вимагає від гравця точного та умілого керування персонажем. Гравець повинен правильно виміряти свій стрибок, спрямувати його та вчасно приземлятись на ворога, щоб його знищити. Це додає елемент виклику та взаємодії між гравцем і грою.

– Ризик та винагорода: Знищення ворогів за допомогою стрибка на них може мати елемент ризику. Гравець повинен бути уважним та визначити оптимальний момент для стрибка, оскільки неправильний розрахунок може призвести до втрати життя. В той же час, успішне знищення ворогів надає гравцю відчуття винагороди та задоволення від досягнення мети.

Порівняно з іншими варіантами знищення ворогів в платформерах, такими як стріляння, ближній бій або використання спеціальних знарядь, знищення ворогів за допомогою стрибка на них має свої унікальні переваги. Воно може бути більш простим та інтуїтивно зрозумілим для гравця, не вимагаючи складних комбінацій або додаткових контролів. Крім того, цей спосіб знищення ворогів стимулює гравця до активного руху та експериментування з геймплеєм, щоб знайти оптимальний шлях до перемоги.

Важливо також зробити розумну балансування геймплею, щоб забезпечити виклик та задоволення для гравця. Було створену різну висоту та розташування ворогів, щоб створити різноманітні ситуації та перешкоди, а також додано різні типи ворогів з різними властивостями.

Було написано код на мові програмуваня C# який відповідає за обробку зіткнення головного героя із ворогами, коли герой стрибає на ворогів, знищуючи їх. Лістинг коду подано в додатку А.

Розглянемо кожну функцію коду окремо:

`public void JumpedOn()`: Ця функція викликається при зіткненні головного героя з ворогом, коли герой стрибає на ворога.

`anim.SetTrigger("Death")`: Викликає тригер "Death" в компоненті Animator, що ініціює відтворення анімації смерті ворога.

`death.Play()`: Відтворює звуковий ефект смерті ворога.

`rb.velocity = Vector2.zero`:

Зупиняє горизонтальний рух героя, задавши його горизонтальну швидкість на нуль.

`private void Death()`: Ця функція викликається після анімації смерті ворога.

`Destroy(this.gameObject)`: Знищує об'єкт, до якого прикріплений цей скрипт (ворога).

3.2.10 Створення анімації головного героя

Щоб створити анімації для героя потрібно перейти на вкладку Animation (рисунок 3.10) в Unity. Для початку створюється анімаційний контролер, натиснувши правою кнопкою миші в розділі Project та вибравши Create - Animator Controller.

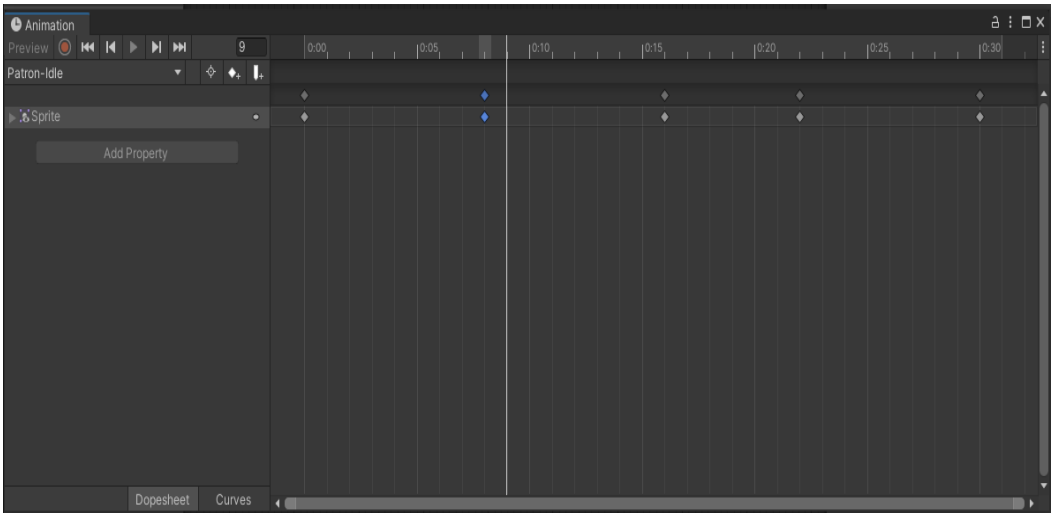

Рисунок 3.10 – Вікно Animation

– Використовується вже готовий спрайт який був створений у програмі Aseprite.

– Процес додавання анімацій:

– Перетягується спрайти для кожного стану на сцену Animator Controller.

– Встановлюється переходи між анімаціями для кожного стану, наприклад, коли персонаж рухається уліво, перехід до анімації руху уліво, коли він рухається управо, перехід до анімації руху управо тощо.

– Налаштовуються параметри переходів, такі як тригери або умови, які спрацьовують для зміни анімацій.

– Задаються швидкість відтворення, повторення анімацій та інші параметри для кожної анімації.

– Далі задаються зв'язки кожної анімації між собою, щоб отримати плавні переходи від однієї анімації до іншої. (рисунок 3.11)

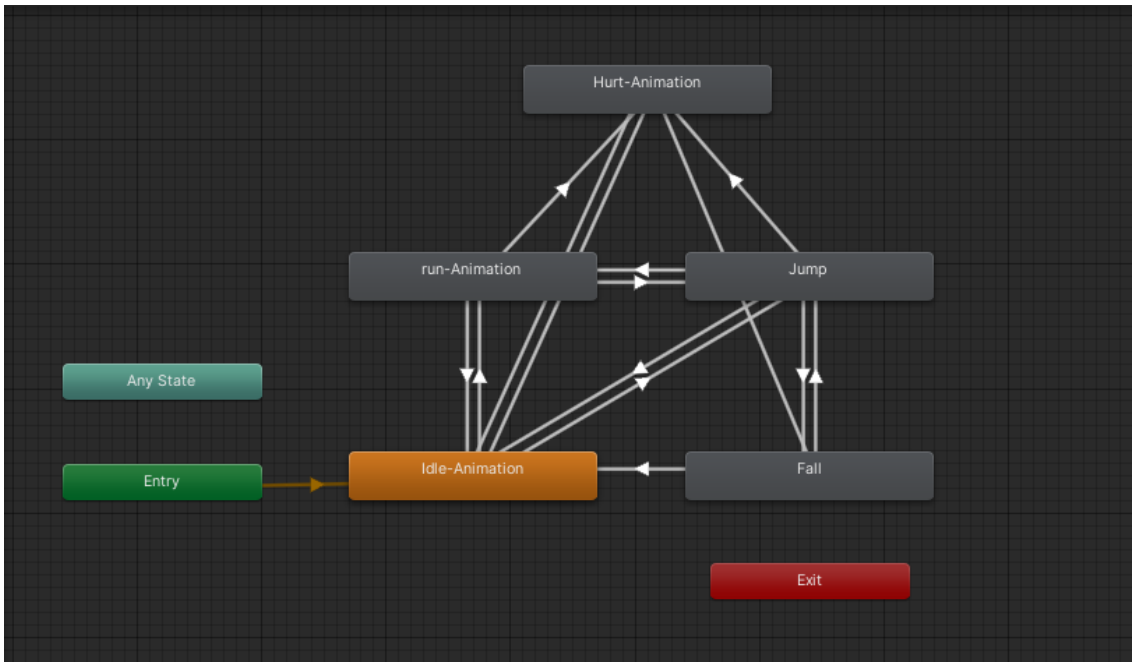

Рисунок 3.11 – Вікно зв’язків анімації головного героя

3.2.11 Реалізація та налаштування функціоналу негативних персонажів

Головним функціоналом негативних персонажів є патрулювання(переміщення в заданому діапазоні та реакція на зіткнення з головним героєм). Розглянемо це на прикладі реалізації функціоналу негативного персонажа “Жаба”.

Для початку був створений клас Enemy, якому підпорядковуються всі типи негативних персонажів:

```
public class Enemy : MonoBehaviour
{
    protected Animator anim;
    protected Rigidbody2D rb;
    protected AudioSource death;
    protected virtual void Start()
    {
        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
        death = GetComponent<AudioSource>();
    }
    public void JumpedOn()
    {
        anim.SetTrigger("Death");
        death.Play();
        rb.velocity= Vector2.zero;
    }
    private void Death()
    {
        Destroy(this.gameObject);
    }
}
```

Потім створюється новий персональний скрипт для кожного негативного персонажа окремо, та задаються дві точки:

```
private float leftCap;
private float rightCap;
```

Ці дві змінні, leftCap і rightCap, є позначками або межами, що використовуються для обмеження руху негативного персонажа у грі.

leftCap (ліва межа): Ця змінна вказує на ліву межа або максимальну ліву позицію, до якої може рухатись об'єкт або персонаж. Вона визначається у глобальних координатах, як певне числове значення.

rightCap (права межа): Ця змінна вказує на праву межу або максимальну праву позицію, до якої може рухатись об'єкт або персонаж. Вона також визначається у глобальних координатах як числове значення.

Негативний персонаж жаба буде пересуватися стрибками тому змінні jumpLength та jumpHeight визначають відстань та висоту стрибка ворога.

`facingLeft` – це змінна, яка показує, чи негативний персонаж повернутий вліво. Вона використовується для зміни напрямку ворога.

`Start()` – це метод, який викликається при старті негативного персонажа. Він викликає базову реалізацію методу з класу-батька (Enemy) та ініціалізує змінну coll.

`Update()` – це метод, який викликається кожен кадр і відповідає за оновлення стану ворога. Він перевіряє, чи ворог перебуває у стані стрибка і переключає анімацію на падіння, якщо ворог знаходиться у вільному падінні. Також перевіряється, чи ворог доторкнувся до "землі" після падіння, і встановлюється відповідний стан анімації.

`Move()` – це метод, який відповідає за рух ворога. В залежності від напрямку, в якому ворог звернутий, та його поточної позиції, ворог рухається вліво або вправо.

Якщо ворог досягне границі leftCap або rightCap, він змінює напрямок руху. Якщо негативний персонаж доторкнувся до "землі", він здійснює стрибок з відповідною силою і встановлює стан анімації "Jumping".

Цей код реалізує патрулювання негативного персонажа від однієї точки до іншої, використовуючи стрибки.

Негативний персонаж рухається зліва направо, дотримуючись встановлених меж leftCap і rightCap, і змінює напрямок, коли досягає однієї з

меж. Коли негативний персонаж доторкнувся до платформи, він здійснює стрибок у встановлений напрямок з відповідною силою.

3.2.12 Передчасне закінчення гри для головного героя

На початку отримується певна кількість «життів», по замовчуванню це 3 «життя». Візуалізація яких знаходиться у лівому куті екрану у вигляді сердечка та кількісного лічильника (рисунок 3.12).

Рисунок 3.12 – Лічильник кількості «життів»

Коли гравець успішно атакований негативним персонажем, він втрачає одне «життя». В разі якщо він падає з платформи, або потрапляє в пастку то втрачає відразу усі «життя». Коли усі «життя» втрачено, з'являється повідомлення про кінець гри і проходження рівеня починається спочатку.

У коді який відповідає за цей процес основна функція HandleHealth містить наступні кроки:

`hurt.Play()`: відтворення звуку пошкодження або болю гравця. Це звуковий ефект, який відтворюється, коли гравець отримує шкоду.

`health -= 1`: зменшення змінної health на 1. Це припускає, що health є змінною, яка відстежує кількість здоров'я гравця.

`healthAmount.text = health.ToString()`: оновлення текстового відображення кількості здоров'я гравця. healthAmount є посиланням на об'єкт тексту, який відображає здоров'я на екрані гри.

`if (health <= 0)` – перевірка , чи здоров'я гравця досягло або стало менше нуля, що означає, що гравець втратив усе здоров'я і повинен бути вважати програним.

`Cursor.lockState = CursorLockMode.Confined`: установка режиму миші в обмежений режим. Це означає, що мишка буде обмежена в межах вікна гри, що запобігає виходу курсора за межі гри.

`Time.timeScale = 0`: зупинка часу у грі. Встановлення значення 0 призводить до зупинки руху об'єктів і анімацій у грі.

`gameManager.gameOver()`: виклик функції gameOver() з об'єкта gameManager. Цей функція викликає екран програшу.

`gameOVER.Play()`: відтворення звуку, який сигналізує про кінець гри. `gameOVER` є змінною, який містить звуковий ефект для сигналу про кінець гри.

Отже, коли головний герой отримує пошкодження, його «життя» зменшується на 1 і оновлюється відображення «життя». Якщо «життя» досягло або нуля, виконуються додаткові дії, такі як блокування курсора, зупинка часу, виклик функції gameOver() і відтворення звукового ефекту, пов'язаного з кінцем гри.

3.2.13 Реалізація переходу на наступний рівень гри

Перехід на наступний рівень відбувається коли гравець активує тригер, який прикріплений до Box Collider і знаходиться у кінці кожного рівня.

У Unity можна налаштувати Box Collider як тригер (trigger), щоб він функціонував як область виявлення зіткнень без впливу фізичного зіткнення.

Коли об'єкт, до якого потрібно додати Box Collider вибрано, в інспекторі об'єкта потрібно натиснути на кнопку "Add Component".У випадаючому меню вводится "Box Collider" який обирається зі списку результатів. Це додасть Box Collider до об'єкта.

Обирається Box Collider як тригер.В інспекторі об'єкта знаходиться компонент Box Collider.Поле "Is Trigger" який перемикається в положення "Ввімкнено" (Enabled). Це встановить Box Collider як тригер.

Після цього додається скрипт до того ж об'єкта, до якого додано Box Collider.У скрипті реалізується наступний код:

```
public class LoadScren : MonoBehaviour
public class SceneChange : MonoBehaviour
{    [SerializeField] private string sceneName;
    private void OnTriggerEnter2D(Collider2D collision)
    {        if (collision.gameObject.tag == "Player")
        {            SceneManager.LoadScene(sceneName);
        }
```

Тепер коли гравець потрапляє у зону дію тригера його переміщує на наступний вищий рівень.У випадку досягнення останього рівня, з’являється відповідне повідомлення та аудіо супровід.

3.2.14 Реалізація меню паузи

У грі є можливість ставити її на паузу. Для цього необхідно просто натиснути клавішу “Esc”. Після чого на екрані з'явиться вікно паузи, що зображено на рисуноку 3.13. З'явиться 2 варіанти дій: Продовжити гру, або ж перейти до головного меню.

Рисунок 3.13 – Меню «Пауза»

У скрипті використано клас Input і призначена клавіша “Esc”.

Також скрипт містить у собі 3 функції:

- `public void Resume();`
- `public void Pause();`
- `public void LoadMenu();`

`public void Resume()`:

Ця функція відповідає за продовження гри після паузи.

`Cursor.lockState = CursorLockMode.Locked`: встановлює режим курсора в "заблокований", що означає, що курсор буде прихований та заблокований в центрі вікна гри.

`pauseGameMenu.SetActive(false)`: вимикає активність об'єкта меню паузи (pauseGameMenu), що приховує його на екрані.

`Time.timeScale = 1f`: встановлює швидкість часу на значення 1, що відновлює нормальний хід гри.

`PauseGame = false`: становлює змінну PauseGame в значення "false", що вказує на те, що гра не перебуває в режимі паузи.

`public void Pause()`:

Ця функція відповідає за зупинку гри та відображення меню паузи.

`Cursor.lockState = CursorLockMode.Confined`; встановлює режим курсора в "обмежений", що означає, що курсор буде обмежений у межах вікна гри.

`pauseGameMenu.SetActive(true)`: увімкнює активність об'єкта меню паузи (pauseGameMenu), що відображає його на екрані.

`Time.timeScale = 0f`: встановлює швидкість часу на значення 0, що зупиняє рух об'єктів та анімацію в грі.

`public void LoadMenu()`:

Ця функція відповідає за завантаження головного меню гри.

`Time.timeScale = 1f`: встановлює швидкість часу на значення 1, що відновлює нормальний хід гри перед завантаженням меню.

`SceneManager.LoadScene("StartM")`; завантажує сцену з назвою "StartM", що відповідає головному меню гри.

4 ТЕСТУВАННЯ 2D ГРИ ПЛАТФОРМЕРА«ПРИГОДИ ПСА ПАТРОНА»

4.1 Тестування кнопок головного меню 2D гри платформера «Пригоди Пса Патрона»

Після запуску гри гравець зустрічає головне меню (рисунок 4.1). На ньому зображено головного персонажа гри “Пригоди Пса Патрона”, який зображений у стилі піксель арт. Також тут зображені деякі негативно налаштовані персонажі яких можна буду зустріти під час проходження гри.

На вибір гравця є 3 кнопки: Почати гру, Обрати рівень та вихід.

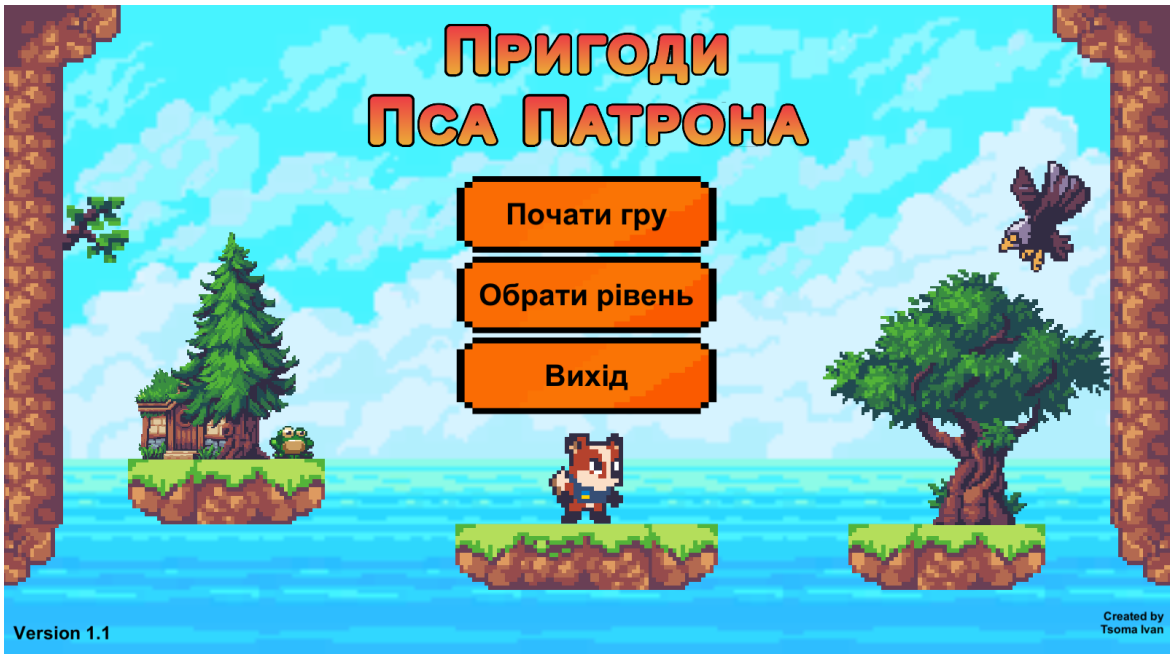

Рисунок 4.1 – Головне меню 2D гри платформера «Пригоди Пса Патрона»

Натиснувши “Почати гру”, гравець переходить на перший рівень і розпочинається гра, а якщо натиснути кнопку “Обрати рівень” то відкриється меню з вибором рівнів (рисунок 4.2) і вибравши один із них він відразу запуститься.

Рисунок 4.2 – Меню «Обрати рівень»

У тому випадку якщо натиснути кнопку “Вихід” гра просто закриється.

4.2 Тестування ігрового процесу 2D гри платформера«Пригоди Пса Патрона»

Після натиснення кнопки “Почати гру” гра розпочинається. Керувати персонажем можна за допомогою кнопок на клавіатурі A, D та Space. Кнопка A – це рух вліво, D – рух вправо, а Space – стрибок.

Гравець бачить головного героя, навколишній світ, негативних персонажів, платформи по яким можна стрибати, вишеньки які необхідно збирати, кількість його життів та графічний інтерфейс у лівому куті екрану, який вказує на кількість зібраних вишеньок. (рисунок 4.3).

Рисунок 4.3 – Інтерфейс гри «Пригоди Пса Патрона»

Якщо стрибнути на негативного персонажа то він зникає, але перед цим програється анімація вибуху на місці де був знищений персонаж (рисунок 4.4).

Рисунок 4.4 – Знищення негативного персонажа

Коли гравець не атакує негативного персонажа стрибком зверху, а просто стикається з ним, то втрачає 1 «життя» і програється анімація пошкодження героя (рис. 4.5).

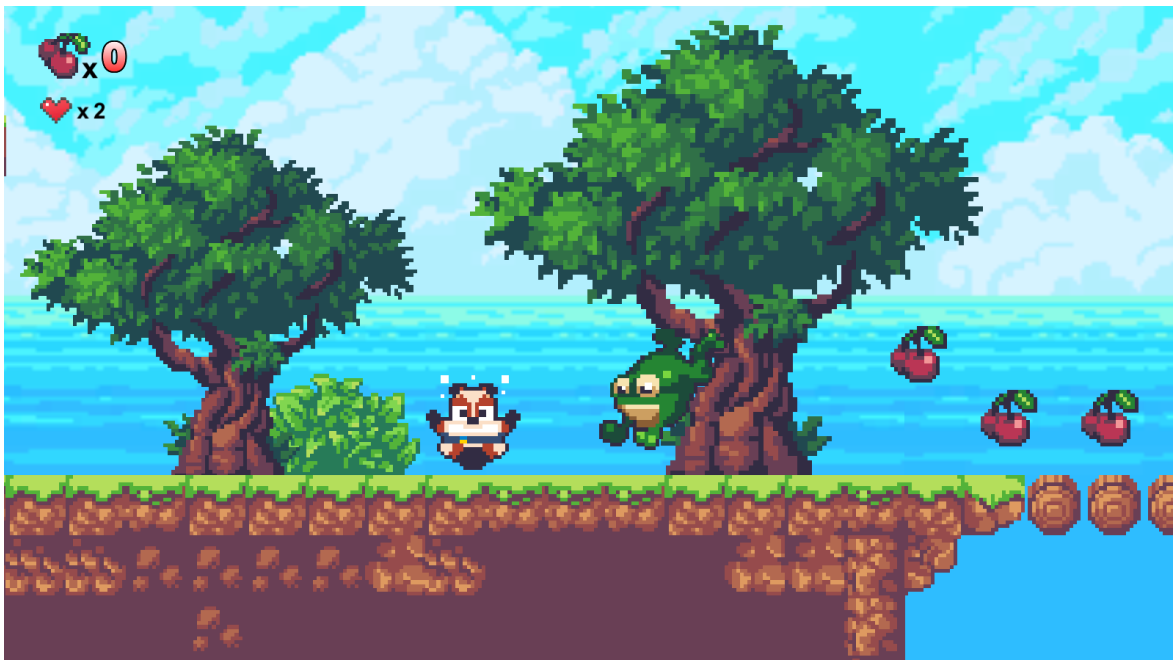

Рисунок 4.5 – Анімація пошкодження головного героя

На своєму шляху гравець буде зустрічати вишеньки які необхідно збирати, просто зіткнувшись з ними. В разі підбору вишеньки зараховуються бали, кількість яких зображено у лівому куті екрану (рисунок 4.6). Коли головний герой підбирає вишеньку то вона зникає, а натомість він отримує один бал.

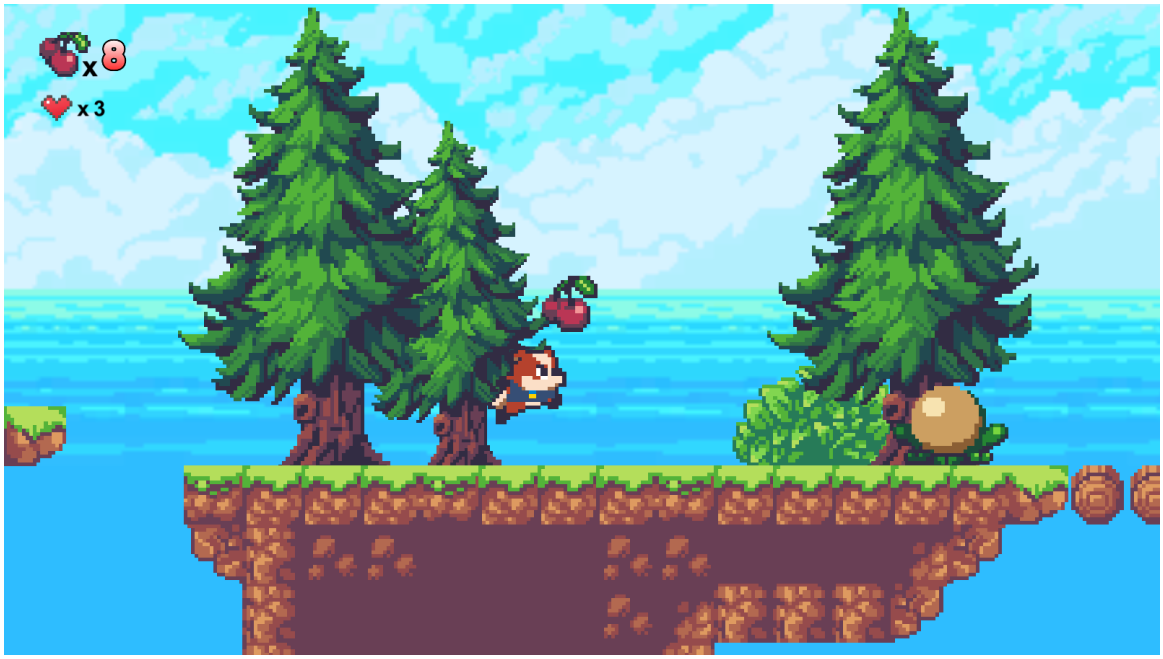

Рисунок 4.6 – Підбір вишеньки

4.3 Тестування успішного завершення 2D гри платформера «Пригоди Пса Патрона»

По сюжету головний герой відправляється на пошуки діаманта, який знаходиться на останньому рівні у печері (рисунок 4.7).

Рисунок 4.7 – Діамант у печері

Але перед цим він повинен здолати на своєму шляху негативних персонажів які заважають йому пройти далі, та уникати пасток, які розкидані по рівнях.

Діставшись до діаманта і підібравши його з'явиться напис, та аудіо супровід які сповіщають про успішне завершення гри (рисунок 4.8).

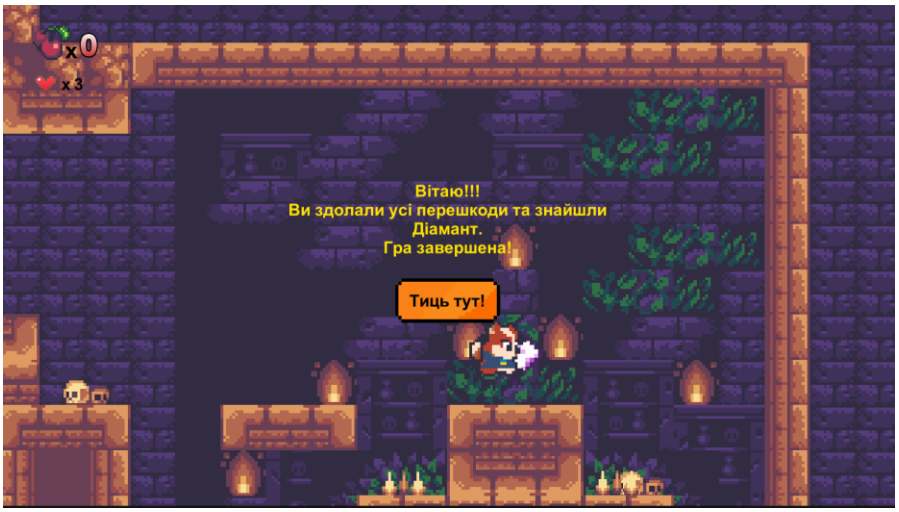

Рисунок 4.8 – Кінець гри

Після натискання на кнопку «Тиць тут!» гра завершується і запуститься головне меню.

5 ТЕХНІКО-ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ

5.1. Аналіз ринку

Аналізуючи ринок, варто зосередити увагу на аналізі ринку платформерів.

Економічно оцінивши ринок платформерів, стає зрозуміло, що він є значимим з економічної точки зору. Популярні платформери можуть мати значні прибутки для розробників та видавців. Велика аудиторія, яку привертають платформери, створює комерційну цінність для продукту. Комерційний успіх залежить від кількох факторів, таких як якість гри, маркетингова стратегія, цінова політика та конкуренція на ринку.

Платформери можуть бути випущені для різних платформ, таких як комп'ютери, консолі та мобільні пристрої. Розробники та видавці можуть залучити доходи з продажу самої гри, додаткових контентів (DLC), підписок та мікротранзакцій. Модель бізнесу може варіюватися залежно від конкретної гри та стратегії розробника.Крім того, платформери можуть створювати додаткові прибутки через ліцензування бренду або персонажів для інших продуктів, таких як іграшки, комікси або фільми. Це дозволяє розширити комерційний потенціал платформера та збільшити його вплив на ринку розваг.

Ринок платформерів в Україні демонструє зростання та потенціал для розвитку. За останні роки спостерігається збільшення інтересу до цього жанру серед українських гравців. Популярність платформерів пояснюється кількома факторами. По-перше, цей жанр є універсальним та привабливим для гравців різного віку та досвіду.

Він має класичні механіки та надає можливість насолоджуватися геймплеєм без складностей, пов'язаних з 3D-середовищем. По-друге, розробники платформерів в Україні продемонстрували високу якість та творчість в своїх проектах, що залучає увагу гравців.

Розробники та ігри

Україна має декілька відомих студій, які спеціалізуються на розробці платформерів. Наприклад, однією з найбільш відомих є студія "MADFINGER Games", яка створила популярну серію ігор "Shadowgun" та "Dead Trigger". Вони випустили також платформер "Monzo", який отримав позитивні відгуки.

Іншою відомою студією є "frogwares", яка розробила платформер "Magrunner: Dark Pulse" - гру з науково-фантастичною тематикою та цікавими головоломками.

Українські інді-розробники також активно допринають до ринку платформерів. Наприклад, студія "Alawar Premium" випустила платформер "Distrust", який отримав визнання за свою атмосферу та геймплейні механіки.

Тенденції та перспективи

Ринок платформерів в Україні має потенціал для подальшого росту та розвитку. Зростання інтересу до відеоігор серед української аудиторії, розвиток технологій та залучення талановитих розробників сприяють збільшенню популярності платформерів.

Однією з перспектив є збільшення кількості українських розробників, які спеціалізуються на платформерах та створюють якісні та оригінальні ігри. З'явлення нових талановитих студій та інді-розробників може призвести до появи нових цікавих проектів та розширення асортименту платформерів на ринку.Крім того, сприятливі умови для розвитку геймдеву в Україні, такі як підтримка уряду, активна геймдев-спільнота та зростання інтересу до індустрії, сприяють залученню інвестицій та розширенню можливостей для розробників платформерів.

Саме тому було взято за ідею створення 2D платформера, з головним героєм у ролі якого виступає усім відомий пес рятівник по кличці Патрон.

– Характеристика ринку збуту:

Ринок: Глобальний, національний, регіональний

Орієнтовна цільова аудиторія: Вікова категорія 10 - 35 років, фанати платформерів та ретро-ігор.

Конкуренти: Інші безкоштовні 2D-платформери, ретро-стильні ігри зі схожою механікою

Унікальність: Гра має відомого персонажа - пса патрона, що може привернути увагу фанатів цього персонажа

– Потенційні замовники (покупці) виробу:

Фанати платформерів та ретро-ігор

Геймери, які цінують безкоштовні ігри

Любителі пригодницьких ігор з яскравою графікою

Покупці, цікаві в іграх з відомими персонажами

– Ринки реалізації:

Онлайн-магазини громадських платформ (наприклад, Steam, Epic Games Store)

Офіційний веб-сайт гри

– Очікуваний попит на гру "Пригоди пса патрона":

Очікується попит серед фанатів платформерів, ретро-ігор та пригодницьких ігор.

Безкоштовна модель дозволяє привернути широку аудиторію гравців, в тому числі тих, хто шукає доступні ігри.

Очікується популярність серед фанатів відомого персонажа "пса патрона".

Попит може зростати з часом за рахунок позитивних відгуків, рекомендацій та вірусного поширення серед геймерської спільноти.

– Методи продажу виробу:

Гра «Пригоди пса патрона» буде доступна безкоштовно для завантаження та гри.

Опційно можуть бути включені внутрішні покупки або додатковий контент для заробітку на грі.

– Організація сервісного обслуговування:

Допродажне обслуговування: Можливість пропонувати додатковий контент, розширення або косметичні зміни для гри, які гравці можуть отримати.

Післяпродажне обслуговування: Випуск оновлень, виправлення помилок та реагування на фідбек користувачів для вдосконалення гри.

– Можливі обсяги продажу виробу:

Безкоштовна модель дозволяє залучити велику аудиторію гравців.

Обсяги продажу можуть залежати від популярності гри середньої цільової аудиторії, маркетингових зусиль та оцінок користувачів.

– Головні конкуренти на ринку аналогічної продукції:

Інші безкоштовні 2D платформери, ретро-стильні ігри.

Ігри з використанням відомих персонажів.

Ретро-платформери з подібним геймплеєм.

– Продукція конкурентів:

Основні техніко-економічні показники конкурентів можуть включати функціональні можливості гри, якість графіки та звуку, рівень складності геймплею, наявність додаткових функцій або режимів гри.

Споживчі показники можуть включати рейтинги та відгуки користувачів, загальну популярність гри середньої цільової аудиторії, інноваційність геймплею та оригінальність концепції.

– Рівень ціна на продукцію конкурентів:

Цінова політика конкурентів може бути різною і залежати від багатьох факторів, таких як розмір ігрової компанії, популярність гри, обсяг продажу, витрати на розробку та маркетинг.

Деякі конкуренти можуть пропонувати гру безкоштовно тоді за допомогою внутрішніх покупок або реклами, як інші можуть пропонувати гру за фіксовану ціну.

Конкурентні переваги:

Якість виробу та його надійність: Гарантована робота гри без помилок або збоїв, якісна графіка та звук, плавний геймплей.

Сервісне обслуговування: Гарантія вчасних оновлень та виправлення помилок, відповідь на фідбек користувачів, додатковий контент або розширення.

5.2. Розрахунок витрат на проектування

Витрати на проектування 2D гри "Пригоди пса патрона" можуть бути розраховані згідно з формою таблиці 2.1 вартості витрат.

Таблиця 2.1 – Розрахунок заробітної плати проектувальників

Найменування статей витрат	Сума, грн	Обґрунтування
1. Зарплата проектувальників	55 000	Сума зарплати, яка буде виплачуватися проектувальникам.
2. Відрахування на соціальні потреби	1 540	Сума відрахувань, яка буде сплачуватися за соціальними програмами та страхуванням.
3. Контрагентські роботи і послуги	1 050	Вартість робіт та послуг, які будуть виконуватися контрагентами (наприклад, графічний дизайн, музика, звукові ефекти тощо).
4. Витрати на відрядження	3 000	Витрати, пов'язані з відрядженнями та перебуванням учасників проекту в інших місцях.
5. Інші прямі витрати	3 150	Інші прямі витрати, що не входять в інші категорії (наприклад, закупівля обладнання, програмного забезпечення тощо).
6. Усього прямих витрат	63 740	Сума всіх прямих витрат.
7. Накладні витрати	19 122	Вартість накладних витрат, пов'язаних з управлінням проектом (наприклад, оренда офісу, комунальні послуги, офісне приладдя тощо).
8. Планові накопичення	16 559	Сума коштів, яка буде виділена для накопичення на майбутні проекти або розвиток компанії.
9. Усього, кошторисна вартість проекту	99 421	Загальна кошторисна вартість проекту, що складається з прямих та накладних витрат.
10. Податок на додану вартість	19 884	Сума податку на додану вартість, яка буде сплачуватися згідно з податковим законодавством.
11.Загалом, договірна ціна розробки Зп.	282 469	Загальна вартість проекту, включаючи всі витрати та податки.

1. Зарплата проектувальників.

Заробітна плата працівників визначається виходячи з кількості виконавців, діючих посадових окладів та кількості місяців їх участі в розробці проекту.

Розрахунок зарплати проводиться по формі таблиці 2.2.

Таблиця 2.2 – Розрахунок заробітної плати проектувальників

№	Посада	Оклад, грн/міс	Відрахування, грн/міс	Кількість чол.	Місяців	З/п, грн.
1	Проектувальник	7 000	1 365	1	5	28 700
2	Дизайнер	7 000	1 365	1	5	28 700
3	Усього зарплати:					57 400

Працівникам нараховані за повний відпрацьований місяць 7 000 грн. Податкова соціальна пільга до такої заробітної плати не застосовується

- Податок на доходи фізичних осіб: 7 000 Х 18% (ставка податку на доходи фізичних осіб) = 1 260 грн.
- Військовий збір**:** 7 000 Х 1,5% (ставка військового збору) = 105 грн.
- Єдиний внесок**:** 7 000 Х 22 % (ставка ЄСВ) = 1 540 грн.

Відрахування – 1 365 грн. (1 260 грн. + 105 грн.)

До виплати працівникові – 5 740 грн. (7 000 грн. – 1 260 грн. - 105 грн.)

2. Відрахування на соціальні потреби: 7 000 Х 22 % = 1 540 грн.

3. Контрагентські роботи і послуги: 7 000 Х 15 % = 1 050 грн.

4. Витрати на відрядження: = 3 000 грн.

5. Інші прямі витрати: 7 000 Х 45 % = 3 150 грн.

6. Усього прямих витрат: 66 140 грн. (57 400 + 1 540 + 1 050 + 3 000 + 3 150)

7. Накладні витрати: 66 140 Х 30% = 19 122 грн.

8. Планові накопичення: 82 799 Х 20 % = 16 559 грн.

9. Усього кошторисна вартість проекту – сума прямих і накладних витрат та планових накопичень : (63 740 + 19 122 + 16 559 = 99 421 грн.)

10. Податок на додану вартість визначається по діючому нормативу від кошторисної вартості проекту (20%): 99 421 Х 20 % = 19 884 грн.

5.3. Обґрунтування необхідності розробки

Задоволення потреб замовників: Безкоштовний 2D платформер може привернути увагу любителів ігор, які шукають якісний розважальний продукт. Замовники (покупці) будуть мати можливість насолоджуватись геймплеєм у своє задоволення без необхідності платити за це. Це може викликати позитивну реакцію у гравців та збільшити їхню лояльність до розробника.

Реклама та просування бренду безкоштовного платформера може бути використаний як інструмент для реклами та просування бренду. Розробник може включити в гру відображення свого логотипу або назви компанії, що забезпечить йому більшу впізнаваність. Це може привернути увагу до інших продуктів або послуг, які пропонує розробник.

Монетизація платформер за допомогою іншого джерела доходів є безкоштовним, розробник може використовувати інші джерела доходів, наприклад, рекламу або мікротранзакції. Вставлення рекламних банерів або пропонування платних додаткових елементів (наприклад, косметичні зміни для головного персонажу) може приносити розробнику прибуток.

Безкоштовний платформер може створити сприятливу атмосферу для розвитку спільноти гравців. Розробник може взаємодіяти зі спільнотою, проводити опитування, враховувати побажання гравців та оновлювати гру з урахуванням їхніх пропозицій.

Узагальнюючи, розробка безкоштовного 2D платформера з головним персонажем у ролі пса Патрона може привернути увагу гравців не тільки з України, ай і з усього світу та сприяти покращенню економічних показників через рекламу, монетизацію та розвиток спільноти. В подальшому розглядається можливе удосконалення продукту з цілю просування його на ринку платформерів.

ВИСНОВКИ

Результатом виконання завдань дипломного проєктування є 2D гра платформер «Пригоди Пса Патрона».

Данний продукт містить весь необхідний набір функціональних можливостей 2D гри платформера «Пригоди Пса Патрона». Інтерфейс розроблено в піксельному стилі та в яскравій кольоровій гамі.

Під час роботи над дипломним проєктом було:

– Досліджено та проаналізовано предметну область;

– Створено та описано сценарій 2D гри платформера «Пригоди Пса Патрона»;

– Розроблено алгоритми дій головного героя та другорядних персонажів;

– Спроєктовано інтерфейс користувача для 2D гри платформера «Пригоди Пса Патрона»;

– Проаналізовано та здійснено вибір інструментарію для реалізації 2D гри платформера «Пригоди Пса Патрона»;

– Детально описано окремі дії по реалізації функціоналу та інтерфейсу 2D гри платформера «Пригоди Пса Патрона»;

– Проведено успішне тестування на всіх етапах 2D гри платформера «Пригоди Пса Патрона» та техніко економічне обгрунтування.

До переваг створеної гри можна віднести: простий інтуїтивно зрозумілий україномовний інтерфейс користувача для будь - якої вікової категорії.

В майбутньому передбачено розширення функціоналу та безкоштовне розповсюдження 2D гри платформера «Пригоди Пса Патрона» для популяризації українського геймдеву

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Платформер. *Вікіпедія*: вебсайт. URL: https://uk.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D1%80 (дата звернення: 10.03.2023).
2. Введення в історію геймдизайну. Частина 2: Платформери. *Vokigames*: вебсайт. URL: https://vokigames.com/vvedenie-v-istoriyu-gejmdizajna-chast-2-platformery/ (дата звернення: 10.03.2023).
3. Онлайн сервіс. *Steam* : вебсайт. URL: https://store.steampowered.com/ (дата звернення: 10.03.2023).
4. Онлайн сервіс. *Electronic Arts*: вебсайт. URL: https://www.ea.com/?setLocale=en-us (дата звернення: 10.03.2023).
5. Онлайн сервіс. *Gog Galaxy*: вебсайт. URL: https://www.gog.com/en (дата звернення: 10.03.2023).
6. Платформер. *Super Meat Boy - Вікіпедія* : вебсайт. URL: https://uk.wikipedia.org/wiki/Super_Meat_Boy (дата звернення: 10.03.2023).
7. Онлайн сервіс Steam. *Платформер Broforce*: вебсайт. URL: https://store.steampowered.com/app/274190/Broforce/ (дата звернення: 10.03.2023).
8. Платформер. *Hollow Knight - Вікіпедія* : вебсайт. URL: https://uk.wikipedia.org/wiki/Hollow_Knight (дата звернення: 10.03.2023).
9. Онлайн сервіс Steam. *Платформер Starbound*: вебсайт. URL: https://store.steampowered.com/app/211820/Starbound/ (дата звернення: 10.03.2023).
10. Онлайн сервіс Steam. *Платформер Deadlight*: вебсайт. URL: https://store.steampowered.com/app/211400/Deadlight/ (дата звернення: 10.03.2023).
11. Журналісти дізналися, кому належать права на зображення пса Патрона. *Bird in flight*: вебсайт. URL: https://birdinflight.com/novini/20230109-patron-copyright.html (дата звернення: 20.04.2023).

12. David B. Hands-On *Game Development Patterns with Unity* / Baron David., 2019. – 116 с.

13. Створення 2D на Unity. Skillbox.ru: вебсайт. URL: https://skillbox.ru/media/code/kak_sozdat_prostuyu_2d_igru_na_unity/ (дата звернення: 20.04.2023).

14. Пояснення про спрайти, тайли. *Econdude* : вебсайт. URL: https://www.econdude.pw/2017/08/chto-takoe-graficheskij-sprajt-sprite.htmL (дата звернення: 20.04.2023).

15. Hocking J. *Unity in Action. Multiplatform game development in C# with Unity 5* / Joseph Hocking., 2015. – 352 с

16. Smith G. *Basic Math for Game Development with Unity 3D* / G. Smith, K. Sung., 2018. – 279 с.

17. Halpern J. *Developing 2D Games with Unity: Independent Game Program-ming with C#* / Jared Halpern., 2018. – 408 с

18. Unity Real-Time Development Platform. *Unity*: вебсайт. URL: https://unity.com/ru (дата звернення: 16.01.2023).

19. Unreal Engine 5. *UnrealEngine* : вебсайт. URL: https://www.unrealengine.com/en-US/unreal-engine-5 (дата звернення: 25.04.2023).

20. Godot Engine - Free and open source 2D and 3D game engine. *GodotEngine* : вебсайт. URL: https://godotengine.org/ (дата звернення: 25.04.2023).

ДОДАТКИ

Додаток А

Лістинг програми

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

public class Enemy : MonoBehaviour
{
    protected Animator anim;
    protected Rigidbody2D rb;
    protected AudioSource death;

    protected virtual void Start()
    {
        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
        death = GetComponent<AudioSource>();
    }

    public void JumpedOn()
    {
        anim.SetTrigger("Death");
        death.Play();
        rb.velocity= Vector2.zero;
    }

    private void Death()
    {
        Destroy(this.gameObject);
    }
                                    }
```

PlayerController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using UnityEngine.SceneManagement;

public class PlayerController : MonoBehaviour
{
    //Start() змінні
    private Rigidbody2D rb;
    private Animator anim;
    private Collider2D coll;

    //Екран GameOver
    public GameManager gameManager;

    //Екран Finish
    public Text Finish;

    //FSM
    private enum State {idle, running, jumping, falling, hurt}
    private State state = State.idle;

    //Інспектор змінних
    [SerializeField] private LayerMask ground;
    [SerializeField] private float speed = 5f;
    [SerializeField] private float jumpForce = 10f;
    [SerializeField] private int cherries = 0;
    [SerializeField] private TextMeshProUGUI cherryText;
    [SerializeField] private float hurtForce = 10f;
    [SerializeField] private AudioSource cherry;
    [SerializeField] private AudioSource footstep;
    [SerializeField] private AudioSource hurt;
    [SerializeField] private AudioSource jump;
    [SerializeField] private AudioSource gameOVER;
    [SerializeField] private int health;
    [SerializeField] private Text healthAmount;



    private void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        coll = GetComponent<Collider2D>();
        healthAmount.text = health.ToString();
    }

    private void Update()
```

```
    {
        if (state != State.hurt)
        {
            Movement();
        }
        AnimationState();
        anim.SetInteger("state", (int)state);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        // Фінал Trigger
        if (collision.tag == "Finish")
        {
            Cursor.lockState = CursorLockMode.Confined;
            Finish.gameObject.SetActive(true);
            Time.timeScale = 0;
        }

        if(collision.tag == "Collectable")
        {
            cherry.Play();
            Destroy(collision.gameObject);
            cherries += 1;
            cherryText.text = cherries.ToString();
        }
    }

    private void OnCollisionEnter2D(Collision2D other)
    {
        Enemy enemy = other.gameObject.GetComponent<Enemy>();
        if (other.gameObject.tag == "Enemy")
        {

            if (state == State.falling)
            {
                enemy.JumpedOn();
                Jump();
            }
            else
            {
                state = State.hurt;
                HandleHealth();

                if (other.gameObject.transform.position.x >
transform.position.x)
                {
                    //
                    rb.velocity = new Vector2(-hurtForce,
rb.velocity.y);
                }
                else
                {
```

```
                    //
                    rb.velocity = new Vector2(hurtForce,
rb.velocity.y);
                }
            }
        }
    }

    private void HandleHealth()
    {
        hurt.Play();
        health -= 1;
        healthAmount.text = health.ToString();
        if (health <= 0 )
        {
            Cursor.lockState = CursorLockMode.Confined;
            Time.timeScale = 0;
            gameManager.gameOver();
            gameOVER.Play();
        }
    }

    private void Movement()
    {
        float hDirection = Input.GetAxis("Horizontal");

        //Рух ліворуч
        if (hDirection < 0)
        {
            rb.velocity = new Vector2(-speed, rb.velocity.y);
            transform.localScale = new Vector2(-1, 1);
        }

        //Рух праворуч
        else if (hDirection > 0)
        {
            rb.velocity = new Vector2(speed, rb.velocity.y);
            transform.localScale = new Vector2(1, 1);
        }

        //Стрибок
        if (Input.GetButtonDown("Jump") &&
coll.IsTouchingLayers(ground))
        {
            Jump();
            jump.Play();
        }
    }

    private void Jump()
    {
        rb.velocity = new Vector2(rb.velocity.x, jumpForce);
        state = State.jumping;
```

```
    }


    private void AnimationState()
    {
        if(state == State.jumping)
        {
            if(rb.velocity.y < .1f)
            {
                state = State.falling;
            }
        }
        else if(state == State.falling)
        {
            if(coll.IsTouchingLayers(ground))
            {
                state= State.idle;
            }
        }
        else if (state == State.hurt)
        {
            if(Mathf.Abs(rb.velocity.x) < .1f)
            {
                state = State.idle;
            }
        }


        else if(Mathf.Abs(rb.velocity.x) > 1f)
        {
            //Pyx
            state = State.running;
        }
        else
        {
            state = State.idle;
        }
    }

    private void Footstep()
    {
        footstep.Play();
    }
                                    }
```

GameManager.cs

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

public class GameManager : MonoBehaviour
{
```

```
    public GameObject gameOverUI;

    public void gameOver()
    {
        gameOverUI.SetActive(true);
    }

    public void restart()
    {

SceneManager.LoadScene(SceneManager.GetActiveScene().name);
        Time.timeScale = 1;
    }

    public void MainMenu()
    {
        SceneManager.LoadScene("StartM");
        Time.timeScale = 1;
    }

    public void quit()
    {
        Application.Quit();
    }

    public void EndGame()
    {
        SceneManager.LoadScene("StartM");
        Time.timeScale = 1;
    }
```

CameraController.cs

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

public class CameraController : MonoBehaviour
{
    public Transform player;

    private void Update()
    {
        transform.position = new Vector3(player.position.x,
player.position.y, transform.position.z);
    }
}
```

Enemy.cs

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

```
public class Enemy : MonoBehaviour
{
    protected Animator anim;
    protected Rigidbody2D rb;
    protected AudioSource death;

    protected virtual void Start()
    {
        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
        death = GetComponent<AudioSource>();
    }

    public void JumpedOn()
    {
        anim.SetTrigger("Death");
        death.Play();
        rb.velocity= Vector2.zero;
    }

    private void Death()
    {
        Destroy(this.gameObject);
    }
                                    }
```

Frog.cs

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

public class Frog : Enemy
{
    [SerializeField] private float leftCap;
    [SerializeField] private float rightCap;

    [SerializeField] private float jumpLength = 10f;
    [SerializeField] private float jumpHeight = 15f;
    [SerializeField] private LayerMask ground;
    private Collider2D coll;


    private bool facingLeft = true;

    protected override void Start()
    {
        base.Start();
        coll = GetComponent<Collider2D>();
    }

    private void Update()
    {
```

```
            //Перехід від стрибка до падіння
            if (anim.GetBool("Jumping"))
            {
                if (rb.velocity.y < .1)
                {
                    anim.SetBool("Falling", true);
                    anim.SetBool("Jumping", false);
                }
            }
            //Перехід від падіння до руху
            if(coll.IsTouchingLayers(ground) &&
anim.GetBool("Falling"))
            {
                anim.SetBool("Falling", false);
            }

        }

        private void Move()
        {
            if (facingLeft)
            {
                if (transform.position.x > leftCap)
                {
                    if (transform.localScale.x != 1)
                    {
                        transform.localScale = new Vector3(1, 1);
                    }

                    if (coll.IsTouchingLayers(ground))
                    {
                        rb.velocity = new Vector2(-jumpLength,
jumpHeight);
                        anim.SetBool("Jumping", true);
                    }
                }
                else
                {
                    facingLeft = false;
                }
            }

            else
            {
                if (transform.position.x < rightCap)
                {
                    if (transform.localScale.x != -1)
                    {
                        transform.localScale = new Vector3(-1, 1);
                    }

                    if (coll.IsTouchingLayers(ground))
                    {
```

```
                    rb.velocity = new Vector2(jumpLength,
jumpHeight);
                    anim.SetBool("Jumping", true);
                }
            }
            else
            {
                facingLeft = true;
            }
        }
    }
```

DialogManager.cs

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

public class DialogManager : MonoBehaviour
{
    public Text dialogText;
    public Text nameText;

    public Animator boxAnim;
    public Animator startAnim;

    private Queue<string> sentences;

    private void Start()
    {
       sentences = new Queue<string>();
    }

    public void StartDialog(Dialog dialog)
    {
        boxAnim.SetBool("boxOpen", true);
        startAnim.SetBool("startOpen", false);

        nameText.text = dialog.name;
        sentences.Clear();

        foreach(string sentence in dialog.sentences)
        {
            sentences.Enqueue(sentence);
        }
        DisplayNextSentence();
    }

    public void DisplayNextSentence()
    {
        if(sentences.Count == 0)
```

```
            {
                EndDialog();
                return;
            }
            string sentence = sentences.Dequeue();
            StopAllCoroutines();
            StartCoroutine(TypeSentence(sentence));
        }

        IEnumerator TypeSentence(string sentence)
        {
            dialogText.text = "";
            foreach(char letter in sentence.ToCharArray())
            {
                dialogText.text += letter;
                yield return null;
            }
        }

        public void EndDialog()
        {
            boxAnim.SetBool("boxOpen", false);
        }
    }
```

Fall.cs

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

public class Fall : MonoBehaviour
{
    [SerializeField] private AudioSource fall;
    public GameManager gameManager;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.gameObject.tag == "Player")
        {
            Cursor.lockState = CursorLockMode.Confined;
            Time.timeScale = 0;
            gameManager.gameOver();
            fall.Play();
        }
    }
}
```

StartScren.cs

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

```
public class StartScrean : MonoBehaviour
{
    public GameObject levels;

   public void PlayGame()
    {
        SceneManager.LoadScene("LoadLevel1");
    }

    public void Levels()
    {
        levels.SetActive(true);
    }

    public void ExitGame()
    {
        Application.Quit();
    }

    public void Exit()
    {
        levels.SetActive(false);
    }

    public void Level1()
    {
        SceneManager.LoadScene("LoadLevel1");
    }

    public void Level2()
    {
        SceneManager.LoadScene("LoadLevel2");
    }

    public void Level3()
    {
        SceneManager.LoadScene("LoadLevel3");
    }
}
```

Dino.cs

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

[RequireComponent(typeof(BoxCollider2D))]

public class Dino : Enemy
{
    public List<Transform> points;
    public int nextID;
```

```
    int idChangeValue = 1;
    public float speed = 5;

    private void Reset()
    {
        Init();
    }

    void Init()
    {
        GetComponent<BoxCollider2D>().isTrigger = true;

        GameObject root = new GameObject(name + "_Root");
        root.transform.position = transform.position;
        transform.SetParent(root.transform);
        GameObject waypoints = new GameObject("Waypoints");
        waypoints.transform.SetParent(root.transform);
        waypoints.transform.position = root.transform.position;
        GameObject p1 = new GameObject("Point1");
p1.transform.SetParent(waypoints.transform); p1.transform.position
= root.transform.position;
        GameObject p2 = new GameObject("Point1");
p2.transform.SetParent(waypoints.transform); p2.transform.position
= root.transform.position;
        points = new List<Transform>();
        points.Add(p1.transform);
        points.Add(p2.transform);
    }

    private void Update()
    {
        MoveToNextPoint();
    }

    void MoveToNextPoint()
    {
        Transform goalPoint = points[nextID];

        if (goalPoint.transform.position.x > transform.position.x)
            transform.localScale = new Vector3(-1, 1, 1);
        else
            transform.localScale = new Vector3(1, 1, 1);

        transform.position =
Vector2.MoveTowards(transform.position, goalPoint.position, speed
* Time.deltaTime);

        if (Vector2.Distance(transform.position,
goalPoint.position) < 1f)
        {
            if (nextID == points.Count - 1)
                idChangeValue = -1;
```

```
            if (nextID == 0)
                idChangeValue = 1;

            nextID += idChangeValue;

        }
```

PauseMenu.cs

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

public class PauseMenu : MonoBehaviour
{
    public bool PauseGame;
    public GameObject pauseGameMenu;

    private void Start()
    {

    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (PauseGame)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        Cursor.lockState = CursorLockMode.Locked;
        pauseGameMenu.SetActive(false);
        Time.timeScale = 1f;
        PauseGame = false;
    }

    public void Pause()
    {
        Cursor.lockState = CursorLockMode.Confined;
        pauseGameMenu.SetActive(true);
        Time.timeScale = 0f;
        PauseGame = true;
```

```
    }

    public void LoadMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("StartM");
    }
}
```

LoadScren.cs

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

public class LoadScren : MonoBehaviour
{
    public int sceneID;

    public Slider loadSlider;

    private void Start()
    {
        StartCoroutine(LoadNextScene());
    }

    IEnumerator LoadNextScene()
    {
        AsyncOperation oper =
SceneManager.LoadSceneAsync(sceneID);
        while (!oper.isDone)
        {
            float progress = oper.progress / 0.9f;
            loadSlider.value = progress;
            yield return null;
        }
    }
}
```

Eagle.cs

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

[RequireComponent(typeof(BoxCollider2D))]

public class Eagle : Enemy
{
    public List<Transform> points;
    public int nextID;
```

```
    int idChangeValue = 1;
    public float speed = 5;

    private void Reset()
    {
        Init();
    }

    void Init()
    {
        GetComponent<BoxCollider2D>().isTrigger = true;

        GameObject root = new GameObject(name + "_Root");
        root.transform.position = transform.position;
        transform.SetParent(root.transform);
        GameObject waypoints = new GameObject("Waypoints");
        waypoints.transform.SetParent(root.transform);
        waypoints.transform.position = root.transform.position;
        GameObject p1 = new GameObject("Point1");
p1.transform.SetParent(waypoints.transform); p1.transform.position
= root.transform.position;
        GameObject p2 = new GameObject("Point1");
p2.transform.SetParent(waypoints.transform); p2.transform.position
= root.transform.position;
        points = new List<Transform>();
        points.Add(p1.transform);
        points.Add(p2.transform);
    }

    private void Update()
    {
        MoveToNextPoint();
    }

    void MoveToNextPoint()
    {
        Transform goalPoint = points[nextID];

        if (goalPoint.transform.position.x > transform.position.x)
            transform.localScale = new Vector3(-1, 1, 1);
        else
            transform.localScale = new Vector3(1, 1, 1);

        transform.position =
Vector2.MoveTowards(transform.position, goalPoint.position, speed
* Time.deltaTime);

        if (Vector2.Distance(transform.position,
goalPoint.position) < 1f)
        {
            if (nextID == points.Count - 1)
                idChangeValue = -1;
```

```
            if (nextID == 0)
                idChangeValue = 1;

            nextID += idChangeValue;

        }
    }
}
```

PlatformFall.cs

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

public class PlatformFall : MonoBehaviour
{
    Rigidbody2D rb;
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    private void OnCollisionEnter2D(Collision2D collision) {
        if (collision.gameObject.name.Equals("Player"))
        {
            Invoke("DropPlatform", 0.1f);
            Destroy(gameObject, 2f);
        }
    }

    void DropPlatform()
    {
        rb.isKinematic = false;
    }
}
```

РЕЦЕНЗІЯ

на дипломний проєкт

студента відділення комп'ютерних технологій

Галицького фахового коледжу імені В'ячеслава Чорновола

студента IV курсу групи КН-41

Цьома Івана Степановича

Спеціальність 122 „Комп'ютерні науки"

Обсяг дипломного проєкту: _80_стор.
Кількість сторінок записки: _97_стор.

Тема: «2D гра платформер на Unity»

Актуальність даного проєкту полягає в необхідності популяризації українськомовного сегменту індустрії розробки ігор. Важливим є використання світових тенденцій при розробці багаторівневого геймплея та включення різних ігрових механік для створення цікавого досвіду для гравців.

Практична або теоретична цінність опрацьованих питань. В ході написання дипломного проєкту було проведено детальний аналіз різних середовищ розробки, зокрема ігрових движків, з метою вибору найбільш підходящого для створення гри. На основі проведеного аналізу було обрано відповідний движок, що забезпечував потрібні можливості для реалізації задуманої гри.

В рамках проєкту були написані скрипти, що відповідають за функціональність гри. Ці скрипти забезпечують правильне функціонування головного героя та взаємодію з різними об'єктами в грі. Крім того, було створено рівні та оточення з використанням наявних спрайтів та текстур.

Недоліки роботи: в роботі варто привести діаграму класів розробленого ПЗ. В дипломі варто відобразити скріни програмного засобу на етапі розробки.

Загальний висновок: робота відповідає вимогам до дипломних проєктів освітньо кваліфікаційного рівня «молодший спеціаліст» зі спеціальності 122 «Комп'ютерні науки» та заслуговує оцінку «добре».

Рецензент Н.Г.Гавришків «23» 06 2023р.

ВІДГУК
на дипломний проєкт
студента відділення комп'ютерних технологій
Галицького фахового коледжу імені В'ячеслава Чорновола

IV курсу групи КН-41

_____ Цьоми Івана Степановича ____
(прізвище та ініціали)

Спеціальність *122„Комп'ютерні науки"*

Керівник ДП_____ Посвятовська О.Б._____

Тема: «2D гра платформер на Unity »

1. Загальна характеристика студента

У процесі роботи над дипломним проєктом студент глибоко розібрався у предметній області, проаналізував сучасні технології реалізації завдання, проявив себе як фахівець, який володіє засобами проєктування та створення ігрових програм, особливу увагу надає дизайнерським рішенням. Під час виконання плану дипломного проєктування студент продемонстрував вміння використовувати сучасні методи пошуку необхідної інформації, та її застосування для вирішення поставлених завдань аналізу джерел, проєктування та реалізації проєкту. Всі поставлені завдання виконував самостійно, дослухаючись до думки керівнику та у встановлені терміни.

2. Практична або теоретична цінність опрацьованих питань_____
полягає в створенні повнофункціонального ігрового продукту відповідної тематики, автор переконливо мотивує своє рішення щодо вибору жанру, створення сценарію, вибраних методів та засобів розробки ігрового продукту. Окрема увага при реалізації приділена технологіям візуалізації ігрових моментів

3. Недоліки роботи_____
В пояснювальній записці зустрічаються стилістичні неточності, що в загальному не позначається на функціонуванні реалізованого ігрового продукту та проєкту в цілому

4. Загальний висновок__ дипломний проєкт виконаний відповідно до поставлених завдань та загальних вимог до дипломних проєктів ОКР «молодший спеціаліст» і заслуговує на оцінку «добре»

Керівник дипломного проєкту _____ Ольга ПОСВЯТОВСЬКА
(прізвище та ініціали)

UNICHECK

Ім'я користувача:
Василь Кузик

ID перевірки:
1015614360

Дата перевірки:
15.06.2023 14:45:30 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
15.06.2023 14:48:20 EEST

ID користувача:
100012366

Назва документа: **Цьома**

Кількість сторінок: **78** Кількість слів: **11468** Кількість символів: **82532** Розмір файлу: **5.33 MB** ID файлу: **1015261941**

2.63%
Схожість

Найбільша схожість: **0.69%** з Інтернет-джерелом (https://ela.kpi.ua/bitstream/123456789/34398/1/Stavcev_bakalavr.pdf)

2.43% Джерела з Інтернету	172	Сторінка 80
0.56% Джерела з Бібліотеки	28	Сторінка 80

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1