МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

першого (бакалаврського) рівня вищої освіти

на тему: «Розробка комп'ютерної гри «Jump and Move» в жанрі 2D платформер з використанням середовища Unity»

Виконав: студент групи Іт-22сп
Спеціальності 126 - «Інформаційні системи технології»
(шифр і назва)

Леськів Н.Р.
(Прізвище та ініціали)

Керівник: Чаплига В.М.
(Прізвище та ініціали)

ДУБЛЯНИ-2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Освітній ступінь «Бакалавр» за спеціальністю –

126 – «Інформаційні системи та технології»

«ЗАТВЕРДЖУЮ»

Завідувач кафедри_____

д.т.н., проф. А.М. Тригуба

«____»_____202_ р.

ЗАВДАННЯ

на кваліфікаційну роботу студенту

Леськів Назарій Романович

1. Тема роботи: «Розробка комп’ютерної гри "Jump and Move" в жанрі 2D платформер з використанням середовища Unity».

Керівник роботи Чаплига Вячеслав Михайлович, д.т.н., професор.

Затверджені наказом по університету « 30 » грудня 2022 р. № 453/к-с.

2. Строк подання студентом роботи 15.06.2023 р.

3. Зміст розрахунково-пояснювальної записки:

1. Аналіз ринку комп’ютерних ігор в жанрі 2D платформер
2. Аналіз та вибір інструментальних засобів та технологій створення комп’ютерної гри "Jump and Move" в жанрі 2D платформер

3. Розробка комп’ютерної гри в жанрі 2D платформер з використанням середовища Unity та мови програмування C#
4. Розрахунок економічної ефективності програмного продукту
5. Охорона праці та безпека в надзвичайних ситуаціях

   Висновки та пропозиції

   Список використаних джерел

   Додатки
6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3, 4	Чаплига В.М., професор кафедри інформаційних технологій		
5	Городецький І.М., доцент кафедри управління проектами та безпеки виробництва		

7. Дата видачі завдання 30 грудня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Приміт-ка
1	Написання першого розділу та означення головних завдань роботи	30.12.2022 – 20.01.23	
2	Виконання другого розділу та вибір доцільних інструментів розробки	22.01.23 - 10.02.23	
3.	Виконання третього розділу та узагальнення отриманих результатів роботи	15.02.2023 -30.02.23	
4.	Написання розділу: «Розрахунок економічної ефективності програмного продукту»	01.03.23 - 15.03.23	
5.	Написання розділу: «Охорона праці та безпека в надзвичайних ситуаціях»	20.03.23 – 01.04.23	
6.	Завершення оформлення розрахунково-пояснювальної записки та аркушів графічної частини	15.04.23 – 30.04.23	
7.	Завершення роботи та перевірка на плагіат	05.05.23 – 15.06.23	

Студент _____ Леськів Н.Р.

(підпис)

Керівник роботи _____ Чаплига В.М.

(підпис)

УДК 004.5

Кваліфікаційна робота: 95 с. текст. част., 59 рис., 7 табл., 11 слайдів, 14 джерел, 3 додатки.

Розробка комп’ютерної гри “Jump and Move” в жанрі 2D платформер з використанням середовища Unity.

Леськів Н. Р. Кафедра ІТ. – Дубляни, Львівський НУП, 2023.

Виконано аналіз ринку комп’ютерних ігор в жанрі 2D платформер, зокрема аналіз сучасних представників жанру, обґрунтовано необхідність створення доступного проекту для широкого кола гравців.

Проведено аналіз та вибір засобів розробки для комп’ютерного 2D платформера. Підібрано оптимальний ігровий рушій та інтегроване середовище розробки для реалізації проекту.

Створено концепцію гри згідно сучасних тенденцій жанру, розроблені блок-схеми для грамотного конструювання архітектури проекту. Були запрограмовані усі необхідні компоненти, створено користувацький інтерфейс, додано анімації та звуковий супровід. Розроблено п’ять рівнів гри згідно базових основ дизайну рівнів. Після компонування всіх елементів розроблюваного платформера створено "exe" файл для запуску гри на платформі Windows.

Обґрунтовано економічну ефективність виконаної роботи.

Запропоновано заходи із охорони праці та безпеки в надзвичайних ситуаціях.

Ключові слова: 2D платформер, геймплей, ігровий рушій, Unity, колайдер, комп’ютерна гра, мова програмування, спрайт.

UDC 004.5

Qualification work: 95 p. text, 59 images, 7 tables, 11 slides, 14 sources, 3 applications.

Development of a computer game "Jump and Move" in the genre of 2D platformer using the Unity environment.

Leskiv N. R. Department of IT. - Dubliany, Lviv National Environmental University, 2023.

The market of computer games in the genre of 2D platformer is analyzed, in particular, the analysis of modern representatives of the genre, the necessity of creating an accessible project for a wide range of players is substantiated.

The analysis and selection of development tools for a 2D computer platformer is carried out. The optimal game engine and integrated development environment for the project are selected.

The game concept was created in accordance with the current trends of the genre, and flowcharts were developed for the competent design of the project architecture. All the necessary components were programmed, the user interface was created, and animations and music were added. Five levels of the game were developed according to the basic principles of level design. After arranging all the elements of the developed platformer, an "exe" file was created to run the game on the Windows platform.

The economic efficiency of the work performed is substantiated.

Measures for labor protection and safety in emergency situations are proposed.

Keywords: 2D platformer, gameplay, game engine, Unity, collider, computer game, programming language, sprite.

ПЕРЕЛІК СКОРОЧЕНЬ

- API (Application programming interface) - набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.
- AR (Augmented reality) – доповнена реальність.
- FPS (Frames per second) - кількість кадрів в секунду на екрані монітора чи телевізора.
- IDE (Integrated development environment) – інтегроване середовище розробки.
- NPC (Non-player character) - персонаж у комп'ютерних іграх, яким керує не гравець, а комп'ютер.
- SSD (Solid-state drive) - Твердотілий накопичувач. Комп'ютерний запам'ятовувальний пристрій на основі мікросхем пам'яті та контролера керування ними, що не містить рухомих механічних частин.
- UE (Unreal Engine) – ігровий рушій Unreal Engine.
- VR (Virtual reality) – віртуальна реальність.
- Ліцензія MIT - ліцензія відкритого і вільного програмного забезпечення, розроблена Массачусетським технологічним інститутом.
- ОС – операційна система.
- ПК – персональний комп'ютер.

Зміст

ВСТУП

Індустрія комп’ютерних ігор була сформована відносно недавно, але вже займає лідируючі позиції світового ринку розваг. Це стало можливим завдяки швидкому розвитку технологій та появі різноманітних інноваційних рішень в галузі комп’ютерного "заліза". Відеоігри сьогодні є відкритим та популярним способом проведення дозвілля порівняно з іншими видами розваг. Окрім цього, це важливий засіб навчання та розвитку. Навчальні ігри використовуються на багатьох платформах для підвищення ефективності освіти в різних сферах та підготовки спеціалістів. Також комп’ютерні ігри різнопланово стимулюють технологічний прогрес, оскільки розробники постійно працюють над покращенням якості та функціональності ігрових компонентів.

Можна стверджувати, що наш проект буде мати попит серед своєї цільової аудиторії. Це пояснюється вибором жанру та стилізації проекту. Платформери є популярними серед гравців завдяки своїй простоті та динамічному геймплею. Ці ігри мають низький поріг входження та не вимагають потужної апаратної частини, що робить їх доступними для широкого кола людей. Варто також зазначити, що спільнота геймерів перенасичена великомаштабними проектами, які вимагають значну кількість часу на освоєння ігрових механік та сучасно обладнаних комп’ютерів. На фоні цього, гра з зрозумілим геймплеєм та низькими системними вимогами зможе привернути увагу людей, які не готові приділяти їй багато часу.

Об’єктом дослідження є процес розробки двовимірних комп’ютерних ігор.

Предметом дослідження є створення комп’ютерної гри в жанрі 2D платформер з використанням середовища Unity.

Актуальність даного проекту полягає у забезпеченні дозвілля певної цільової аудиторії гравців завдяки простоті і зрозумілості жанру платформер.

Метою роботи є розробка комп’ютерної гри “Jump and Move” в жанрі 2D платформер використанням середовища Unity.

Завдання дослідження створити однокористувацький двовимірний платформер для персонального комп’ютера шляхом вирішення наступних задач:

- Аналіз ринку комп’ютерних ігор в жанрі 2D платформер.
- Аналіз та вибір інструментальних засобів і технологій створення комп’ютерної гри "Jump and Move" в жанрі 2D платформер.
- Розробка комп’ютерної гри в жанрі 2D платформер в середовищі Unity з використанням мови програмування C#.

Для досягнення мети було проаналізовано сучасні тенденції в розробці комп’ютерних ігор, досліджено засоби розробки комп’ютерних ігор та аналогічні проекти. На основі цих елементів створено однокористувацький двовимірний платформер для персональних комп’ютерів, що містить в собі п’ять рівнів з основною ігровою механікою.

Для створення комп’ютерного 2D платформера були використані такі технології та засоби: мова програмування C#, ігровий рушій Unity, IDE Microsoft Visual Studio.

Практична цінність нашої кваліфікаційної роботи полягає в наступних аспектах:

- Освоєння технологій: Розробка гри з використанням середовища Unity дозволяє студенту поглибити свої знання та навички у галузі геймдевелопменту та програмування.
- Розширення технічних навичок: Робота з Unity надає можливість вивчити і використовувати різноманітні функції та інструменти, такі як створення рівнів, анімацій, фізики, штучного інтелекту та інших компонентів, що розширює багаж технічних навичок студента.
- Проектування та управління процесом розробки: Розробка комп’ютерної гри вимагає створення концепції, дизайну рівнів, геймплею та інтерфейсу. Це дозволяє студенту отримати досвід у проектуванні та управлінні процесом розробки гри.
- Творчість та самовираження: Розробка власної комп’ютерної гри дає нам можливість втілити свої ідеї, творчість та власний стиль у грі. Це дозволяє розкрити свій потенціал як розробника і виразити себе через створену гру.

- Практичний досвід та портфоліо: Результатом дипломної роботи є готова комп’ютерна гра, яку можна включити до свого портфоліо. Це дає змогу демонструвати свої навички та досягнення потенційним роботодавцям у галузі розробки відеоігор.
- Задоволення користувачів: Кінцевим результатом розробки гри є задоволення гравців. Практична цінність полягає у тому, що ми маємо можливість створити розважальний продукт, який може надати задоволення і радість гравцям.

Отже, розробка комп’ютерної гри з використанням Unity має практичну цінність, оскільки сприяє освоєнню технологій, розширенню технічних навичок, розвитку творчості та самовираження, набуттю практичного досвіду та підвищенню нашої конкурентоспроможності на ринку роботи у сфері розробки відеоігор.

РОЗІДЛ 1
АНАЛІЗ РИНКУ КОМП'ЮТЕРНИХ ІГОР В ЖАНРІ 2D ПЛАТФОРМЕР

1.1 Поняття 2D платформера

2D платформер - це жанр ігор, в яких гравець керує персонажем, що рухається по платформах у двовимірному просторі. Головною механікою гри є переміщення персонажа вліво, вправо та стрибки, а також взаємодія з різноманітними об'єктами на рівнях, такими як платформи, сходи, вороги, перешкоди та різні предмети. В деяких проектах є можливість отримувати нові навички та покращувати свої характеристики по ходу гри, а також виконувати додаткові різноманітні дії.

Головне завдання під час гри полягає в успішному подоланні різних перешкод, збиранні предметів, битвах з "ворогами" та проходженні рівнів.

Платформери є одним з найстаріших та найвпізнаваніших жанрів в індустрії відеоігор. Вони зародилися ще в епоху аркадних автоматів та 8-бітних консолей і з тих пір завоювали серця багатьох гравців.

Історію 2D платформерів можна відслідкувати ще з початку ери відеоігор. У 1980-х роках, зокрема, вийшла широковідома гра "Donkey Kong", яка стала першим культовим представником жанру платформерів. Ця гра вперше ввела основні механіки, такі як біг, стрибки та переміщення персонажа по рівнях, повних викликів і перешкод.

Рисунок 1.1 – Ігровий автомат "Donkey Kong"

Після успіху "Donkey Kong" компанія Nintendo у 1983 році випустила гру "Mario Bros", в якій з’явився кооперативний геймплей та елементи багатокористувацької гри. Вона заклала основу для культової серії "Super Mario Bros", яка дебютувала на Nintendo Entertainment System (NES) у 1985 році. "Super Mario Bros" зробила революцію в жанрі платформерів завдяки захоплюючому геймплею, складному дизайну рівнів і різним культовим елементам, що надовго залишаться в пам’яті геймерів [11].

Наприкінці 1980-х - на початку 1990-х років платформери зазнали значного розвитку та інновацій. Консоль Sega Genesis вивела "Їжачка Соніка" на перший план у 1991 році. З її акцентом на швидкості та барвистому оточенні, "Sonic the Hedgehog" запропонувала унікальну альтернативу повільнішим іграм Mapio.

Рисунок 1.2 – Гра "Sonic the Hedgehog"

У 1990-х роках жанр пережив золотий вік. Такі ігри, як "Super Mario World", "Sonic the Hedgehog" та "Mega Man" розсунули межі 2D-платформерів, створивши більші світи, складний дизайн рівнів та персонажів, що запам’ятовуються. Ця епоха стала свідком появи культових талісманів і франшиз, які продовжують впливати на жанр і сьогодні [9].

З наближенням 2000-х років 3D-ігри стали новою межею, а 2D-платформери відійшли на другий план. Однак вони пережили відродження з появою цифрової дистрибуції та незалежної розробки ігор. Такі ігри, як "Super Meat Boy" (2010) та

"Braid" (2008) продемонстрували креативність та інноваційність, яких все ще можна досягти в жанрі 2D-платформерів.

Рисунок 1.3 – Гра "Super Meat Boy"

Гра "Super Meat Boy" здобула популярність завдяки високому рівню складності, пропонуючи особливий досвід для гравців. Гра вирізняється динамічним і безкомпромісним геймплеєм з численними перешкодами та ворогами. Головний герой має чутливе управління, що дозволяє здійснювати точні стрибки та маневри для уникнення небезпечних пасток. Унікальний візуальний стиль "Super Meat Boy" поєднує піксельну графіку з кривавими та інтенсивними ефектами. Понад 300 рівнів гри пропонують значну кількість контенту та різноманітних головоломок. Гра отримала схвальні відгуки критиків і стала одним з найбільш впізнаваних і улюблених платформерів останніх років, відомим своїм складним геймплеєм, швидкістю та особливим стилем.

2D платформери є популярним жанром серед гравців різного віку, оскільки вони пропонують захоплюючий геймплей, вимагають від гравця швидкості реакції та точності, а також дають можливість досліджувати різноманітні рівні та розв’язувати головоломки. 2D платформери можуть бути як класичними, віддзеркалюючи старі ігри на автоматах, так і сучасними, використовуючи нові технології та механіки для створення унікального геймплею.

Також ці ігри мають попит на різних платформах включаючи консолі, персональні комп’ютери та мобільні пристрої. Це робить платформери більш популярними, оскільки не кожен жанр може прижитися на усіх ігрових пристроях.

Різноманітність та жвавість геймплею, доступність на усіх платформах та низький поріг входження, робить жанр 2D платформерів цікавим широкому колу людей. Кожен може знайти особливий проект, що буде відповідати інтересам і дозволить гарно провести час.

1.2 Тенденції в жанрі сучасних 2D платформерів

Сьогодні 2D-платформери продовжують процвітати, приваблюючи як ностальгуючих гравців, так і нові покоління. Вони увібрали в себе усі найкращі практики та аспекти своїх попередників, пропонуючи новий унікальний та захопливий досвід. Цей жанр залишається улюбленою частиною ігрового світу, поєднуючи в собі складний ігровий процес, дослідження та відчуття пригод.

"Limbo," що побачила світ у 2010 році, та "Fez," що дебютувала у 2012 році, належать до новаторських платформерів, що здійснили експерименти з ігровими механіками, наративом та художніми стилями. "Limbo" поглинала гравців своїм атмосферним чорно-білим стилем, хвилюючим настроєм та головоломками, що ґрунтувалися на методі проб і помилок. Вона пропонувала мінімалістичну наративну лінію, що закликала гравців розкривати її приховані сенси та розгадувати таємниці моторошного світу.

Рисунок 1.4 – Гра "Fez"

З іншого боку, "Fez" представив новаторську механіку, яка дозволяла гравцям обертати ігровий світ на 90 градусів, відкриваючи нові перспективи та приховані шляхи. Піксельна ретро-графіка в поєднанні з яскравими кольорами створює візуально приголомшливу та ностальгічну атмосферу. Хитромудрі головоломки гри вимагали від гравців розшифровувати складні символи та візерунки, додаючи до ігрового процесу ще один рівень інтелектуального виклику.

І "Limbo", і "Fez" розширили межі жанру платформерів, експериментуючи з атмосферною естетикою та елементами розв’язання головоломок. Вони продемонстрували творчий потенціал інді-ігор у створенні захопливого ігрового досвіду, який залучає гравців на кількох рівнях. Ці ігри стали впливовими в ігровій індустрії, надихнувши інших розробників досліджувати нетрадиційні підходи до ігрового дизайну та сторітелінгу.

Окремо варто розповісти про "Hollow Knight", що посідає особливе місце серед 2D-платформерів, виділяючись як справді видатна гра, що завоювала серця гравців по всьому світу. Випущена у 2017 році талановитою інді-студією Team Cherry, Hollow Knight принесла свіжий погляд на жанр, представивши інноваційні механіки геймплею, чудово промальований світ та захоплюючу атмосферу. Вона швидко здобула визнання і відтоді стала улюбленою грою, що славиться своїми унікальними якостями та винятковим досвідом, який пропонує гравцям.

"Hollow Knight" також має глибокий і взаємопов’язаний світ для дослідження. Гравці мандрують величезним підземним королівством, зустрічаючись з різноманітними персонажами, розкриваючи таємниці та борючись зі складними ворогами. Нелінійний розвиток гри дозволяє гравцям відкривати нові області у власному темпі, заохочуючи допитливість і дослідження.

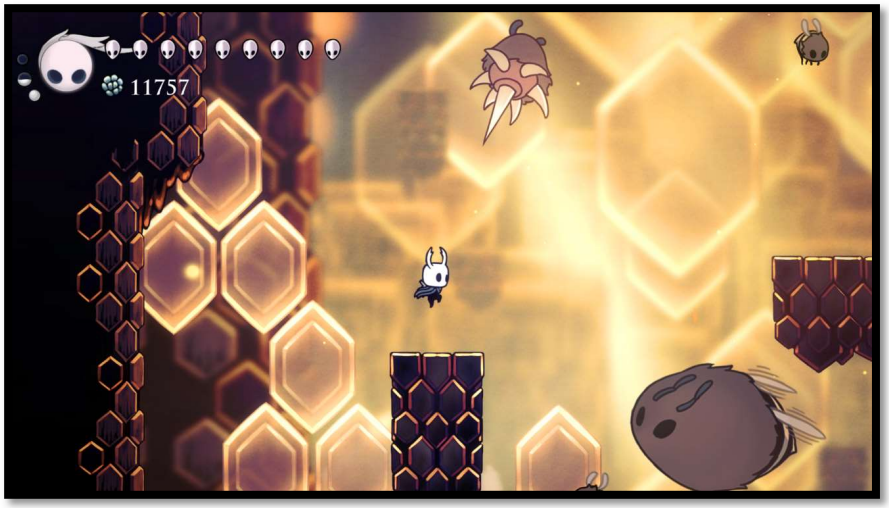

Рисунок 1.5 – Гра "Hollow Knight"

Бій у "Hollow Knight" точний і складний. Гравці повинні оволодіти різноманітними бойовими прийомами та здібностями, щоб здолати грізних "босів" і "ворогів". У грі дотримано баланс між стратегічним боєм і чутливим управлінням, що забезпечує задоволення від ігрового процесу.

Загалом, "Hollow Knight" дивує гравців захоплюючими візуальними ефектами, дизайном рівнів, складним геймплеєм та захоплюючим сюжетом. Здатність гри бездоганно поєднувати ці елементи принесла їй відданих фанатів і закріпила за нею статус видатної гри у світі 2D-платформерів.

Також в жанрі виділяється унікальний проект під назвою "Cuphead". Це високо оцінений 2D-платформер, який вийшов у 2017 році. Він був розроблений студією StudioMDHR і здобув визнання завдяки своєму унікальному художньому стилю та захоплюючому геймплею. Однією з визначних особливостей "Cuphead" є його мальована анімація, яка імітує класичні мультфільми 1930-х років. Цей особливий візуальний стиль, разом із джазовим саундтреком, створює ностальгічний та захоплюючий досвід для гравців.

Що вирізняє "Cuphead" у жанрі 2D-платформерів, так це її складність. Гра відома своїм складним і невблаганним геймплеєм, що вимагає точного розрахунку часу і швидких рефлексів. Гравці стикаються з серією напружених битв з "босами" та платформними секціями, які вимагають вмілого маневрування та розпізнавання

образів. Високий рівень складності додає відчуття досягнення та задоволення від подолання ігрових викликів.

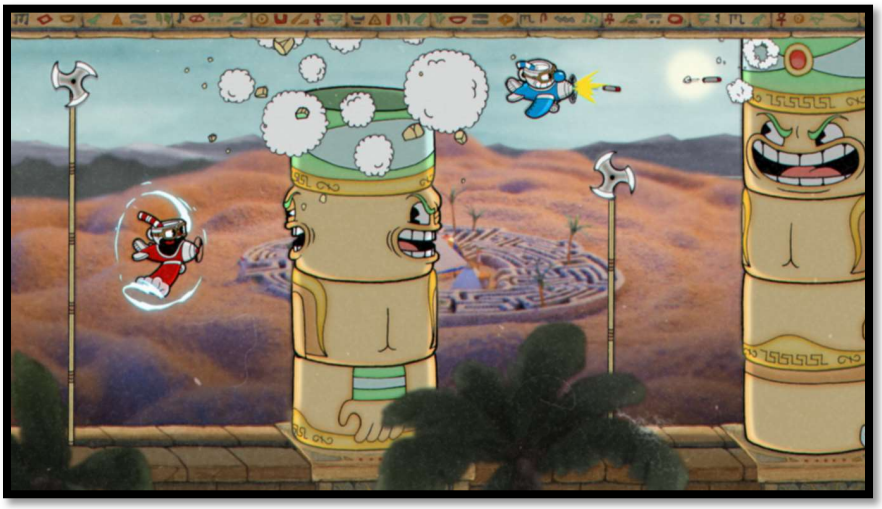

Рисунок 1.6 – гра "Cuphead"

Ще одним важливим аспектом "Cuphead" є можливість спільного проходження, що дозволяє гравцям об’єднатися з другом для спільної гри. Цей кооперативний режим додає додатковий рівень веселощів та товариськості, оскільки гравці можуть розробляти стратегію та працювати разом, щоб перемогти складних "босів".

Загалом, поєднання унікально художнього стилю, складного геймплею та кооперативного режиму зробили "Cuphead" видатною грою у світі 2D-платформерів. Вона пропонує особливий та захоплюючий досвід для гравців, які шукають нових випробувань.

Сучасні платформери залишаються надзвичайно популярними, і це не дивно. Нові представники жанру постійно розвиваються та еволюціонують, впроваджуючи нові механіки та інновації. Вони не обмежуються простими переходами і стрибками, але включають складніші рухи, пазли, елементи розвідки та багато іншого. Це дозволяє гравцям досліджувати різноманітність геймплейних ситуацій та випробовувати свої навички.

Ще одним аспектом популярності є низький поріг входження. Це означає, що навіть новачки в геймінгу можуть з легкістю освоїти цей жанр. Багато з них

пропонують короткі рівні або можливість зберігати прогрес, що дозволяє грати в них у вільний час або на коротких перервах між справами.

Навіть новинки жанру приваблюють гравців низькими вимогами до комп'ютерного заліза, що дозволяє насолоджуватися грою навіть на менш потужних пристроях.

Крім того, сучасні платформери звертають особливу увагу на візуальний і звуковий дизайн. Вони використовують вражаючу графіку, виразні кольори та деталізацію, щоб створити чарівні та захоплюючі світи. Звуковий дизайн, включаючи музику та звукові ефекти, вносить важливий внесок у створення атмосфери та настрою гри.

Взагалі, сучасні платформери поєднують у собі набір нових механік, унікальні художні рішення, а також цікаві сюжети, що перевершують очікування і стереотипи про цей жанр. Вони надають можливість гравцям погрузитися в чарівні світи та випробувати свої навички в захоплюючих пригодах.

1.3 Постановка задачі створення комп'ютерної гри "Jump and Move" в жанрі 2D платформер

На сьогоднішній день ігрова індустрія розвивається неймовірно швидко, кількість активних гравців зростає з кожним роком, а також зростає дохід, який впродовж останніх шести років майже подвоївся і на кінець 2016 року становив понад 90 мільярдів доларів. На всьому ігровому ринку доходи зросли ще більше - з 73 мільйонів доларів у 2010 році до майже 3 мільярдів у 2016 році. Все це свідчить про те, що розробка відеоігор є одним з найбільш перспективних напрямків у сфері розробки програмного забезпечення [14].

Можна точно сказати, що розроблювана гра буде успішною серед своєї цільової аудиторії. Цьому сприяють обрані жанр і стилізація проекту. Жанр платформер досить популярний серед гравців, оскільки являє собою простий і зрозумілий, а також цікавий ігровий процес. Маючи дуже низький поріг

входження, такі ігри доступні надзвичайно широкому колу людей, зокрема тим, хто мало цікавиться іграми.

Так само варто пам’ятати, що геймери не завжди можуть приділяти багато часу грі. Для прикладу на проходження сучасного високобюджетного проекту потрібно виділити як мінімум кілька десятків годин. На їхньому тлі платформер з інтуїтивним геймплеєм виглядатиме більш простим та доступним для людей, у яких є бажання, але мало часу для розваг.

На основі вище перерахованого легко виділяються мета і призначення проекту.

Мета проекту полягає в наданні засобу для приємного проведення часу.

Призначення проекту полягає в розважанні потенційного споживача.

Щоб цього досягнути потрібно виконані наступні кроки:

- Провести огляд сучасних ігрових рушіїв.
- Вибрати оптимальний рушій для розробки 2D платформера.
- Проаналізувати та вибрати IDE.
- Створити концепції гри на основі сучасних тенденцій в жанрі.
- Запрограмувати елементи гри мовою C#.
- Створити анімації та додати звукові ефекти в середовищі Unity.
- Спроектувати рівні 2D платформера.

У ході всіх перелічених вище дій буде отримано повноцінний 2D платформер, що буде відповідати потребам цільової аудиторії користувачів.

РОЗДІЛ 2
АНАЛІЗ ТА ВИБІР ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТА ТЕХНОЛОГІЙ СТВОРЕННЯ КОМП'ЮТЕРНОЇ ГРИ "JUMP AND MOVE" В ЖАНРІ 2D ПЛАТФОРМЕР

2.1 Огляд сучасних ігрових рушіїв

Сучасна ігрова індустрія вражає своїм розмаїттям і кількістю ігрових рушіїв, які використовуються для розробки відеоігор. Ігровий двигун представляє собою комплексну програмну систему, яка надає розробникам інструменти для створення ігор.

Головна мета ігрового рушія полягає в спрощенні процесу розробки ігор та забезпеченні потужних функцій і можливостей для розробників. Він включає в себе різноманітні інструменти для моделювання, створення анімації, модулі фізики, штучного інтелекту та інші компоненти, які значно полегшують створення ігрових світів, персонажів та геймплею.

У даній роботі будуть розглянуті такі сучасні ігрові рушії, як Unreal Engine, Unity та Godot. Вони відомі своєю потужністю та гнучкістю, адже забезпечують розробників інструментами та ресурсами для створення різноманітних ігрових проектів. Вони використовуються як професійними студіями, так і незалежними розробниками, що свідчить про їхню актуальність та ефективність.

Рисунок 2.1 – Емблема рушія Unreal Engine

Unreal Engine - це потужний інтегрований ігровий двигун, розроблений і підтримуваний компанією Epic Games. Вперше представлений у 1998 році, UE став одним з найпопулярніших рушіїв і знайшов застосування в широкому спектрі проектів, від відеоігор до віртуальної реальності, архітектурних візуалізацій та тренування інтелектуальних агентів [4].

Двигун пропонує розробникам широкі можливості для створення вражаючого ігрового світу. Цей двигун має вбудовану сумісність з різними платформами, такими як персональні комп'ютери, консолі, мобільні пристрої та віртуальна реальність. Він надає можливість використовувати високоякісну графіку та реалістичну фізику, що дозволяє розробникам створювати деталізовані світи зі змінним освітленням, водою, частинками та іншими ефектами.

Даний рушій, як і багато інших аналогів, має свою систему версій, які відображають прогрес та розвиток цього двигуна. Ось кілька ключових версій:

- Unreal Engine 1: Була випущена у 1998 році і спрямована в основному на розробку ігор для персональних комп'ютерів того часу. Вона мала вражаючі графічні можливості та багато інноваційних функцій для свого часу.
- Unreal Engine 2: Була випущена у 2002 році і включала в себе значні поліпшення в графіці та фізичному моделюванні.
- Unreal Engine 3: Вийшла у 2006 році і стала однією з найпопулярніших версій. Вона отримала значні вдосконалення графічного двигуна, включаючи підтримку тривимірних моделей персонажів, освітлення, тіней, фізики та інших ефектів.
- Unreal Engine 4: Випущена у 2014 році, ця версія принесла значні поліпшення у графіці, засновані на фізичному моделюванні, включаючи підтримку реалістичного освітлення та матеріалів. UE4 також вводить систему Blueprints, що дозволяє розробникам без досвіду програмування створювати ігрову логіку за допомогою візуального програмування.
- Unreal Engine 5: Випущена в 2021 році, ця версія ігрового рушія відзначається значними покращеннями в графічній частині та продуктивності. UE

5 впроваджує нову технологію "Nanite", яка дозволяє розробникам працювати з безліччю високо деталізованих моделей без затримок у продуктивності. Крім того, введена система "Lumen", яка забезпечує реалістичне освітлення в реальному часі.

Ці версії UE представляють лише деякі ключові моменти в його історії. Epic Games продовжує активно розвивати та випускати оновлення для рушія.

Unreal Engine використовує мову C++ для програмування ігрових елементів. Використання цієї мови дає розробникам повний контроль над логікою гри, графікою, фізикою та іншими аспектами проекту.

Однією з переваг використання C++ є швидкодія. C++ є низькорівневою мовою програмування, що дозволяє ефективно використовувати ресурси та оптимізувати продуктивність гри. Розробники можуть писати власні оптимізовані алгоритми та керувати розподілом пам’яті, що робить Unreal Engine потужним інструментом для створення великих ігор з вражаючою графікою та широким функціоналом.

Для спрощення розробки Unreal Engine надає систему Blueprints, що з’явилась у UE4. Blueprints є візуальною системою програмування, яка дозволяє розробникам створювати логіку гри та взаємодію об’єктів без необхідності писати код.

Замість написання коду, Blueprints надає графічний інтерфейс, де розробники можуть візуально з’єднувати різні вузли, наприклад: вхідні та вихідні дані, математичні операції, умовні оператори, функції для створення потрібної логіки. Це дозволяє швидко і просто створювати складні системи поведінки, ігрові механіки, анімації та багато іншого.

Unreal Engine зробив значний вплив на індустрію відеоігор та інтерактивних додатків завдяки своїй потужності, гнучкості та простоті використання. Численні відомі проекти, такі як "Fortnite", "Gears of War", "Bioshock", і "Hellblade: Senua’s Sacrifice", використовують Unreal Engine як основу для своїх ігор, демонструючи його потенціал.

Рисунок 2.2 – гра "Fortnite"

Загалом, Unreal Engine є одним з найбільш сильних та впливових ігрових двигунів на ринку, що надає розробникам інструменти для створення вражаючих графічних та ігрових світів. Його висока функціональність, розширені можливості та активна спільнота розробників роблять Unreal Engine привабливим вибором для багатьох проектів.

Рисунок 2.3 – Емблема рушія Unity

Unity вперше випущений у 2005 році як рушій тільки для Mac OS X, а тепер працює у Windows та Linux, він також відомий своєю кросплатформеністю на всі мобільні, консольні, ПК-платформи і навіть AR/VR. Unity є потужним інструментом розробки, який пропонує широкий спектр функціональності та

можливостей. Він є найпопулярнішим двигуном у розробників ігор, причому як серед інді, так і більших студій [13].

Unity відрізняється своєю простотою в освоєнні та зручним інтерфейсом. Однією з великих переваг Unity є його низький поріг входження для новачків. Незалежно від рівня досвіду розробника, він може швидко освоїти цей інструмент та почати розробляти свої власні ігри.

Unity-розробники високо оцінюють зручність "drag and drop" інтерфейсу, що дозволяє легко перетягувати й розміщувати готові компоненти та об’єкти в сцену, прискорюючи процес розробки.

Крім того, Unity використовує мову програмування C#, яка є досить простою для освоєння, навіть для початківців. Синтаксис даної мови дозволяє легко виражати ідеї та реалізовувати функціонал ігри. Разом із зручним інтерфейсом Unity це робить процес розробки приємним та ефективним.

Unity також пропонує технологію Bolt для швидкого створення прототипів ігор. Bolt - це візуальний графічний інструмент, який дозволяє розробникам з’єднувати блоки логіки та налаштовувати їх поведінку без необхідності в програмуванні. Ця технологія дозволяє розробникам швидко експериментувати з ідеями, створювати складні системи та зосереджуватись на творчому процесі.

Unity є одним з найпопулярніших двигунів для розробки ігор і займає видатне місце у галузі мобільної розробки. За статистикою, близько половини мобільних ігор, доступних на ринку, створено саме з використанням Unity. Беззаперечно, цей двигун завоював собі довіру розробників завдяки своїй потужності та гнучкості. Unity, в першу чергу, орієнтований на інді-проекти та проекти середнього розміру, де він демонструє особливу ефективність.

Unity відомий своєю часткою в розробці численних популярних ігор, які стали справжніми хітами. Один з найвідоміших прикладів - "Pokémon GO", мобільна гра, яка стала справжнім феноменом, поєднуючи в собі доповнену реальність та популярну франшизу "Pokémon". Інша видатна гра, розроблена на Unity - "Genshin Impact". Ця захоплююча екшн-RPG отримала величезну увагу як

з боку гравців, так і з боку критиків завдяки своєму відкритому та цікавому світу. Крім того, варто згадати "Hearthstone: Heroes of Warcraft", відому карткову гру від компанії Blizzard Entertainment, яка також використовує Unity для свого легендарного проекту.

Рисунок 2.4 – Гра "Genshin Impact"

Крім своїх переваг, даний двигун має і свої недоліки. Unity - чудовий рушій для дрібних мобільних проектів, але для ААА-гри (triple-A, високобюджетні комп’ютерні ігри) це не найкращий інструмент. Ось чому:

- Продуктивність. Потрібно добре знати тонкощі розробки користувацького інтерфейсу, щоб зробити його продуктивним.
- Оптимізація. Кросплатформні та кросжанрові рушії мають меншу продуктивність порівняно з вузькоспрямованими рушіями. Це впливає на швидкість роботи, якість графіки та FPS (частота кадрів за секунду). Оптимальний діапазон кадрів для якісної картинки в грі складає 30-60 FPS. Щоб домогтися стабільного результату, розробники ігор рівня ААА створюють власні рушії під проекти.

- Немає шаблонів. Простий проект можна зібрати на поганому коді. Але щойно гра стає трохи складнішою, потрібна добре продумана архітектура, інакше її не вийде випустити.

Крім того, додатки, створені на Unity, досить "великовагові": навіть найпростіша піксельна гра може займати кілька сотень мегабайт на ПК. Так, для жорсткого диска комп'ютера це невеликий об'єм, але, якщо проект розробляється і для мобільних платформ, слід замислитися про оптимізацію його розміру.

Unity - це потужний інструмент, який часто використовується для розробки малих та середніх проектів з невеликою командою розробників. Він пропонує широкі можливості для швидкої розробки та прототипування ігор, що особливо корисно для інді-розробників та початківців. Незважаючи на деякі обмеження в продуктивності, функціональність все ще може задовольнити потреби простіших проектів. Проте, якщо потрібні ширші можливості, перехід на більш потужний двигун буде розумним рішенням.

Рисунок 2.5 – Емблема рушія Godot

Godot Engine - відкритий кросплатформний ігровий рушій для розробки 2D/3D-відеоігор і застосунків для ПК, мобільних пристроїв, веб-платформ. Адаптований до всіх поширених операційних систем, включно з Linux, macOS, Windows, Android та iOS.

Godot розробили 2007 року два програмісти з Аргентини - Хуан Лінецький і Аріел Манзур. Спочатку його використовували кілька ігрових студій Латинської Америки. У 2014 році розробники виклали рушій на GitHub за ліцензією MIT, що дозволяє розробникам мати повний доступ до вихідного коду та використовувати його безкоштовно для комерційних та проектів. У грудні того ж року вийшла перша стабільна версія 1.0. З цього моменту почався розвиток проекту і його поширення в інших країнах [10].

До основних можливостей Godot входить:

- Робота з 2D та 3D графікою - підтримка ефектів віддзеркалення, динамічних тіней, статичного і динамічного глобального освітлення, повноекранної обробки постфактум (засвічення, глибини різкості, гамма-корекції і т.д.).
- Підтримка реалістичної фізики - системи частинок (диму, туману, пари, вибухів тощо), властивостей динамічних і статичних тіл, зіткнень і руйнувань, трасування променів та інших фізичних процесів.
- Робота з анімацією - опції створення скелетної анімації, накладення об’єктів, кат-сцен у реальному часі.
- AR/VR - вбудований мобільний інтерфейс доповненої та віртуальної реальності з використанням спеціальних датчиків на телефоні.
- Процедурна генерація - автоматичне створення внутрішньоігрового контенту (оточення, NPC, об’єктів, зброї тощо) за допомогою алгоритмів.

Можливості Godot можна розширити за допомогою плагінів і сторонніх додатків. Наприклад, рушій дає змогу імпортувати сцени з графічних редакторів із налаштованим освітленням, камерами, фізикою зіткнень, анімованими персонажами тощо. Також є можливість підключати розроблені самостійно або сторонні плагіни віртуальної/доповненої реальності, кодеки для відтворення відео та аудіо, візуальні редактори.

Godot практикує підхід "все є сцена" і заточений під деревоподібну структуру елементів. На практиці це означає, що нові об’єкти на рівень додаються

як гілки до вже наявних на ньому вузлів, їх можна згортати в префаби та відкривати окремо. Таким чином дуже зручно редагувати всілякі збережені складові об’єкти. Єдиний момент - якщо на вашому рівні було виставлено світло, то, припустімо, зайшовши в локальну сцену персонажа, ви цього освітлення не побачите, а налаштовуючи його матеріали, не зрозумієте, який вигляд вони мають на головній сцені. Цю проблему можна розв’язувати по-різному, однак всі варіанти не надто зручні.

Godot - порівняно простий в освоєнні та використанні ігровий рушій. Працювати з ним можуть як досвідчені розробники, так і ентузіасти-початківці.

У Godot є власна адаптована мова програмування GDScript, що має схожий з Python синтаксис. Також є спеціальний модуль GDNative, що дає змогу використовувати код, написаний іншими мовами програмування як C# чи C++.

За бажанням можна працювати у системі візуального програмування VisualScript. Це вдалий варіант для новачків не тільки в геймдеві, а й у програмуванні загалом.

Godot створювався як двомірний ігровий рушій, 3D-редагування з’явилося пізніше. Розробники залишили для кожного режиму свій редактор, а не стали використовувати псевдо-двовимірний. Особливо це цінують розробники невеликих 2D інді-проектів.

До недоліків Godot можна віднести недостатнє опрацювання 3D. За цим параметром Godot поступається конкурентам на кшталт Unity або Unreal Engine.

Відкритий вихідний код, простота й особливості редактора Godot зробили рушій популярним у середовищі розробників інді-ігор. З відомих гейм-девелоперів, які використовували його у своїх проектах, - аргентинська компанія OCAM Studio і LRDGames, що випустила 2021 року сатиричну стратегію "Rogue State Revolution".

Рисунок 2.6 – Гра "Rogue State Revolution"

Godot популярний як навчальний посібник для навчання азам комп'ютерної графіки, програмування та геймдеву. Його часто використовують у спеціалізованих і загальноосвітніх навчальних закладах.

У сегменті ігор категорії АА та ААА ігровий рушій Godot Engine поки що не набув поширення. Але в останніх версіях почав розвиватися 3D-редактор, з'явилися нові функції та вбудовані інструменти для роботи з тривимірною високополігональною графікою.

2.2. Вибір оптимального рушія для розробки 2D платформера

Розглядаючи популярні ігрові рушії можна скласти основні критерії, які будуть ключовими у виборі серед трьох кандидатів. І одним з найважливіших критеріїв є ціна.

Одразу варто віддати належне Godot, як повністю безкоштовному двигуну, що поширюється за ліцензією MIT. Це в свою чергу означає, що розроблена гра або інший програмний продукт належить безпосередньо розробнику і нема ніяких обмежень у будь-якому вигляді [1].

Unreal Engine відповідно до стандартної ліцензійної угоди можна використовувати безкоштовно для навчання та розробки внутрішніх проектів. Крім того, можна розповсюджувати комерційні проекти без сплати жодної комісії Epic Games. Це включає спеціальні проекти для клієнтів, лінійний вміст, такий як фільми і телешоу, а також будь-який продукт, що не приносить доходу або чий дохід падає нижче порогу роялті.

Якщо ви розповсюджується готовий продукт, який містить код Unreal Engine, і валовий дохід від цього продукту за весь період перевищує 1 мільйон доларів США, тоді потрібно сплатити 5% роялті [4].

Ця модель дозволяє розробникам зменшити фінансові ризики, особливо під час ранньої стадії розробки, коли прибуток ще не є стабільним. Вона також надає мотивацію для компанії Epic Games, що стоїть за Unreal Engine, продовжувати підтримку і розвиток двигуна, оскільки їх дохід пов’язаний з успіхом розробників, які використовують Unreal Engine.

У Unity все працює наступним чином: якщо ваш оборот інвестицій на діяльність, пов’язану з вашим використанням Unity, за останні 12 місяців становив менше 100 тис. $ (або загальний обсяг обороту та інвестицій вашого бізнесу становить менше 100 тис. $), ви можете скористатися безкоштовною версією Unity Personal. Якщо ваш оборот інвестицій на діяльність, пов’язану з вашим використанням Unity, за останні 12 місяців склав більше 100 тис. $ але менше 200 тис. $ (або загальний обсяг обороту та інвестицій вашого бізнесу становить менше 200 тис. $), вам необхідно придбати Unity Plus вартістю 369$ в рік або 37$ на місяць за одне робоче місце. Також дана версія відрізняється кращими інструментами звіту про перебої та помилки. Unity Pro та Unity Enterprise призначені для компаній та для професійного використання. Unity Pro 1877$ на рік або 170$ на місяць за місце, а що стосується Unity Enterprise ціна є індивідуальною в залежності від певних чинників. Дані версії не передбачають фінансових обмежень для використання та мають доступ до закритих платформ PlayStation, Xbox та Nintendo Switch, інструменти розробки Unity Mars для AR,

компоненти Havok Physics. Окремою перевагою Unity Enterprise є доступ до вихідного коду та опція його редагування за додаткову плату, а також покращена клієнтська підтримка [3].

Слід зазначити, що важливим аспектом при порівнянні ігрових рішень є різноманітність підтримуваних мов програмування. Чим більше мов програмування доступно для використання, тим більше гнучкості та можливостей мають розробники при створенні ігор. Кожна мова програмування має свої унікальні особливості і синтаксис, які можуть бути більш або менш зрозумілими для розробників.

Unity. Мова програмування C# відзначається своєю доступністю та відносною простотою, що робить її привабливою для широкого кола розробників. Ця мова має логічний та зрозумілий синтаксис, що сприяє швидкому освоєнню та розробці ігор. Додатковою перевагою є наявність багатої документації, підручників та онлайн-ресурсів, які допомагають розробникам ефективно опановувати C# і реалізовувати свої ідеї в ігрових проектах.

Мова програмування C#, незважаючи на свою простоту, надає широкий спектр можливостей для розробки ігор. Вона пропонує потужну стандартну бібліотеку, яка включає різноманітні класи та функції для роботи з графікою, звуком, фізикою, мережевими з’єднаннями та багатьма іншими аспектами гри. Крім того, розробники мають доступ до сторонніх бібліотек та розширень, які дозволяють збільшити функціональність і гнучкість C# для реалізації будь-яких ідей у ваших ігрових проектах.

Unreal Engine. Використовує C++ як основну мову програмування. C++ є потужною мовою загального призначення, яка надає розробникам високу продуктивність і гнучкість для створення складних ігрових систем і досягнення високої швидкодії гри. Використання C++ у рушії Unreal Engine дозволяє глибоко налаштовувати функціональність, реалізовувати розширені ігрові механіки та використовувати низькорівневі можливості, які важливі для розробки великих і графічно потужних проектів.

Варто відзначити, що в порівнянні з іншими мовами програмування, C++ може вимагати додаткового зусилля та глибшого розуміння, особливо для новачків. Його складний синтаксис та потреба самостійно керувати пам’яттю можуть становити виклик для розробників, які не мають достатнього досвіду з цією мовою. Однак, знання C++ дозволяє досягти більшого контролю над процесами в грі та оптимізації її продуктивності. У результаті, Unreal Engine разом з C++ стає потужним інструментом для розробки великих, складних та вражаючих ігрових проектів.

Godot. Має власну мову програмування - GDScript. GDScript є простою та ефективною мовою, розробленою спеціально для Godot, яка дозволяє легко створювати ігрову логіку, керувати об’єктами та реалізовувати взаємодію в ігровому світі. Її зручний синтаксис робить її досить доступною для новачків. Однак, важливо відзначити, що GDScript має обмежену документацію порівняно з мовами, такими як C# або C++, і не має таких потужних можливостей.

Окрім GDScript, Godot також підтримує інші мови програмування, такі як C# та C++. Розробники можуть застосувати ці мови, використовуючи модулі розширення. Таким чином, Godot надає розробникам вибір між простотою та ефективністю GDScript або потужністю C# та C++, що робить Godot більш гнучким інструментом для розробки ігор.

Важливим показником при виборі ігрового рушія є його системні вимоги.

До найважливіших моментів слід віднести вимоги до центрального процесора, графічної карти, до об’єму оперативної пам’яті та вільного простору на диску.

Наша гра буде розроблятись на ноутбуці з наступними характеристиками:

- ОС: Windows 10 Home
- Процесор: шестиядерний AMD Ryzen 5 4600H (3.0 — 4.0 ГГц)
- Відеокарта: RTX 2060 6 ГБ
- Об’єм оперативної пам’яті: 16 ГБ
- Обсяг SSD – 720 ГБ

Вся інформація про системні вимоги рушіїв, що наведена у таблиці 2.1 взята з офіційних джерел розробників [4], [3], [4] і може змінюватися в залежності від масштабу та платформи проектів. Дані відповідають мінімально-рекомендованим системним вимогам для платформи Windows.

Таблиця 2.1 - Системні вимоги до рушіїв

Назва рушія	Системні вимоги				
	Операційна система	Процесор	Оперативна пам’ять	Графічна карта	Місце на диску
UE 5	Windows 10 64-bit версія або вище	Чотириядерний Intel або AMD, 2,5 ГГц або швидше	8 ГБ оперативної пам’яті	Підтримка DirectX 11 або 12	Від 60 ГБ вільного простору на диску
Unity	Windows 7 (SP1+), Windows 10 і Windows 11, лише 64-розрядні версії	Архітектура X64 з підтримкою набору інструкцій SSE2	8 ГБ оперативної пам’яті	Підтримка Однієї з версій DX10, DX11, DX12	Від 8 ГБ вільного простору на диску
Godot	Windows 10	Двоядерний Intel або AMD, 2 ГГц або швидше	8 ГБ оперативної пам’яті	Підтримка OpenGL 3.3 абоVulkan 1.2	Від 500 МБ вільного простору на диску

У своїх офіційних джерелах розробники пишуть що, системні вимоги до ігрових рушіїв можуть відрізнятися в залежності від проекту та інших факторів. Тому сказати, щось однозначно доволі складно і треба відштовхуватись в першу чергу від сценаріїв використання рушія.

У порівнянні з конкурентами, системні вимоги Unreal Engine є дійсно вищими, а вільного місця на диску треба значно більше. Це пов’язано з тим, що Unreal Engine надає широкий спектр функцій і можливостей для розробки

масштабних ігор. Висока якість графіки, фізичного моделювання та інших ефектів вимагають більш потужного обладнання для оптимальної продуктивності.

З іншого боку, рушій Godot, який відомий своєю легкістю, може вимагати довантаження додаткових компонентів для певних функцій.

Середовище Unity займає не так багато місця, але не варто забувати, що його проекти є досить великими в незалежності від їх складності.

Враховуючи порівняння системних вимог до трьох рушіїв, можна зробити висновок, що наша система відповідає мінімальним вимогам кожного з них. Це означає, що вона буде достатньо потужною для запуску і роботи з будь-яким з обраних рушіїв. Проте, важливо враховувати, що продуктивність та якість роботи рушія можуть варіюватися в залежності від апаратних можливостей системи та складності самого проекту.

Наявність великої спільноти розробників та документації є важливим аспектом при виборі ігрового рушія. Така підтримка дозволяє розробникам отримати доступ до розширених знань, досвіду та ресурсів, які можуть значно полегшити процес розробки ігор. Велике ком’юніті створює сприятливе середовище для обміну ідеями, питаннями та рішеннями. Доступна та добре організована документація дозволяє швидко засвоювати концепції та функціональні можливості рушія, а також ефективно його використовувати. Все це сприяє підвищенню продуктивності, зменшенню кількості часу на вирішення проблем і покращенню якості розроблених ігор.

Unity та Unreal Engine мають активні та великі спільноти розробників, які надають допомогу один одному. Однак, вони можуть відрізнятися за своєю спрямованістю та рівнем документації.

Unity має досить широку спільноту розробників. Існує велика кількість форумів, блогів, вебінарів, курсів та підручників, присвячених Unity. Це означає, що можна легко знайти відповіді на питання, рішення для проблем та поради від інших розробників. Крім того, Unity є найпопулярнішим двигуном розробки ігор, тому для нього існує велика кількість документації спеціально для новачків, яка

детально пояснює основні концепції та використання різних компонентів двигуна [13].

У Unreal Engine також є велике та активне співтовариство. Існують різноманітні форуми, статті, книги та інші ресурси, де розробники можуть обговорювати свої проекти, ділитися знаннями та отримувати підтримку. Unreal Engine надає розширену офіційну документацію, яка охоплює багато аспектів розробки ігор, включаючи роботу з Blueprints, C++ та інші складові.

Варто відзначити, що Unity, як найпопулярніший двигун, має ще більший розповсюджений інструментарій, що робить його документацію більш доступною для новачків. Unreal Engine, з іншого боку, може бути більш націлений на досвідчених розробників, особливо з урахуванням використання мови програмування C++.

Godot також має активне та зростаюче співтовариство розробників, яке надає підтримку новачкам. Хоча спільнота Godot може не бути так широкою, як у Unity або Unreal Engine, вона все ж демонструє активний розвиток і привертає все більше уваги.

Однак, коли мова йде про документацію, Godot має меншу кількість офіційних матеріалів порівняно з конкурентами. Все ж, розробники зможуть знайти значну кількість посібників, уроків та прикладів коду на форумах, блогах та відкритих ресурсах, які сприятимуть вирішенню проблем і навчанню різних аспектів розробки в Godot.

Оцінка загального порогу входження відображає суб’єктивну оцінку складності кожного ігрового двигуна. В рамках цієї оцінки розглядаються такі критерії, як доступність навчальних ресурсів та простота мови програмування. При оцінці використовується десятибальна шкала, де 1 означає найнижчий рівень складності, а 10 - найвищий рівень складності.

Згідно нашого аналізу, Unity отримує низьку оцінку відносно загального порогу входження. Даний рушій має велике активне ком’юніті, яке готове надавати підтримку та допомогу, а також багату документацію, включаючи

офіційні ресурси, підручники та відеоуроки. Мова програмування C#, яку використовує Unity, вважається простою для вивчення та розуміння, що полегшує розробку ігор. Крім того, доступність навчальних ресурсів, онлайн-курсів та спільноти робить Unity привабливим вибором для новачків у галузі геймдеву.

Unreal Engine, з іншого боку, має трохи вищий поріг входження. Хоча він також має значне ком'юніті і багато ресурсів документації, проте його мова програмування, C++, вважається більш складною для багатьох початківців. Розробка ігор на Unreal Engine може вимагати більше знань і досвіду в програмуванні, що може бути викликом для новачків. Однак, для тих, хто має досвід роботи з C++ та потребує більш просунутих функцій та можливостей, Unreal Engine може бути потужним інструментом для розробки масштабних проектів.

Godot, зі свого боку, також має низький поріг входження. Однак, ком'юніті Godot ще не таке розвинуте, а документація може бути менш об'ємною порівняно з Unity та Unreal Engine. Втім, Godot також набирає популярність, і з часом можна очікувати зростання кількості ресурсів та підтримки для цього двигуна.

Unity є найбільш популярним середовищем розробки і має значну кількість документації. Відносно проста мова програмування C# робить його досить доступним для новачків, і поріг входження оцінюється на рівні 5 з 10.

Unreal Engine, з свого боку, пропонує велику кількість посібників і ресурсів, але через складніший інтерфейс та використання мови програмування C++, яка вимагає керування пам'яттю, поріг входження оцінюється на рівні 8 з 10.

Godot, зокрема, вирізняється своєю простотою використання, але недостатня кількість документації та необхідність вивчення спеціальної мови програмування GDScript роблять його поріг входження на рівні 6 з 10.

Проаналізувавши вартість, підтримувані мови програмування, наявність великої спільноти та документації, а також системні вимоги, можна зробити висновок, що Unity виявляється кращим варіантом для створення 2D платформера. Розглянемо приведену таблицю порівнянь 2.2.

Таблиця 2.2 – Порівняння рушіїв

Назва двигуна	Наявність безкоштовної версії	Підтримувані мови програмування	Наявність великої спільноти та обширеної документації	Сумісність системних вимог	Загальний поріг входження
UE	Безкоштовна версія при валовому заробітку до 1 млн. $ за весь час	C++	+	+	8/10
Unity	Безкоштовна версія, якщо оборот або обсяг залучених інвестицій не перевищує 100 тис. $ за останні 12 місяців	C#	+	+	5/10
Godot	Повністю безкоштовний	GDScript C# C++	-	+	6/10

Unity має безкоштовну версію, яка надає значну функціональність для створення ігор. Велике ком’юніті Unity забезпечує доступ до безлічі ресурсів, включаючи готові рішення, плагіни, статті та підтримку від інших розробників. Це допомагає знайти відповіді на питання та розв’язати проблеми швидко та ефективно.

Unity також підтримує мову програмування C#, яка є досить простою для освоєння навіть початківцям. Це дозволяє швидко створювати ігрову логіку, візуальні ефекти та анімацію для платформера.

Наш ноутбук цілком відповідає системним вимогам Unity, що дозволить комфортно працювати в середовищі навіть при більш складних проектах.

Отже, з урахуванням усіх цих факторів, Unity є оптимальним варіантом для розробки 2D платформера. Він надає зручний інтерфейс, потужні інструменти та велику підтримку спільноти, що допоможе здійснити нашу творчу ідею.

2.3 Аналіз та вибір інтегрованого середовища розробки для мови програмування C#

У даному розділі ми зосередимося на аналізі та виборі інтегрованого середовища розробки для мови програмування C#. Зокрема, ми розглянемо такі популярні IDE: Rider, Visual Studio та Visual Studio Code. Кожне з цих середовищ має свої особливості та переваги, які потрібно проаналізувати.

Рисунок 2.7 – Емблема IDE Rider

Rider - це інтегроване середовище розробки, створене компанією JetBrains. Воно призначене для розробки на мові C# і володіє великим набором функцій, які полегшують процес розробки в Unity.

Одна з основних переваг Rider полягає в його потужних можливостях автодоповнення коду, аналізу та рефакторингу. Редактор має інтелектуальний режим автодоповнення, який допомагає швидко писати код і запобігає

допущенню помилок. Rider також надає велику кількість інструментів для аналізу коду, включаючи можливість перевірки стильових правил і виявлення потенційних проблем.

Щодо інших особливостей, Rider має інтеграцію з Git, що дозволяє легко працювати з контролем версій. Він також підтримує відладку Unity-проектів, включаючи можливість встановлення точок зупинки, перегляду значень змінних у режимі реального часу та інше.

Rider пропонує кілька версій підписки, починаючи з найдешевшої базової вартістю 178$, закінчуючи різними корпоративними варіантами. Однак є безкоштовні підписки для студентів акредитованих навчальних закладів. Для отримання безкоштовної ліцензії, необхідно надіслати заявку з електронної пошти навчального закладу або пред’явити міжнародну карту студента [2]. Також варто зазначити, що точного часу розгляду заявки немає і взагалі це доволі незручний варіант.

Загалом, Rider є потужним інструментом для розробки на мові C# у середовищі Unity. Його функціональність, продуктивність та підтримка заслуговують на увагу розробників.

Рисунок 2.8 - Емблема IDE Visual Studio

Visual Studio - це інтегроване середовище розробки, створене компанією Microsoft. Воно є одним з найпопулярніших інструментів для розробки програмного забезпечення, включаючи розробку на мові C# у середовищі Unity.

Середовище Visual Studio надає широкий набір функцій, що полегшують процес розробки. Воно має потужні можливості автодоповнення коду, аналізу помилок, вбудовану документацію та інструменти для рефакторингу коду.

Редактор також має велику кількість плагінів і розширень, які дозволяють налаштувати середовище під власні потреби розробника.

Однією з ключових особливостей VS є його інтеграція з екосистемою Microsoft. Воно підтримує широкий спектр інструментів, бібліотек та сервісів, таких як Azure, Git, Team Foundation Server та багато інших. Це робить його потужним інструментом для розробки проектів на основі платформи Microsoft.

Одна з основних переваг Visual Studio для Unity - це підказки та автодоповнення коду, спеціально налаштовані для роботи з API Unity. Вони допомагають швидше та зручніше розробляти функціонал, використовуючи наявні в Unity класи, методи та властивості.

Крім того, Visual Studio надає можливість відлагодження скриптів Unity. Розробники можуть встановлювати точки зупинки, аналізувати стек викликів, переглядати значення змінних та використовувати інші інструменти для з’ясування причин помилок чи неправильної роботи коду.

Щодо вартості, Visual Studio має кілька різних варіантів ліцензування. Існує безкоштовна версія Visual Studio Community, яка призначена для некомерційного використання та навчання. Крім того, є платні версії, такі як Visual Studio Professional та Visual Studio Enterprise, які мають розширені функції та можливості для комерційного використання [5].

Загалом, Visual Studio є потужним та розширюваним інструментом для розробки на мові C# у середовищі Unity. Його інтеграція з екосистемою Microsoft, розширені можливості редагування коду та інструменти для відладки роблять його популярним вибором серед розробників.

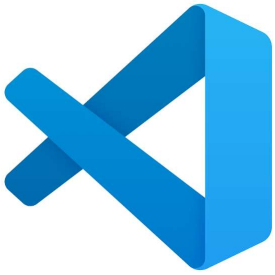

Рисунок 2.9 - Емблема IDE Visual Studio Code

Visual Studio Code є редактором коду, який можна розширити до певного рівня функціональності IDE. Він є відкритим та безкоштовним інструментом, розробленим компанією Microsoft.

Однією з переваг VS Code є його гнучкість та розширюваність. Для роботи з даною мовою програмування потрібно встановити розширення C# для VS Code, яке надає підтримку синтаксису, автодоповнення, перевірку помилок та інші функції, спеціально налаштовані для розробки на C#. Також можна встановити розширення для роботи з Unity, яке надає додаткові можливості для розробки, включаючи функції, які допомагають у редагуванні сцен, скриптів та інших компонентів [5].

Однак, варто відзначити, що VS Code не має всіх функцій інтегрованої розробки, які присутні в повноцінній версії Visual Studio. Наприклад, він може бути менш потужним у налаштуванні та підтримці спеціалізованих інструментів для Unity. Тому, якщо потрібна повна функціональність IDE, варто розглянути використання Visual Studio або інших рішень.

У будь-якому випадку, VS Code є популярним вибором серед розробників завдяки своїй універсальності, розширюваності та активній спільноті розробників, яка підтримує велику кількість розширень та плагінів для роботи з різними мовами програмування та фреймворками.

Після аналізу різних IDE для розробки на C# у середовищі Unity, можна зробити певні висновки.

Rider є потужним інструментом для розробки і не поступається аналогічному по функціоналу Visual Studio. Однак студенти, що потребують безкоштовної версії Rider, можуть відчувати проблеми з отриманням студентської ліцензії, оскільки це вимагає певного часу на узгодження.

Незважаючи на розширюваність та популярність Visual Studio Code не є повноцінною IDE. Її використання для роботи з Unity вимагає додаткових налаштувань та встановлення розширень, щоб забезпечити необхідну функціональність. Це займає досить багато часу та потребує певного досвіду

оскільки потрібно знати всі необхідні компоненти, що стануть в нагоді під час розробки [12].

Натомість, Visual Studio, зокрема його безкоштовна версія, є широко визнаною та добре адаптованою IDE для розробки в середовищі Unity. Вона надає повну підтримку C#, забезпечує зручні інструменти для роботи з Unity-проектами, включає у себе всі необхідні інтегровані ресурси та допоміжні функції, що спрощують процес розробки. Це робить Visual Studio привабливим вибором для розробників, особливо тих, хто працює з Unity.

Отже, враховуючи зручність, швидкість роботи, повноту функціоналу та адаптованість під наш рушій, Visual Studio буде оптимальним вибором для розробки на C# у середовищі Unity.

РОЗДІЛ 3
РОЗРОБКА КОМП'ЮТЕРНОЇ ГРИ В ЖАНРІ 2D ПЛАТФОРМЕР З ВИКОРИСТАННЯМ СЕРЕДОВИЩА UNITY ТА МОВИ ПРОГРАМУВАННЯ C#

3.1 Створення концепції гри та вибір асетів для 2D платформера

Аналізуючи сучасні тенденції, планується реалізувати 2D платформер у яскравому стилі з барвистих асетів оточення та персонажів, а також з приємним звуковим дизайном. Основною механікою гри буде рух персонажа вперед, назад та можливість стрибати по платформам.

Додатково, будуть створені вороги та різні пастки, які ускладнюватимуть проходження рівнів. При цьому, персонаж зможе знищувати ворога, стрибнувши на його верхню частину колайдеру. Для досягнення різноманітності випробувань потрібно створити кілька типів ворогів та пасток, щоб гра проходила цікавіше.

Для стимуляції дослідження та більшої взаємодії з небезпечними ділянками рівня додаємо можливість збирати предмети. У верху екрану гри розмістимо відображення кількості зібраних предметів, а якщо гравець збере усі, то в кінці рівня його привітають веселою мелодією. Це збагатить геймплей та надасть гравцеві відчуття досягнення, що є обов'язковим та невід'ємним елементом будь-якої гри.

Планується створити п'ять рівнів у трьох різних стилях, кожен з яких матиме свої унікальні плитки текстур та музичний супровід. Таким чином буде досягнуто ефекту прогресу та зміни.

Кожен новий рівень буде включати появу нових ворогів та складніших пасток, що створить зростаючий рівень виклику для гравця. Це забезпечить поступове підвищення складності гри і збереже захоплення та інтерес гравця протягом усієї гри.

Щоб гравець у випадку смерті не відправлявся на початкову точку рівня,

необхідно реалізувати механіку точок збереження. Це допоможе уникнути неприємних емоцій користувача, пов’язаних з потребою кожен раз починати все з початку.

Важливо грамотно продумати графічний інтерфейс. Одним із елементів, які потрібно розробити є сцена головного меню, яке містить кнопки, що ведуть до початку гри, виходу з неї та вимкнення музичного супроводу. Крім того, важливо розробити меню завершення гри, що дозволяє гравцю повернутися до головного меню та завершити гру. У самому процесі гри, важливо надати можливість поставити гру на паузу, вимкнути музику, повернутися до головного меню або навіть вийти з гри, забезпечуючи комфорт для гравця.

Окремою задачею є створення виконуваного "exe" файлу для нашої гри. Цей файл дозволить гравцям запускати гру без необхідності встановлення середовища Unity на своєму комп’ютері. Це забезпечить зручність та доступність для широкого кола користувачів.

Враховуючи ці елементи, наш 2D платформер буде мати цікавий геймплей, різноманітність у рівнях та ворогах, а також надаватиме гравцеві ті самі емоції та досвід, що дарують звичні та улюблені йому проекти.

Наступним кроком ми провели пошук оптимальних асетів у магазині Unity [6]. Розглядалися виключно безкоштовні варіанти, які б могли задовольнити наші потреби. Зокрема, завантажили асети під назвою "Pixel Adventure 1", які включають в себе асети середовища, головного героя, пасток та інші ігрові елементи, такі як коробки з призами та точки збереження. Також, завантажили асети під назвою "Pixel Adventure 2", які містять асети ворогів.

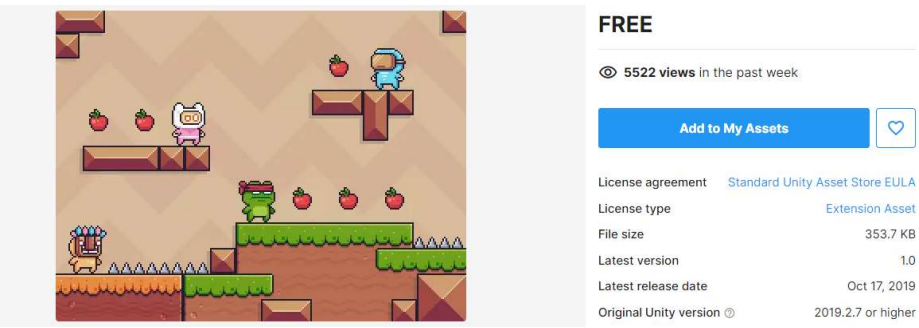

Рисунок 3.1 – Набір асетів

Для створення звукового супроводу нашої гри, ми також завантажили безкоштовні аудіофайли для звукових ефектів та мелодії для заднього фону. Ці звукові елементи ідеально підійдуть під настрій гри та додадуть хороших емоцій до проекту.

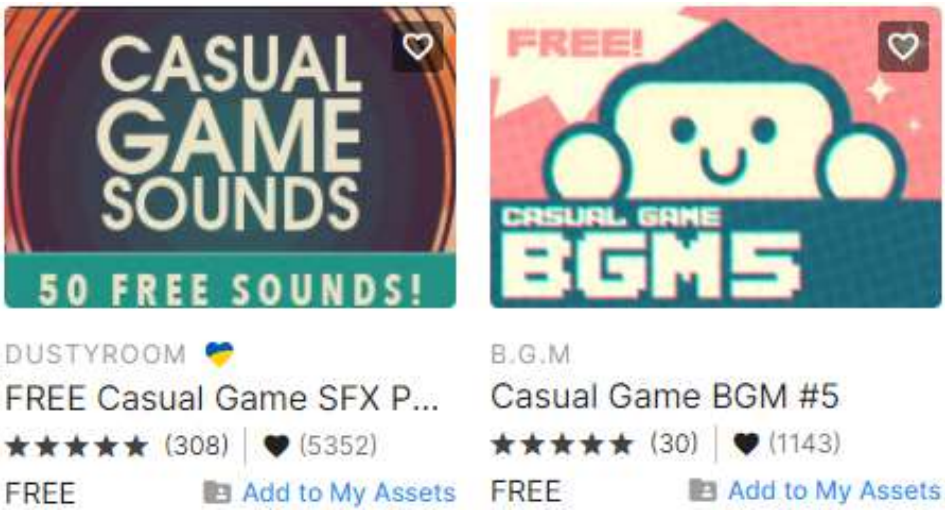

Рисунок 3.2 – Набір звукових файлів

Додатково завантажили набір елементів графічного інтерфейсу для створення кнопок меню. Підібрані графічні елементи будуть добре відповідати атмосфері нашого оточення.

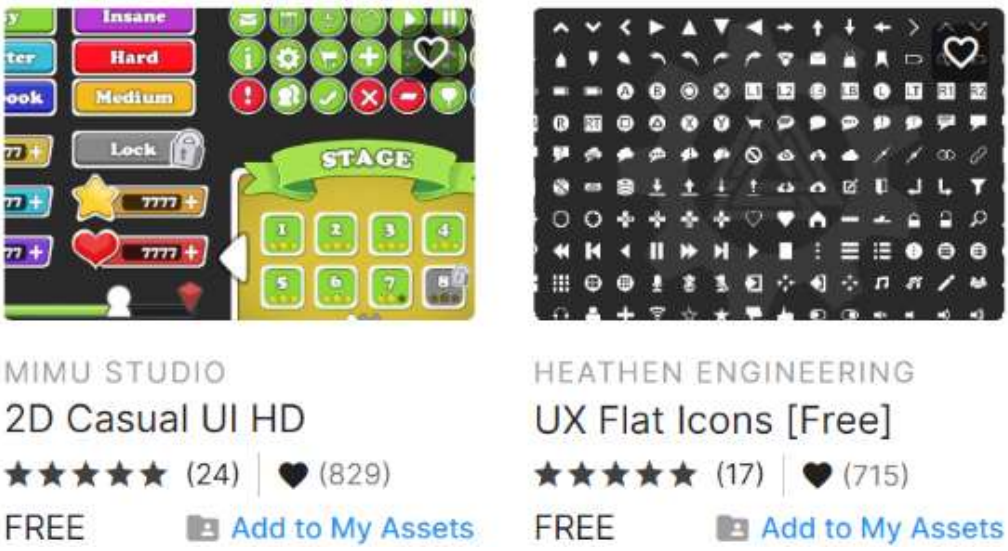

Рисунок 3.3 – Набір елементів графічного інтерфейсу

Органічно підібрані асети, графічні та аудіо файли дозволять реалізувати необхідний настрій та тон гри. Після того, як ми завантажили всі необхідні елементи можна переходити до розробки.

3.2 Програмування елементів гри мовою C#

Перед початком програмування елементів нашого 2D платформера, розробляємо блок-схеми для грамотного конструювання архітектури проекту. Перша блок-схема відобразить процес запуску гри, переключення між сценами, а також поведінку гравця з елементами графічного інтерфейсу, зокрема з функціями кнопок головного меню, меню паузи гри та кінцевого меню гри. Це дозволить нам чітко відстежувати логіку роботи окремих елементів і забезпечить більш глибоке розуміння їх взаємодії.

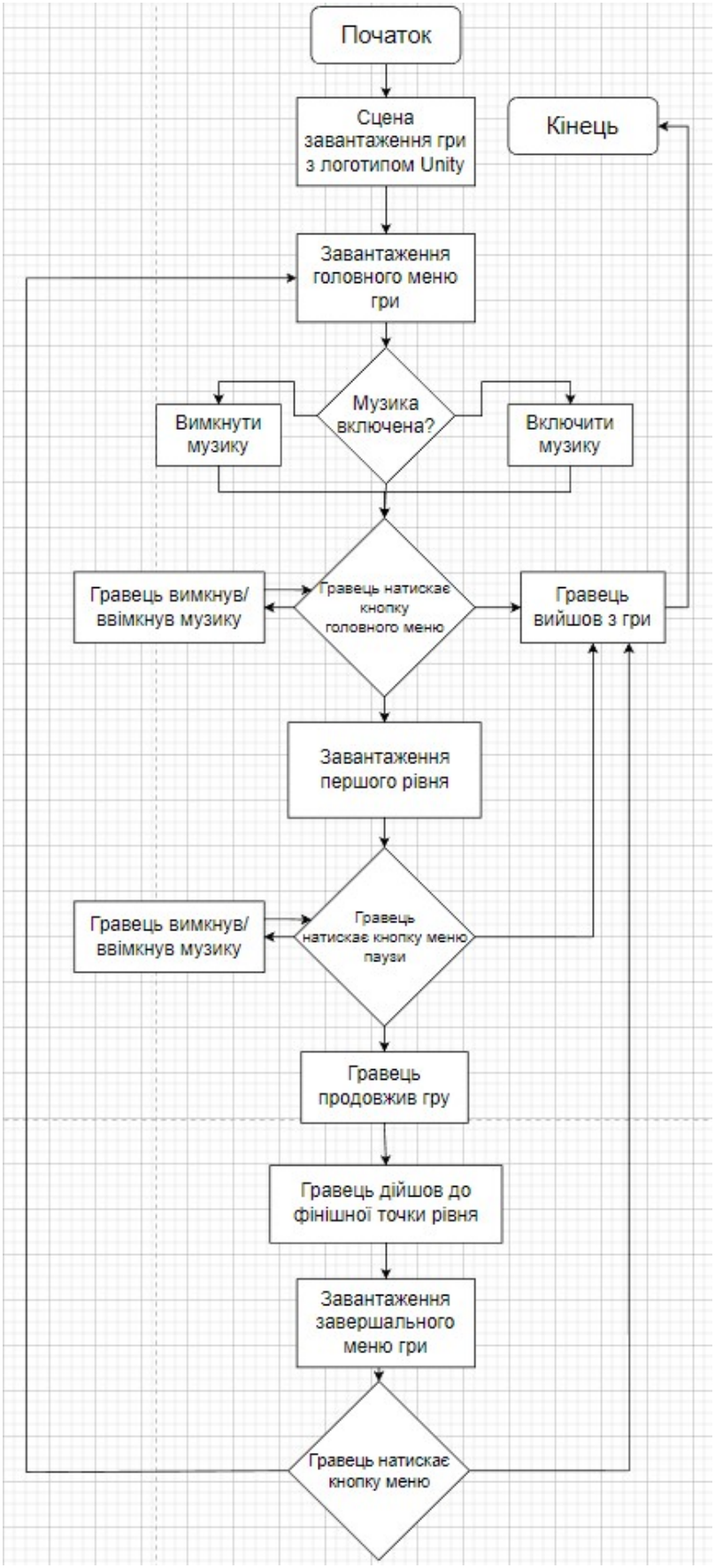

Рисунок 3.4 – Блок-схема запуску проекту гри та взаємодії з графічним інтерфейсом

Окремо розробили блок-схему для поведінки гравця на рівнях платформера. Тут відображені основні дії, які може виконувати гравець.

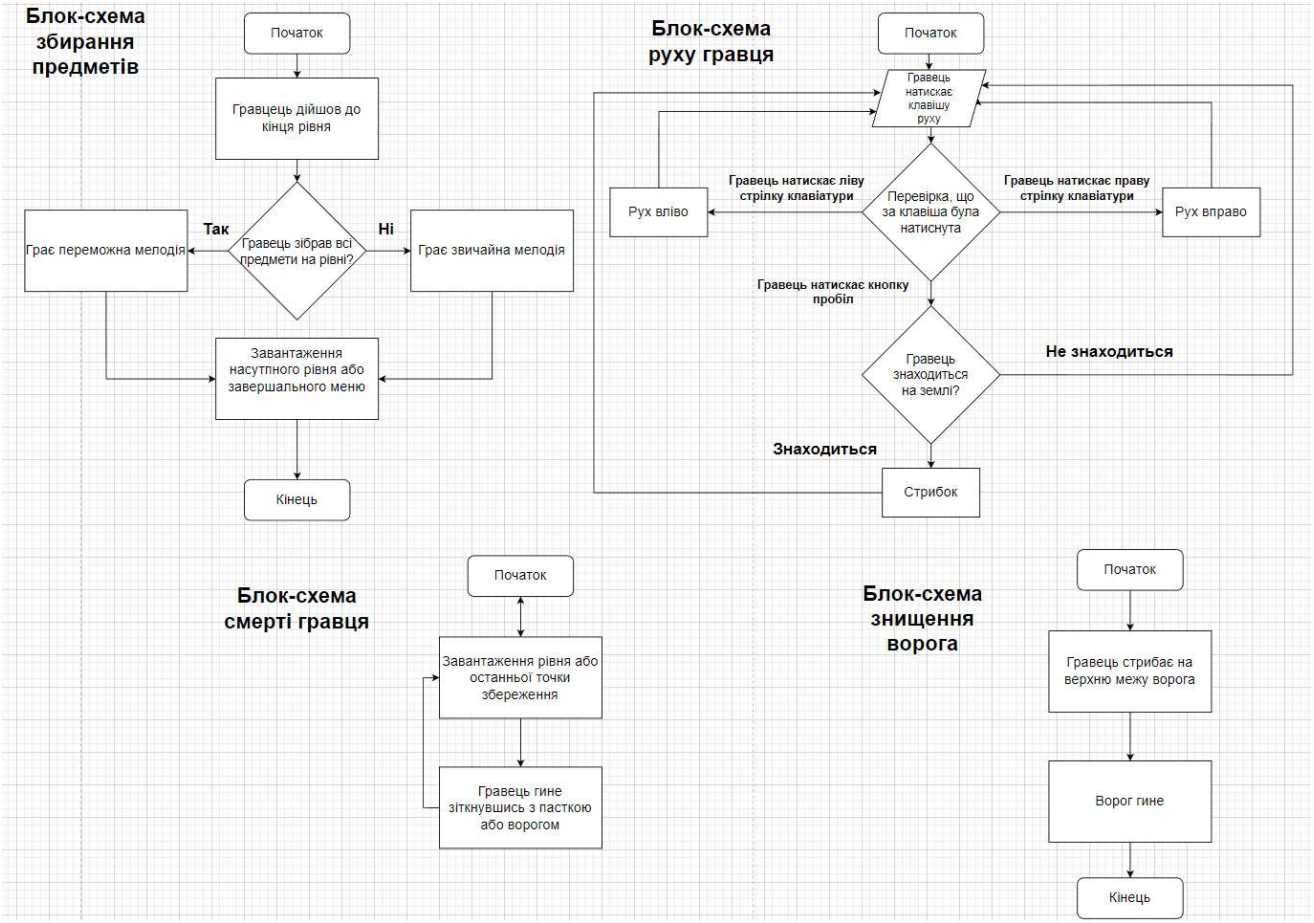

Рисунок 3.5 – Блок-схеми поведінки гравця

Після того як ми створили блок-схеми, процес програмування стане значно простішим. В першу чергу, необхідно реалізувати механіку руху гравця. Для досягнення цієї мети ми розпочинаємо зі створення невеликої ділянки, на якій зможемо протестувати його рух.

Спочатку відкриваємо папку з завантаженими плитковими картами оточення. Плиткові карти представляють собою графічні зображення, які використовуються для створення об’єктів у грі. Змінюємо їх розмір на 16 пікселів, щоб забезпечити дрібну деталізацію та переводимо режим спрайтів у режим множення. Це дозволяє нам обрізати наші плитки на більш дрібні частинки, використовуючи сітку розміром 16 на 16 комірок.

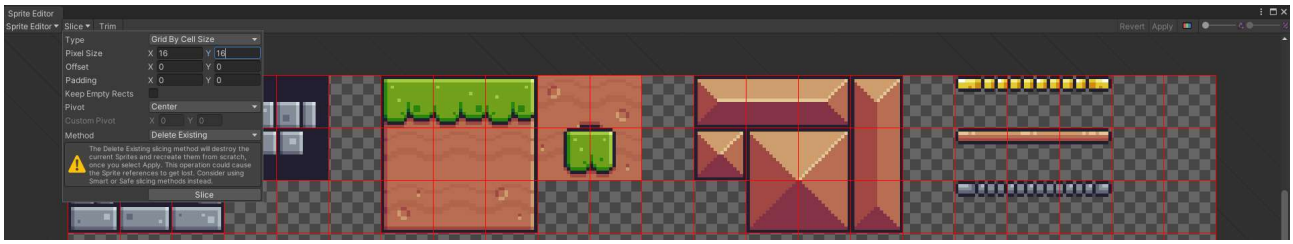

Рисунок 3.6 – Редактор спрайтів

Після цього кроку ми переходимо до ієрархії об’єктів нашого проекту та створюємо 2D сітку плиток у формі чотирикутника для наших платформ, яку називаємо "Terrain". Також створюємо ідентичний об’єкт для заднього фону і даємо йому назву "Background".

Далі створюємо сортуючі шари для кожного з цих об’єктів, щоб платформи знаходилися над заднім фоном. Це допоможе нам забезпечити правильний візуальний ефект у грі, де гравець буде розташовуватись на платформах, а задній фон буде відповідно позаду. Для цього переміщуємо "Background" на нульовий сортуючий шар, а "Terrain" на другий.

Крім того, переміщуємо об’єкт "Terrain" до окремого спеціального шару, який називаємо "Ground". Це має велике значення для майбутнього функціоналу гри, так як нам знадобиться перевірити, чи гравець торкається платформи.

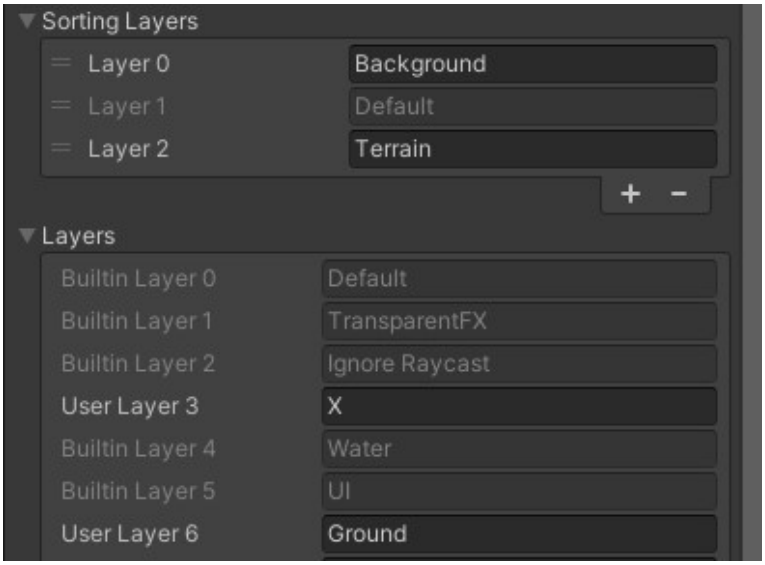

Рисунок 3.7 – Меню налаштування шарів

Після виконання цих кроків ми готові створити нашу першу платформу. Це буде частина "Terrain", на якій гравець зможе стояти та пересуватися.

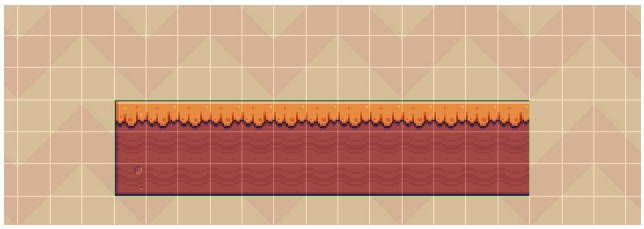

Рисунок 3.8 – Платформа

Для визначення зіткнень об’єктів з платформою ми використовуємо компонент "Tilemap Collider 2D". Цей модуль створює колайдер для кожної плитки у шарі "Terrain". Однак, це може негативно вплинути на продуктивність гри, оскільки кількість колайдерів може стати занадто великою [8].

Для оптимізації цього процесу ми додаємо компонент "Composite Collider 2D". Він дозволяє об’єднувати всі колайдери в один великий для усієї платформи, що спрощує обробку зіткнень у грі та поліпшує продуктивність.

Разом з компонентом "Composite Collider 2D" автоматично додається компонент "Rigidbody 2D". Важливо встановити властивість "Body type" цього компонента як "Static", щоб наші платформи залишалися нерухомими та не падали під впливом гравітації.

На наступному етапі ми створюємо гравця. Для цього в ієрархії об’єктів створюємо новий 2D спрайт і називаємо його "Player". Додаємо модель гравця до цього спрайту, яка буде відображати його вигляд у грі.

Для базової фізики гравця підключаємо вже знайомий компонент "Rigidbody 2D". Налаштовуємо такі параметри, як вага об’єкта, сила тяжіння та матеріал відповідно до наших потреб. Також встановлюємо параметр "Body Type" як "Dynamic". Це дозволяє об’єкту рухатись, взаємодіяти з іншими об’єктами, впливати на них і відповідно відчувати вплив на себе.

Для уникнення ефекту прилипання гравця до платформ створюємо новий матеріал. Встановлюємо значення тертя матеріалу на 0 і одразу кріпимо до

компонента "Rigidbody 2D" гравця. Це дозволяє гравцю рухатись плавно по платформам, уникнувши непотрібного тертя.

В додаток до вищезгаданих кроків, додаємо компонент "Box Collider 2D" для позначення меж моделі гравця. Цей модуль дозволяє визначити область, з якою можуть взаємодіяти інші об’єкти у грі.

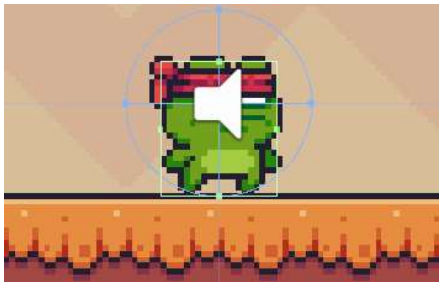

Рисунок 3.9 – Модель гравця

Після виконання цих кроків ми отримали об’єкт гравця з мінімальним набором необхідних фізичних компонентів, що готовий для програмування.

Створюємо новий скрипт для руху та перетягуємо його до об’єкта гравця.

У методі "Update" класу гравця реалізовуємо управління гравцем. На початку отримуємо вхідні дані від гравця щодо горизонтального руху (ліво або право). Далі встановлюю швидкість об’єкта по горизонталі в залежності від значення змінної "dirX" (ліво або право) та "moveSpeed" (швидкість руху).

У блоку перевірки з’ясовуємо чи натиснута клавіша стрибка "Jump" та чи знаходиться об’єкт на землі. Якщо обидві умови виконуються, об’єкту задається вертикальна швидкість, щоб здійснити стрибок.

```
void Update() {
    dirX = Input.GetAxisRaw("Horizontal");
    rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);
    if(Input.GetButtonDown("Jump") && IsGrounded()) {
        jumpSoundEffect.Play();
        rb.velocity = new Vector2(rb.velocity.x, jumpForce);
```

Рисунок 3.10 – Фрагмент скрипту руху гравця

Метод "IsGrounded" використовує променевий тест, що спускається вниз від об’єкта, щоб перевірити, чи знаходиться він на землі. Якщо промінь зіштовхується

з колайдером, який позначений шаром "jumpableGround", метод повертає значення "true", що означає, що об’єкт знаходиться на землі. В іншому випадку метод повертає значення "false". Цей метод був створений, щоб уникнути можливості безперервних стрибків гравцем, поки він не торкнеться землі.

```
private bool IsGrounded()
{ return Physics2D.BoxCast(coll.bounds.center, coll.bounds.size, 0f, Vector2.down, .1f, jumpableGround); }
```

Рисунок 3.11 –Метод перевірки колізії гравця з землею

Далі реалізовуємо механіку смерті гравця. Для цього створюємо окремий скрипт. Він перевіряє, чи сталася колізія гравця з об’єктом, який має тег "Trap" або "Enemies". У разі такої колізії сцена гри перезавантажується, а гравець телепортується на початок рівня.

```
private void OnCollisionEnter2D(Collision2D collision)
{ if (collision.gameObject.CompareTag("Trap") || collision.gameObject.CompareTag("Enemy"))
    { SceneManager.LoadScene(SceneManager.GetActiveScene().name); }}
```

Рисунок 3.12 – Фрагмент скрипту смерті гравця

Щоб гравець не переміщався на початок рівня у випадку смерті необхідно реалізувати механіку точок збереження. Для цього створюємо об’єкт точки збереження і два скрипти. Перший скрипт перевіряє зіткнення гравця з об’єктом і, якщо це перше досягнення даної точки збереження, записує її індекс у статичне поле другого скрипту з глобальними даними. У випадку смерті гравець буде переміщений на останню досягнуту точку збереження згідно його індексу.

```
void Awake()
{ player = GameObject.Find("Player").transform;
    if (DataContainer.checkpointIndex == index)
    { player.position = transform.position; }
}
Unity Message | 0 references
void OnTriggerEnter2D(Collider2D other)
{
    if(other.gameObject.tag == "Player")
    { if (index > DataContainer.checkpointIndex)
        { DataContainer.checkpointIndex = index; }
```

Рисунок 3.13 – Фрагмент скрипту точок збереження

Щоб грати в гру було ще веселіше, додаємо можливість збирати предмети. Для цього створюємо 2D об’єкт яблука і додаємо до нього компонент "Box Collider 2D", щоб виявляти зіткнення з гравцем. Також створюємо нове поле "appleCount" у скрипті з глобальними даними для підрахунку зібраних яблук. У цьому ж скрипті для уникнення повторного генерування зібраних яблук, створюємо статичний словник, куди будемо додавати ідентифікатор кожного зібраного яблука.

У скрипті яблука перевіряємо, чи воно не було зібране раніше. Якщо ні, збільшуємо загальну кількість яблук на 1, додаємо ідентифікатор у словник і знищуємо яблуко (перевірка створена, щоб не зібрати одне яблуко кілька разів поки відбувається вся логіка та знищення яблука). Якщо яблуко вже є в словнику, воно автоматично знищується при запуску сцени [7].

```
void Start()
{ if (DataContainer.CheckIfAppleCollected(idApple))
    { Destroy(gameObject); }
}
Unity Message | 0 references
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Player"))
    {   if (!DataContainer.CheckIfAppleCollected(idApple))
        {   DataContainer.appleCount++;
            DataContainer.AddAppleToDictionary(idApple, false);
            Destroy(gameObject, 0.7f);
```

Рисунок 3.14 – Фрагмент скрипту збору яблука

У верхньому лівому куті екрану ми створили поле з написом для відображення кількості зібраних яблук. Для цього в ієрархії об’єктів створили нове полотно та додали до нього елемент тексту. Через окремий скрипт в постійному режимі перевіряємо поточну кількість зібраних яблук з поля глобального класу.

Рисунок 3.15 – Текстове поле з кількістю зібраних яблук

Також ми додали фінішну точку для переходу на наступний рівень. Для цього ми скопіювали нашу сцену і додали її до менеджера сцен. Після цього створюємо новий об’єкт і перетягуємо до нього спрайт фінішної прямої.

При досягненні фінішної прямої спрацьовує скрипт, що переміщує гравця на наступний рівень, а рахунки зібраних яблук та досягнутих точок збереження очищуються.

```
private void CompleteLevel()
{   SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    DataContainer.ClearData();}
```

Рисунок 3.16 – Фрагмент скрипту переходу на наступний рівень

Наступним кроком запрограмуємо одного з основних ворогів персонажа носорога, що постійно патрулює свою ділянку, переміщуючись від початкової точки до кінцевої. У випадку їх зіткнення, гравець загине, проте у нього буде можливість застрибнути на верхню частину спрайту носорога для його знищення. Коли гравець потрапляє в поле зору носорога, швидкість ворога збільшується, щоб імітувати атаку тварини.

Для реалізації задуму ми створили спрайт 2D об’єкта з назвою "Rino" і надали йому модель носорога. Для нашого носорога ми також додали два компонента “Box Collider 2D”. Перший компонент використовується для визначення тіла носорога, а другий - для позначення межі, на яку гравець може застрибнути, щоб знищити ворога. В налаштуваннях другого компонента встановили прапорець "Is trigger", щоб він виконував певну дію при зіткненні з гравцем. Крім того, присвоїли спрайту тег "Enemy", що допомагає ідентифікувати його як ворожий об’єкт.

Додатково прикріпили порожній об’єкт, який буде відповідати за перевірку знаходження гравця на шляху носорога для збільшення його швидкості. Для цього ми знову скористалися компонентом "Box Collider 2D", встановивши йому значення "Is trigger" і розтягнули на довжину приблизної зони патрулювання. Також присвоїли цьому об’єкту нейтральний тег.

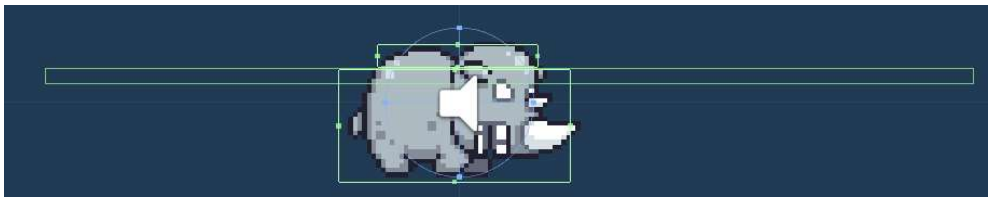

Рисунок 3.17 – Модель носорога з колайдерами

У класі носорога оголосили цілочисельні поля для зберігання значень поточного індексу шляху та швидкості руху спрайту носорога. Також створено масив ігрових об’єктів, який використовується для вказівки початкової та кінцевої точок патрулювання носорога.

В методі "Update" написали логіку, що забезпечує переміщення спрайту між двома точками та розвертання спрайту по осі X, коли досягнута будь-яка з цих точок, створюючи ефект ходьби вперед і назад.

```
private void Update(){
    if (Vector2.Distance(waypoints[currentWaypointIndex].transform.position, transform.position) < .1f)
    { currentWaypointIndex++;
        if (currentWaypointIndex >= waypoints.Length)
        {currentWaypointIndex = 0;}
        if (transform.position.x > waypoints[currentWaypointIndex].transform.position.x)
        {transform.localScale = new Vector3(-1, 1, 1);}
        else
        {transform.localScale = new Vector3(1, 1, 1);}
    }
    transform.position = Vector2.MoveTowards(transform.position,
        waypoints[currentWaypointIndex].transform.position, Time.deltaTime * speed);}
```

Рисунок 3.18 – Метод патрулювання ділянки

При зіткненні колайдеру гравця з верхнім колайдером ворога, спрацьовує метод "OnTriggerEnter2D". В цьому методі змінюється тег ворога на "EnemyHead", щоб одразу не знищити гравця та відразу викликається метод "EnemyDie", що знищує ворога після незначної затримки.

```
private void OnTriggerEnter2D(Collider2D collision)
{ if (collision.gameObject.tag == "Player")
    {   gameObject.tag = "EnemyHead";
        EnemyDie();}}
1 reference
private void EnemyDie()
{Destroy(gameObject, delay);}
```

Рисунок 3.19 – Фрагмент коду знищення носорога

Далі в методі "OnTriggerEnter2D" класу гравця виконується скрипт, що змушує гравця підстрибнути від носорога з тегом "EnemyHead", немов від батута. Це робить процес знищення ворогів трішки веселішим.

```
void OnCollisionEnter2D(Collision2D col)
{ if (col.collider.CompareTag("EnemyHead"))
    { Vector2 bounceForce = new Vector2(0, bounceStrength);
        rb.velocity = bounceForce; }
```

Рисунок 3.20 – Фрагмент коду з встановленням тригеру

Для пустого об’єкта, який відповідає за перевірку наявності гравця на шляху, було створено окремий скрипт. В ньому містяться значення стандартної та збільшеної швидкостей. Якщо відбувається колізія гравця з цим об’єктом, значення більшої швидкості передається до екземпляра носорога, а в іншому випадку швидкість повертається до стандартного значення.

```
public class RinoFind : MonoBehaviour
{
    [SerializeField] int speed1;
    [SerializeField] int speed2;
    private Rino rino;
    Unity Message | 0 references
    private void Start()
    { rino = transform.parent.GetComponent<Rino>();
      rino.speed= speed1;}
    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {if (collision.CompareTag("Player"))
        {rino.speed = speed2;}}
    Unity Message | 0 references
    private void OnTriggerExit2D(Collider2D collision)
    {if (collision.CompareTag("Player"))
        {rino.speed = speed1;}}}
```

Рисунок 3.21 – Фрагмент скрипту для пошуку гравця на шляху

Після збереження скриптів, нам потрібно створити два пусті об’єкти для позначення початкової та кінцевої точок патрулювання. Ми назвали їх "Waypoint1" і "Waypoint2" відповідно. Для кращого сприйняття додаємо їм візуальні іконки у вигляді кристалів.

Після цього залишається лише перетягнути ці об’єкти у компоненти скрипту носорога. В скрипті пустого об’єкта ми повинні вказати значення стандартної швидкості та збільшеної швидкостей.

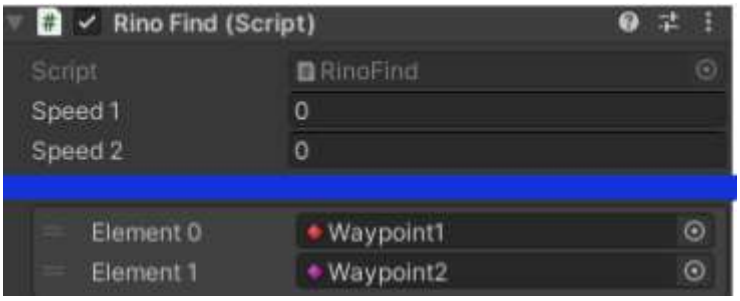

Рисунок 3.22 – Налаштовані поля значень класів

Також, з метою забезпечення комфорту гравця, була додана можливість призупинити гру за допомогою клавіші "Esc". Під час паузи гравець зможе повернутися до головного меню, вимкнути музичний супровід, вийти з гри або продовжити її.

Для реалізації цих функцій, на вже існуючому полотні під назвою "Canvas", створили панель "Pause_menu" та додали до неї чотири кнопки.

Рисунок 3.23 – Об’єкт полотна

Після цього розробили дизайн кнопок, використовуючи попередньо завантажені графічні елементи, з метою надання їм привабливого вигляду. Ретельно було підібрано розмір та стиль шрифту, щоб забезпечити гармонійний зовнішній вигляд кнопок та забезпечити зручне сприйняття інформації гравцем.

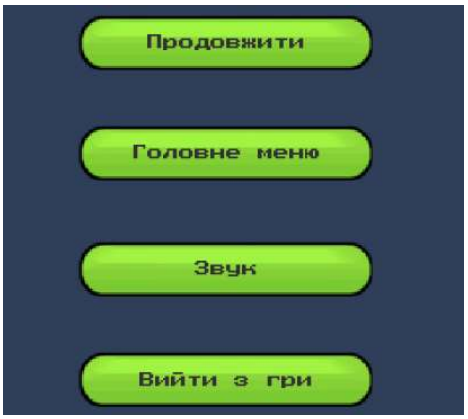

Рисунок 3.24 – Меню паузи

На даному етапі було запрограмовано логіку для кнопок паузи та продовження гри. Для цього ми визначили умову перевірки, яка спрацьовує при натисканні клавіші "Esc" під час гри. Якщо гравець натискає цю клавішу, виконується метод "Pause()", що призводить до відображення меню паузи і зупинки плину часу в грі. Таким чином, гравець може звернутися до різних опцій у меню паузи.

У разі, якщо гравець натискає кнопку продовження гри або повторно натискає "Esc", панель меню паузи приховується, а плин часу відновлюється, що дозволяє гравцю безперешкодно продовжити гру.

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {   if (GameIsPaused)
        {
            Resume();
        }
        else
        {
            Pause();
        }}}
1 reference
public void Resume()
{   pauseMenuUI.SetActive(false);
    Time.timeScale = 1f;
    GameIsPaused = false;}

1 reference
private void Pause()
{   pauseMenuUI.SetActive(true);
    Time.timeScale = 0f;
    GameIsPaused = true;}
```

Рисунок 3.25 – Фрагмент скрипту управління меню паузи

Така реалізація кнопок паузи та продовження гри забезпечує зручний та інтуїтивний спосіб керування грою і надає гравцеві можливість зупинити відлік часу у необхідний момент та повернутися до гри зручним для нього способом.

3.3. Створення анімацій та додавання звукових ефектів в середовищі Unity

Для створення анімації гравця враховуємо його можливі стани та переходи між ними. Загалом, нам потрібно п’ять анімацій: анімація спокою, анімація руху, анімація падіння, анімація стрибка та анімація смерті. Ми завантажили всі спрайти для анімацій, і можемо відтворити їх послідовно на доріжці анімації, щоб побачити реалістичний рух персонажа.

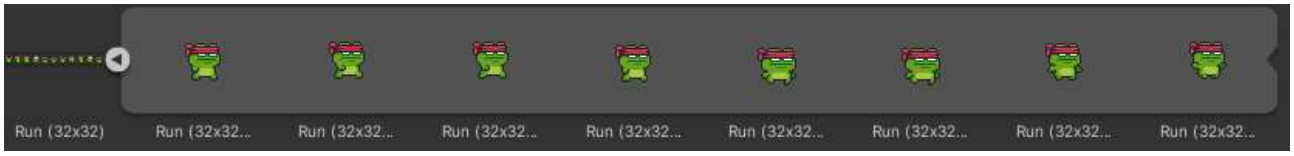

Рисунок 3.26 – Спрайти для анімацій

Після створення усіх п’яти анімацій, ми прикріплюємо їх до гравця і налаштовуємо такі параметри, як швидкість відтворення анімацій та інтервали між ними. Після додавання анімації до гравця з’являється компонент під назвою "Animator Controller". Відкривши побачимо, що всі наші анімації розташовані випадковим чином. Для забезпечення коректного переходу між анімаціями, розташовуємо зв’язки у правильній послідовності.

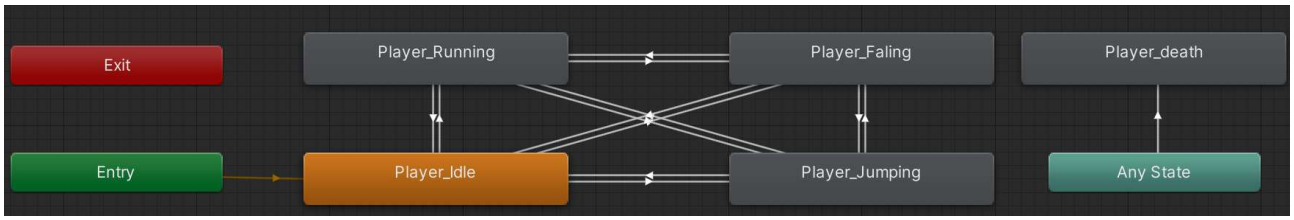

Рисунок 3.27 – Зв’язки анімацій гравця

Реалізовуємо логіку, виходячи з якої від кожного стану гравця можна миттєво перейти до іншого. Крім цього, окремо виділяємо стан смерті, який відбувається при виконанні певної умови. Щоб Unity розумів, коли саме треба використовувати та переключати анімації, потрібно запрограмувати спеціальні тригери або інші спеціальні змінні, які при зміні своїх значень запускають процес переходу до наступної анімації. В даному випадку, створюються тригер "death" (смерть) та цілочисельне значення "state" (стан), яке використовується для переключення між анімаціями руху та смерті.

Враховуючи, що у нас є 4 анімації руху, ми можемо призначити цілочисельні значення 0, 1, 2 та 3 кожній з них. Значення 0 відповідатиме стану спокою, 1 - бігу, 2 - стрибку, а 3 - падінню. Щоб цього досягти, нам потрібно в налаштуваннях зв’язків вказати умови переходу залежно від значення змінної "state". Залишається тільки реалізувати цю логіку в коді.

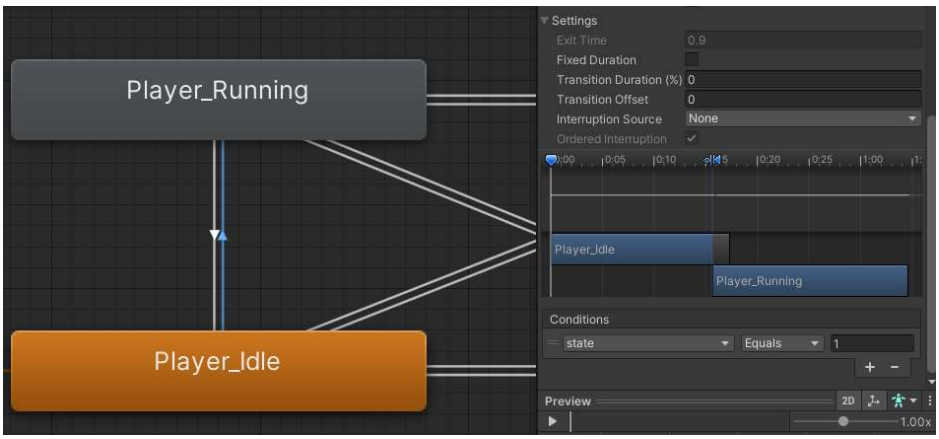

Рисунок 3.28 – Налаштування зв’язків анімацій

У скрипті руху гравця використовується перерахування (enum) під назвою "MovementState", яке визначає різні стани гравця, такі як "спокій" (idle), "біг" (running), "стрибок" (jumping) та "падіння" (falling).

```
private enum MovementState { idle, running, jumping, falling }
```

Рисунок 3.29 – Перерахування станів гравця

Створюємо змінну "state" типу "MovementState", яка визначає поточний стан руху гравця.

У наступних рядках коду перевіряється значення "dirX", яке представляє горизонтальний рух гравця. Якщо "dirX" більше 0, це означає, що гравець рухається вправо, тому стан встановлюється на "running", а також спрайт персонажа повертається і віддзеркалюється в необхідному напрямку по осі X. Аналогічно, якщо "dirX" менше 0, гравець рухається вліво, тому стан встановлюється на "running", а спрайт персонажа віддзеркалюється в зворотньому напрямку. Якщо "dirX" дорівнює 0, це означає, що гравець не рухається по горизонталі, тому стан встановлюється на "idle".

Далі по коду перевіряється вертикальна швидкість гравця "rb.velocity.y". Якщо вона більше 0.1, це означає, що гравець знаходиться у стані стрибка, тому стан встановлюється на "jumping". Якщо вертикальна швидкість менше -0.1, це означає, що гравець знаходиться у стані падіння, тому значення встановлюється на "falling".

За допомогою методу "anim.SetInteger("state", (int)state)" параметр "state" отримує приведене цілочисельне значення, яке відповідає переліку "MovementState". Таким чином, відповідна анімація відтворюється в залежності від поточного стану руху гравця.

```
private void UpdateAnimationUpdate()
{   MovementState state;
    if (dirX > 0)
    {   state = MovementState.running;
        sprite.flipX = false;}
    else if (dirX < 0f)
    {   state = MovementState.running;
        sprite.flipX= true;}
    else {   state = MovementState.idle;}
    if (rb.velocity.y > .1f)
    {state = MovementState.jumping;}
    else if (rb.velocity.y < -.1f)
    {state = MovementState.falling;}
    anim.SetInteger("state", (int)state);}
```

Рисунок 3.30 – Метод переключення анімацій

В методі, що відповідає за смерть гравця, пишемо рядок з наказом встановити значення тригеру на "death" активним.

```
private void Die()
{   rb.bodyType = RigidbodyType2D.Static;
    anim.SetTrigger("death");}
```

Рисунок 3.31 – Фрагмент коду з встановленням тригера

Наступним кроком буде анімація попередньо створеного ворога рослини, що стріляє по певній траєкторії зеленим ядром. Нам необхідно реалізувати анімацію спокою, анімацію пострілу, перехід між ними та анімацію смерті. При запуску сцени одразу відбувається анімація пострілу після чого перехід до стану спокою, поки куля не досягне кінцевої точки, після чого процес повторюється. Щоб реалізувати перехід від атаки до спокою, ми не вказуємо ніяких додаткових умов, а лише налаштовуємо автоматичний перехід до наступної анімації при закінченні поточної. А от для переходу з стану спокою в стрільбу має спрацювати попередньо створений тригер "Shoot".

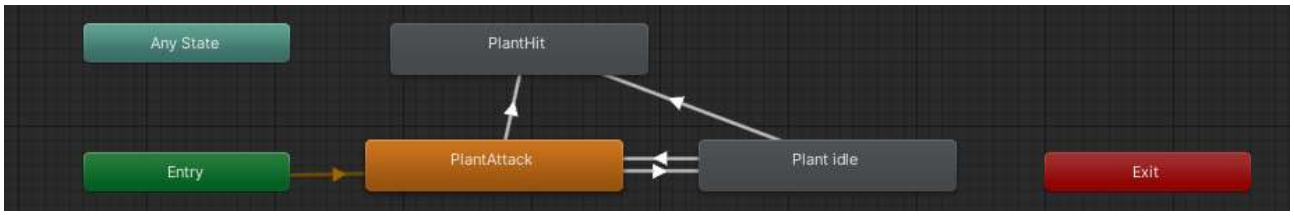

Рисунок 3.32 – Зв’язки анімацій ворога рослини

В блоці коду, що перевіряє чи дійшла куля до кінця, вставляємо рядок активації тригера. Таким чином, коли куля досягне кінцевої точки і знищиться, спрацює тригер і відбудеться анімація пострілу, після чого спрацює повернення до стану спокою.

```
if (Vector3.Distance(currentBullet.transform.position, endPoint.position) < 0.01f)
{   anim.SetTrigger("Shoot");
    Destroy(currentBullet);
    canShoot = true; }
```

Рисунок 3.33 – Фрагмент коду з встановленням тригера

При зіткненні гравця з верхньою частиною колайдера ворога, скрипт рослини вступає в дію і знищує її. В цей момент спрацьовує другий тригер під назвою "plantDead", який ініціює процес знищення рослини. Це призводить до плавної анімації, яка відображає, як рослина поступово зникає.

Рисунок 3.34 – Анімації ворога рослини

Для створення атмосфери додаємо музику на задній фон. Для цього створюємо новий об’єкт і додаємо до нього компонент "AudioSource". Потім перетягуємо обрану композицію до цього компонента та налаштовуємо опції для досягнення оптимального звучання.

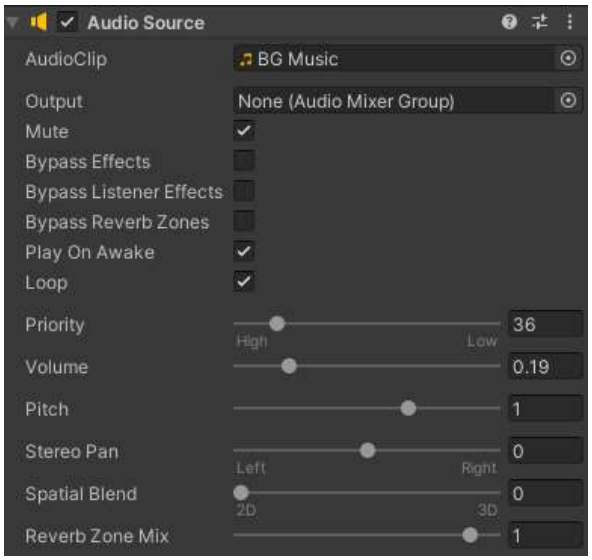

Рисунок 3.35 – Налаштування аудіо-компонента

Додатково додали музичні ефекти для різних подій у грі, щоб збагатити звукову атмосферу. Зокрема, встановили ефекти для відтворення звуків стрибка гравця, його загибелі, звуку збирання яблук, звуку стрибку від батуту та звуків знищення різних об’єктів та ворогів.

Коли ми встановили музичний фон та аудіоелементи, які відтворюються відповідно до виконаних дій та подій у грі, геймплей став більш захоплюючим і цікавим для гравця.

3.4. Проектування рівнів 2D платформера та створення "exe" файлу

Коли всі компоненти нашого 2D платформера готові, ми приступаємо до створення рівнів.

Перший рівень має вступний характер і складається з простих перешкод. Це зроблено з метою ознайомлення гравця з основними механіками гри, такими як ходьба, стрибки та збирання яблук.

Під час проходження першого рівня гравець зустріне кілька ворогів і навчиться їх знищувати, стрибаючи на їх голови. Він також зіткнеться з декількома пастками, включаючи небезпечне сховище з яблуками. Крім того, гравець буде перестрибувати над небезпечною ямою з шипами, щоб дістатись до кінця рівня. Для зручності гравця на цьому рівні розміщено дві точки збереження, розташовані на невеликій відстані один від одного, що допомагає новачкам легше освоїтись з грою.

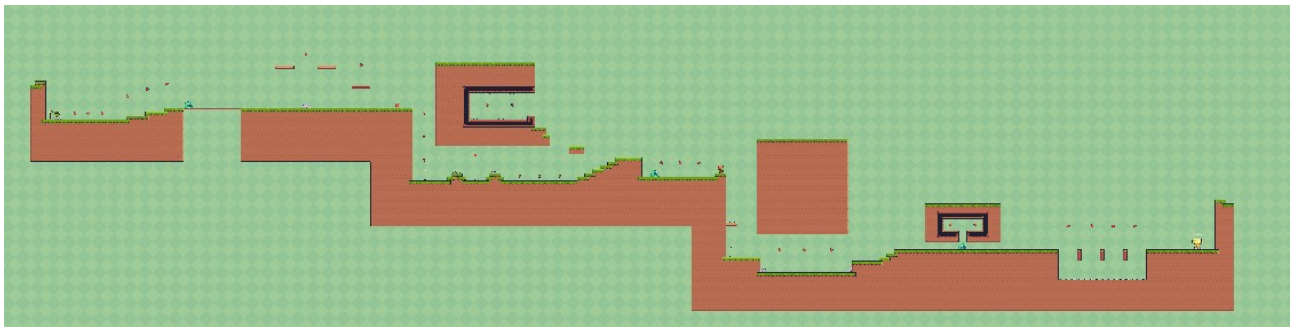

Рисунок 3.36 – Схема першого рівня

Другий рівень запропонує гравцю більший виклик. Тут він вперше зіткнеться з новим небезпечним та швидким ворогом - носорогом і вперше скористається батутом для досягнення вищих платформ. Крім того, гравця

чекатиме приховане місце з великою кількістю яблук, а також підказка, як знайти це місце.

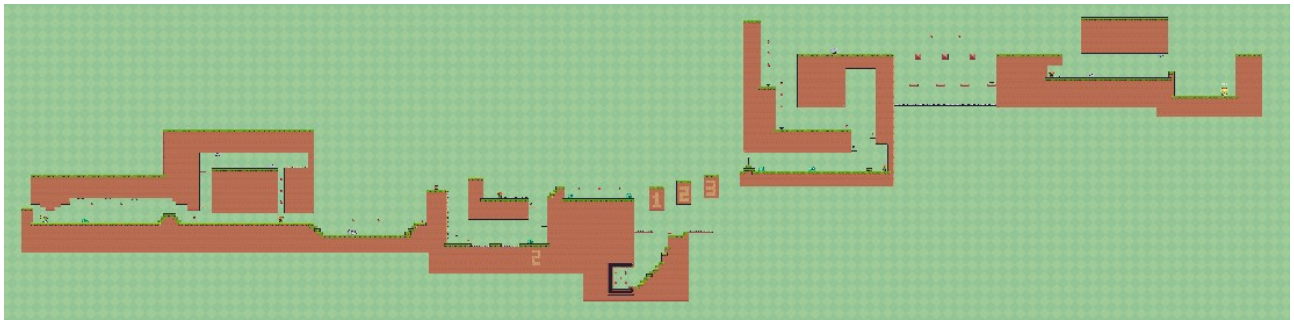

Рисунок 3.37 – Схема другого рівня

Третій рівень буде відрізнятися візуальним та звуковим оформленням. Тут гравець зустріне нових ворогів, таких як синя птаха, яка заважатиме руху між островами на рухомій платформі, а також свиней, які матимуть кілька життів, що ускладнює їх знищення. Крім того, будуть з’являтися черепахи з шипами, які періодично випускатимуть ці шипи, їх неможливо знищити, зате можна використовувати як батут. Кількість ворогів зросте, а пастки стануть небезпечнішими, створюючи більший виклик для гравця.

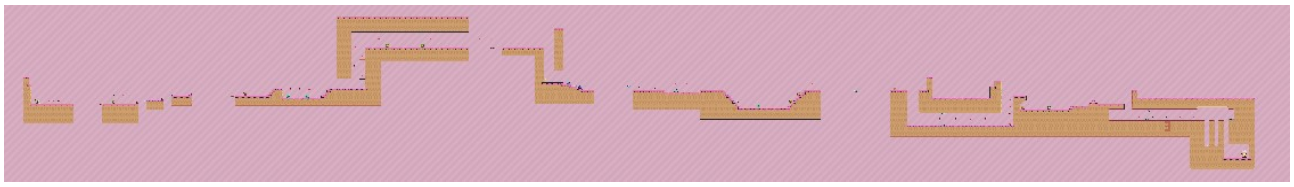

Рисунок 3.38 – Схема третього рівня

Четвертий рівень буде містити велику кількість паркуру, де гравець зможе використовувати свої навички пересування. Також на цьому рівні буде приховане місце з великою кількістю яблук. Рівень буде трохи простіший порівняно з попередніми, щоб гравець міг відпочити та підготуватись перед фінальним випробуванням.

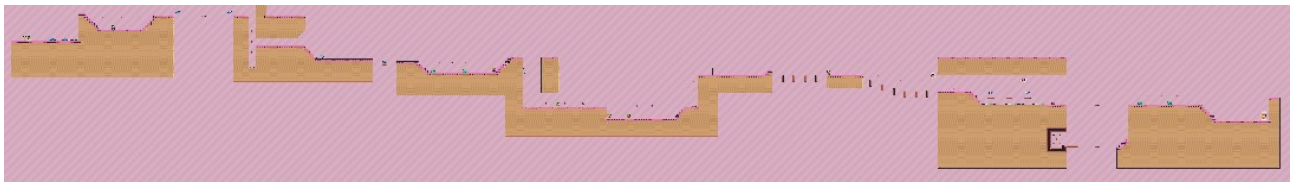

Рисунок 3.39 – Схема четвертого рівня

П’ятий рівень пропонує нову музику на задньому плані та змінене візуальне оформлення, щоб створити особливу атмосферу. Хоча рівень буде меншим у розмірах, він стане справжнім випробуванням для гравця перед фінальним етапом.

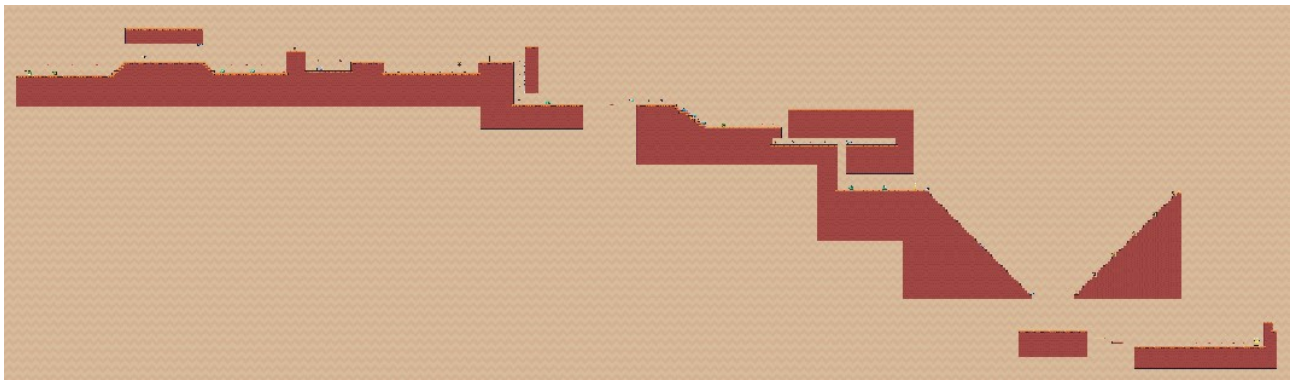

Рисунок 3.40 – Схема п’ятого рівня

Східці, які з’являться перед фіналом, створюють перешкоди, які можуть затримати гравця на кілька спроб. З одного боку сходи будуть патрулювати три пили, а з іншого боку вороги-рослини будуть постійно обстрілювати гравця. Ця складна ділянка буде розташована недалеко від точки збереження, щоб гравець не втратив мотивацію та залишився з веселими спогадами після її успішного проходження.

Для перевірки коректності роботи рівнів, було проведено кілька проходжень гри. Під час ігрового процесу не виявлено багів оточення або інших технічних проблем. Гра працює належним чином, рівні добре збалансовані та надають гравцям саме ті емоції, які вони очікують від гри.

Після завершення розробки всіх сцен рівнів, ми можемо створити виконуваний файл "exe" для нашої гри. Для цього ми переходимо до налаштувань збірки, додаємо усі сцени та налаштовуємо окремі параметри нашого проекту. Ми

додаємо можливість гри в повноекранному режимі, можливість переключення з повноекранного режиму за допомогою клавіші "Win". Також ми встановлюємо цільовою платформою Windows, залишаючи всі інші компоненти налаштувань за замовчуванням.

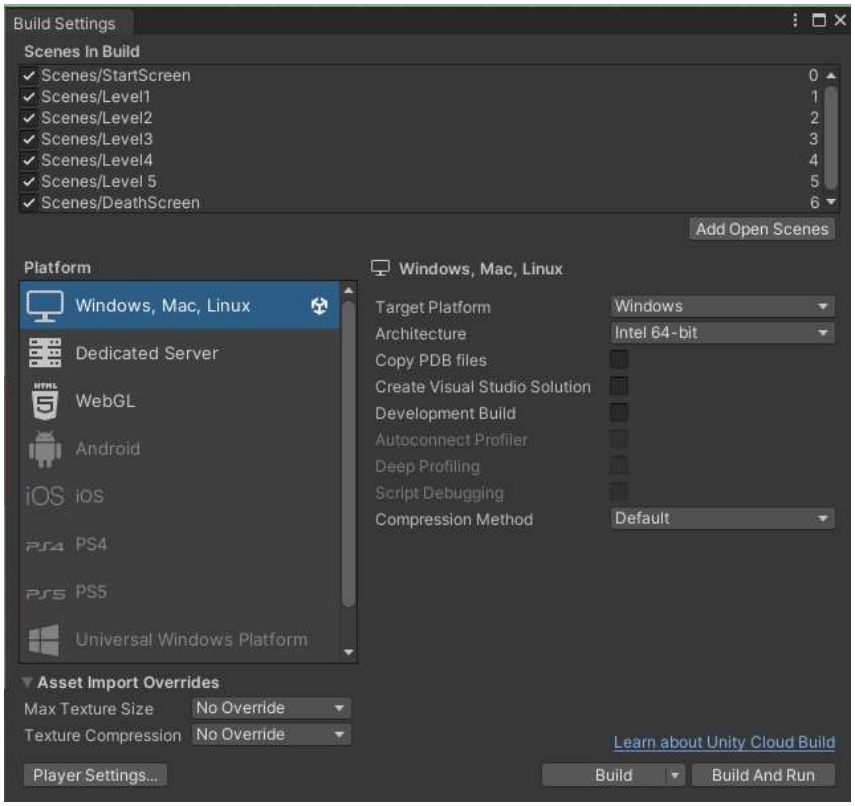

Рисунок 3.41 – Меню налаштування збірок

Цей крок дозволив нам створити виконуваний файл, який гравці зможуть запустити на своїй операційній системі Windows та насолоджуватися нашим 2D платформером.

РОЗДІЛ 4
РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ПРОГРАМНОГО ПРОДУКТУ

4.1 Розрахунок собівартості нового програмного продукту

Розглянемо розрахунок собівартості нашого програмного продукту. Собівартість програмного продукту – це виражені у грошовій формі витрати на його розробку і, при необхідності, реалізацію.

Форма кошторису витрат на розробку програмного продукту наведена у таблиці 4.1.

Таблиця 4.1- Кошторис витрат на розробку програмного продукту

№ з/п	Найменування витрат за економічними елементами	Розмір, грн.	Підстава
1	2	3	4
1	Витрати на оплату праці, зокрема: Заробітна плата керівника дипломної роботи та консультанта з охорони праці	179,6875	Розрахунок за формулами
2	Відрахування на соціальні заходи	39,53125	22,0% від пункту 1
3	Матеріальні витрати	170,00	Розрахунок за текстом методичних рекомендацій
4	Амортизація обладнання	1035,32	Розрахунок за текстом методичних рекомендацій
5	Витрати на електроенергію	154,02	Розрахунок за текстом методичних рекомендацій
6	Витрати на роботи, які виконують сторонні організації	180,00	Розрахунок за квитанціями

№ з/п	Найменування витрат за економічними елементами	Розмір, грн.	Підстава
7	Витрати на машинний час	10800,00	Розрахунок за текстом методичних рекомендацій
8	Накладні витрати	485,22	Розрахунок за формулами
	Загальна собівартість	13043,778	

Завершення таблиці 4.1

4.1.1 Визначення розміру витрат на оплату праці

Середньо-годинна ставка зарплати ($Сг_i$) для кожного з виконавців буде визначатися за формулою 4.1:

$$Сг_i = ЗП_i/F_p \quad (4.1)$$

де F_p – місячний фонд робочого часу (128 години).

$$Сг_{і\text{-}кер} = 12000 / 128 = 93,75 \ (грн.);$$
$$Сг_{іконс\ о.п.} = 11000 / 128 = 85,9375 \ (грн.).$$

Для розрахунку витрат на оплату праці виконавців даного дипломної роботи (табл. 4.2) визначалася трудомісткість роботи кожного з працівників (Ti) (в людино-годинах), виходячи з діаграми завантаження виконавців.

Діаграма завантаження виконавців ДП (Ti, *люд./год.*):

1. Керівник ДП – керівництво ДП – 1 год.
2. Консультант з охорони праці – консультації – 1 год.

Таблиця 4.2 - Розрахунок витрат на оплату праці виконавців дипломної роботи

№ з/п	Посади виконавців	$С_{гі}$ грн./год.	$Т_і$, люд./год.	Витрати на оплату праці (Воп), грн.
1	2	3	4	5
1	Керівник ДП	93,75	1	93,75
2	Консультант з охорони праці	85,9375	1	85,9375
	Разом	-	-	179,6875

Отже, витрати на оплату праці становлять 179,6875 грн.

4.1.2 Відрахування на соціальні заходи

Розрахунок відрахувань здійснюється за наступною формулою:

$$B_{єсв}=0,22*B_{оп}, \quad (4.2)$$

$B_{оп}$ – загальна сума оплати праці.

$$B_{єсв}=0,22 * 179,6875 = 39,53125 \text{ (грн.)}$$

Отже, відрахування на соціальні заходи становлять 39,53125 грн.

4.1.3 Визначення розміру матеріальних витрат

Визначення витрат на матеріали та окремі комплектуючі вироби (табл. 4.3), визначається з урахуванням вирішення поставленого конкретного завдання – розробки програмного продукту.

Таблиця 4.3 - Розрахунок витрат матеріалу

№ з/п	Найменування (вид) матеріалу	Кількість, (один.)	Ціна, (грн.)	Сума (грн.)
1	2	3	4	5
1	Папір, формат А4	1 пачка (200 арк.)	110,00	110,00
2	Фарба для принтера	1 банка	55,00	55,00
3	Диск (DVD-RW)	1 диск	15,00	15,00
	Разом			170,00

Отже, розмір матеріальних витрат становить 170,00 грн.

4.1.4 Визначення розміру амортизаційних відрахувань

Визначення розміру амортизаційних відрахувань (А) визначається за формулою 4.3:

$$A = \frac{K_b * N_a}{12 * 100} * T_{рм}, \quad (4.3),$$

де $K_б$ – балансова вартість однієї ПЕОМ з периферією,

N_a – річна норма амортизаційних відрахувань,

$T_{рм}$ – тривалість розробки програмного продукту у місяцях.

Річні норми амортизаційних відрахувань для ПЕОМ приймаються у розмірі 25%, а для ПЗ – 60%.

$$\text{Т}_{\text{рм}} = \frac{t_{\text{ч}}}{\text{Т}_{\text{міс.}}} T_{рм} = \frac{t_{\text{ч}}}{T_{міс}} \quad (4.4)$$

$$T_{рм} = \frac{30}{31} \text{Т}_{\text{рм}} = \frac{20}{31} = 0{,}96 \text{ (міс.)}$$

Вартість ПК становить 24800,00 грн., принтера – 5900,00 грн. Отже:

$$А_{(ПЕОМ)} = \frac{24800{,}00*25}{12*100} * 0{,}96 = 496{,}00 \text{ (грн.)}$$

$$A_{(принтер)} = \frac{5900{,}00*25}{12*100} * 0{,}96 \frac{15900 \times 25}{12 \times 100} \times 0{,}64 = 118{,}00 \text{ (грн.)}$$

Вартість ліцензії ОС Windows 10 Pro (OEM) становить 5773,41 грн, Microsoft Office 2019 — 3004,00 грн.

$$К_{б(ПЗ)} = 5773{,}41 + 3004{,}00 = 8777{,}41(\text{грн.})$$

Розмір амортизаційних відрахувань для ПЗ:

$$А_{(ПЗ)} = \frac{8777{,}41*60}{12*100} * 0{,}96 \frac{5572{,}00 \times 60}{12 \times 100} \times 0{,}64 = 421{,}32 \text{ (грн.)}$$

Загальний розмір амортизаційних відрахувань:

$$A_{(заг)} = 496{,}00 + 118{,}00 + 421{,}32 = 1035{,}32 \text{ (грн.)}$$

Отже, розмір амортизаційних відрахувань становить 1035,32 грн.

4.1.5 Визначення витрат на електроенергію

Розмір витрат на електроенергію включає:

- витрати на силову електроенергію;
- витрати на електроенергію, яка витрачається на освітлення, та визначається за формулами 4.6, 4.7.

Витрати на силову електроенергію (грн.) визначаються за формулою:

$$З_{с.е} = Т^{год}_{рп} * Ц_е * Р_{ОЕМ,} \quad (4.6),$$

де $Ц_е$ – вартість 1кВт/год. – 1,68 грн.,

$Р_{ОЕМ}$ – сумарна потужність для ПЕОМ з периферією у кіловат-годинах: для комп’ютера – 0,38 кВт/год. та для принтера – 0,12 кВт/год.

$T^{год}_{рп}$ – тривалість розробки програмного продукту у годинах (час використання ПК – 240 год. та час використання принтера – 4 год.).

$$З_{с.е(ПК)} = 240 * 1,68 * 0,38 = 153,21 \text{ (грн.)}$$
$$З_{с.е(принтер)} = 4 * 1,68 * 0,12 = 0,81 \text{ (грн.)}$$

Загальні витрати на силову енергію:

$$З_{с.е} = 153,21 + 0,81 = 154,02 \text{ (грн.)}$$

Витрати на електроенергію для освітлення визначаються за формулою 4.7:

$$З_{ос} = T^{год}_{рп.} * Ц_е * Р_{осв} \quad (4.7),$$

де $Р_{осв}$ – сумарна потужність у кіловат-годинах, яка йде на освітлення.

При виконанні дипломної роботи штучне освітлення не використовувалося.

Загальні витрати на електроенергію складають:

$$З_{загальне} = З_{с.е.} + З_{ос}, \quad (4.8)$$

$$З_{загальне} = 154,02+0=154,02 \text{ грн.}$$

Отже, загальні витрати на електроенергію становлять 154,02 грн.

4.1.6 Розрахунок витрат на роботи сторонніх організацій

До цієї статті належать витрати ($В_{со}$) на виконання окремих робіт для даного дипломної роботи в силу відсутності потрібного обладнання або відповідних спеціалістів і тому виконуються на договірній основі з іншими організаціями.

$В_{со}$ розраховується за формулою 4.9:

$$В_{со} = N*Ц \quad (4.9),$$

де N – кількість робіт (послуг), згідно квитанції,

$Ц$ – ціна 1 роботи (послуги), згідно квитанції.

При розробці даної дипломної роботи було використано наступні послуги сторонніх організацій, як:

1. Вартість брошурування дипломної роботи - 80,00 грн.;

2. Вартість друку -100,00 грн.

Витрати на роботи, які виконують сторонні організації:

$$B_{со} = 1 * 80,00 + 1 * 100,00 = 180,00 \ (грн.)$$

Отже, витрати на роботу, які виконують сторонні організації становлять 180,00 грн.

4.1.7 Розрахунок витрат на машинний час

Розрахунок витрат на машинний час здійснюється за формулою:

$$C_{маш.ч} = Ц_{маш.ч} * T_{маш.ч} \quad (4.10),$$

де $Ц_{маш.ч}$ – собівартість однієї години машинного часу

$Т_{маш.ч}$ – машинний час, використаний для проведення робіт.

Для дипломної роботи приймаємо $Ц_{маш.ч}$ = 45,00 грн.

Необхідна кількість машинного часу для реалізації роботи з розробки програми розраховується за формулою:

$$T_{маш.ч.} = T_i * t_з * T_{ср.маш.}, \quad (4.11),$$

де T_i – трудомісткість робіт, люд.дн,

$t_з$ – тривалість робочої зміни (при п’ятиденному робочому тижні $t_з$ = 8 год.),

$T_{ср.маш}$ – середній коефіцієнт використання машинного часу ($T_{ср.маш}$ = 1).

Машинний час, використаний для проведення робіт:

$$T_{маш.ч.} = 30*8*1 = 240 \text{ (год.)}$$

Витрати на машинний час:

$$C_{маш.ч} = 45,00 * 240 = 10800,00 \text{ (грн.)}$$

Отже, витрати на машинний час складають 10800,00 грн.

4.1.8 Розрахунок накладних витрат

До складу накладних витрат ($B_н$) відносяться:

- витрати, пов'язані з управлінням організацією, де виконується дипломна робота;
- витрати на науково-технічну інформацію;
- витрати на забезпечення нормальних умов праці і техніки безпеки;
- витрати на інші загальногосподарські потреби, тощо.

Накладні витрати розраховуються у відсотках до витрат на оплату праці ($B_{оп}$) і визначаються за формулою 4.12:

$$B_н = \frac{\alpha}{100} * B_{оп}, \qquad (4.12),$$

де a – середньостатистичний відсоток накладних витрат в організації (50%).

$$B_н = \frac{50}{100} * 970,43 = 485,22 \text{ (грн.)}$$

Отже, накладні витрати ($B_н$) становлять 485,22 грн.

4.2 Визначення ціни програмного продукту

Ціна програмного продукту $Ц_{пп}$ визначається за формулою 4.13:

$$Ц_{пп} = С_{пп} + П \quad (4.13),$$

де $С_{пп}$ – собівартість програмного продукту (грн.);

$П$ – прибуток, що планується (грн.) (25% від собівартості):

$$Ц_{пп} = 13043,778 + 3260,9445 = 16304,72 \text{ (грн.)}$$

4.3 Висновки до розділу

Згідно з результатами проведених розрахунків визначено, що кошторисна вартість виконання дипломної роботи становить 16304,72 грн.

Відеоігри це дуже комплексний програмний продукт, що містить багато змінних економічних чинників. Оскільки собівартість аналогів не вказана в офіційних джерелах, то порівнювати її з собівартістю розроблюваного програмного продукту можливості немає.

Врахувавши обчислені відносні показники, можна стверджувати, що розробка була доцільною.

Результати розрахунків економічної частини дипломної роботи зводяться в підсумкову таблицю 4.4.

Таблиця 4.4 - Результати розрахунків економічної частини дипломної роботи

Найменування показника	Значення	Одиниці вимірювання
1	2	3
Собівартість програмного продукту	13043,77	грн.
Прибуток	3260,94	грн.
Ціна програмного продукту	16304,72	грн.

Після проведеного аналізу економічних аспектів розробки нашого 2D платформера можна зробити висновок, що він виявився досить дешевим з точки зору витрат.

Це означає, що ми змогли створити повноцінний 2D платформер, який містить всі основні складові аналогічних ігор, витративши обмежену кількість ресурсів.

Навіть за умов великого інвестування аналогічних проектів, наш 2D платформер буде вирізнятися, як доступний та водночас якісний продукт. Це створює перспективи для успіху нашої гри на ринку та позиціонує нас як конкурентоспроможного учасника відповідної галузі.

Отже, наша досліджувана гра є вигідним і перспективним інвестиційним проектом, здатним досягти успіху у своїй сфері застосування.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1 Аналіз стану охорони праці

Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів і засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі праці.

Основним Законом, що гарантує право громадян на безпечні та нешкідливі умови праці, є Конституція України.

Для аналізу обрано робоче місце, яке розташоване у адміністративній будівлі Львівської районної державної адміністрації.

Робоче місце - це частина простору, в якому працівник здійснює трудову діяльність, і проводить велику частину робочого часу.

Конструкція робочого місця і взаємне розташування усіх його елементів повинно відповідати антропометричним, фізичним і психологічним вимогам.

Даний кваліфікаційна робота розроблялася у кабінеті відділу програмного забезпечення Управління соціального захисту населення Львівської райдержадміністрації, який розташований на першому поверсі в приміщені розміром 5 х 6 х 3 м. Штат працюючих в даному приміщенні - 3 особи. Таким чином на одну людину припадає 10 м2 площі приміщення та 30 м3 його об'єму. Згідно ДСанПіН 3.3.2.007-98 "Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин" на одного працівника встановлюють об'єм виробничого приміщення не менше 19,5м3 та площі приміщення не менше 6м2. Отже, умови для даного приміщення виконуються.

Основним робочим місцем працівників відділу є стіл із встановленим на ньому персональним комп'ютером та лазерним принтером Canon LBP-3000 LJ.

На самопочуття, стан здоров’я людини впливає мікроклімат виробничих приміщень, який визначається дією на організм людини температури, вологості, швидкості руху повітря і теплового випромінювання. Виробничий мікроклімат, як правило, відрізняється значною мінливістю, нерівномірністю по горизонталі та вертикалі, різноманітністю сполучень температури, вологості, рухомості повітря, інтенсивності випромінювання залежно від особливостей технології виробництва, кліматичних особливостей місцевості, конструкцій споруд, організації повітрообміну із зовнішнім середовищем.

Згідно ДСТУ 15251:2011 мікроклімат робочої зони нормується залежно від періоду року та категорії робіт за енерговитратами.

За енерговитратами роботи, що виконуються в приміщенні належать до легких (витрати менше 150 ккал/год).

Згідно ДСТУ 6089:2009 у приміщенні підтримуються метеорологічні умови, які наведені в табл. 5.1.

Таблиця 5.1 Норми оптимальних параметрів мікроклімату в робочій зоні виробничих приміщень.

Параметри	**Холодний**	**Теплий період**
Оптимальні:		
Температура, °С	22-24	23-25
Відносна вологість, %	40-60	40-60
Швидкість руху повітря, м/с	0,1	0,1

При роботі лазерного принтера виділяються озон, оксиди азоту (NO_2, N_2O), ацетон, паперовий пил, ультрафіолетове і інфрачервоне (теплове) випромінювання. При нагріванні паперу активно випаровується формальдегід і водяна пара.

При незадовільному освітленні приміщення знижується продуктивність праці користувачів ПК, можлива поява короткозорості, настає швидка втомлюваність. Тому світлові прорізи вікон в відділі та їх кількість повинні

забезпечувати достатню освітленість робочої поверхні, яка нормується СНиП II-4-79 "Природне і штучне освітлення". Природне освітлення в приміщенні забезпечене, коефіцієнт природного освітлення (КПО) складає 3,2%.

Штучне освітлення в приміщенні здійснюється системою загального рівномірного освітлення. При цьому світильники місцевого освітлення встановлені таким чином, щоб не створювати бліків на поверхні екрана, а освітленість екрана має не перевищувати 300 лк. Як джерела світла штучного освітлення застосовуються люмінісцентні лампи типу ЛБ.

Оскільки в відділі проводяться, в основному, роботи з використанням обчислювальної техніки, то згідно СНиП II-4-79 мінімальна освітленість становить 300 лк.

Джерелами шуму у приміщенні являються вентилятори системних блоків комп'ютерів. Остання перевірка рівня шуму (березень 2020 року) при паспортизації райдержадміністрації показала, що еквівалентні рівні шуму на робочих місцях не перевищують 35 дБА, що відповідає загальним вимогам безпеки, встановленим ДСН 3.3.6.037-99 (згідно вказаних вимог еквівалентні рівні шуму на робочих місцях не повинні перевищувати 50 дБА).

Джерелами випромінювання електромагнітної енергії радіочастотного діапазону є монітори. Електромагнітні поля можуть негативно впливати на організм людини. Для захисту персоналу від їх шкідливої дії використовуються: монітори зі зниженою випромінювальною здатністю, віддалення робочого місця на безпечну відстань, зменшення часу перебування у небезпечній зоні, застосування засобів індивідуального захисту, а також дотримання регламентованого режиму праці і відпочинку.

Розглянемо ергономіку та технічну естетику. Робочий стіл має стабільну конструкцію: площина столу складає 140×100 см і регулюється по висоті в діапазоні 30 см, висота від горизонтальної лінії зору до робочої поверхні столу складає 40 см.

Покриття поверхні столу матове з коефіцієнтом відбиття 15%, легке у

догляді. Кути столу заокруглені, для запобігання поранень.

Сидіння нахиляється по відношенні до горизонталі вперед на 20° і назад на 140°. Розмір сидіння 40×40 см. Висота спинки крісла складає 30 см від поверхні сидіння. Підніжка крісла має п’ять опор, щоб запобігти його падінню.

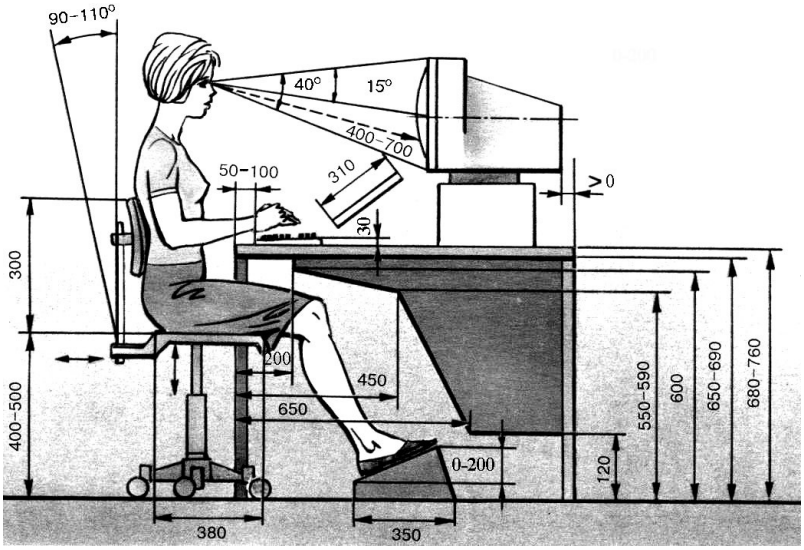

Рисунок 5.1 - Схематичний малюнок ергономії робочого місця

Ергономічне робоче крісло є простим у управлінні: всі важелі без зусиль перемикаються, а механізм трансформації супроводжується мінімумом рухів. Крісло також має динамічне сидіння, що регулюється по висоті. Регулювання спинки по висоті і глибині дозволяє хребту розслабитися. У кріслі передбачена спеціальна адаптивна опора для попереку. Динамічне робоче крісло знижує втомленість і відповідає умовам ергономіки.

Використання чинників виробничого естетичного впливу на працівників у значній мірі може призвести до підвищення рівня їх працездатності. До чинників естетичного впливу належить колір та музика.

Естетичне оздоблення виробничих приміщень сприяє підвищенню продуктивності праці і рівня промислової безпеки та загальному поліпшенню умов праці.

Правильний добір кольору фарбування стелі, стін, обладнання дозволяє забезпечувати сприятливе зорове їх сприйняття, підвищувати трудову активність,

сприяти підвищенню у виробничих приміщеннях ділової атмосфери, чистоти та порядку. Тому стіни пофарбовані у білий колір, що заспокійливо діє на нервову систему та зоровий аналізатор.

5.2 Захисне заземлення

Згідно з ДСТУ 7237:2011 приміщення відноситься до класу приміщення без підвищеної небезпеки. Приміщення без підвищеної небезпеки характеризуються відсутністю умов, що створюють особливу або підвищену небезпеку. В приміщенні використовується однофазна мережа частотою 50 Гц, напругою 220 В. Підвищення електробезпеки досягається застосуванням систем захисного заземлення, занулення, захисного відключення й інших засобів і методів захисту, у тому числі знаків безпеки і попереджувальних плакатів і написів.

Живлення ПК споживачів здійснюється через пристрій захисного відімкнення, який розташований у доступному місці (біля робочого місця). Проводка виконана проводом з подвійною ізоляцією з використанням захисного заземляючого провідника.

Проводимо розрахунок контурного заземлення. Вихідними даними для розрахунку є:

- допустимий опір розтікання струму в землі заземлюючого пристрою ($R_{ЗН}$ =4Ом):

- питомий опір ґрунту в місці спорудження заземлювача $\rho_З$ = 45 Ом-м;

- тип ґрунту – глина;

- тип заземлювача (труба) , його конструктивні розміри : d = 0,05 м,

h = 1 м; l = 2 м;

- конструкція заземлюючого пристрою контурна (заземлювачі розташовані по контуру).

Розрахунок проводимо за наступною методикою:

Розраховуємо питомий опір землі:

$$\rho_{рз} = Y * \rho_з, \quad (5.1)$$

де Y – коефіцієнт сезонності, значення якого вибираємо: $Y = 1{,}3$, тоді

$$\rho_{рз} = 1{,}3 * 45 = 58{,}5 \ (Ом\text{-}м)$$

Визначаємо опір розтікання струму в землі одного вертикального заземлювача, забитого на глибину h від поверхні

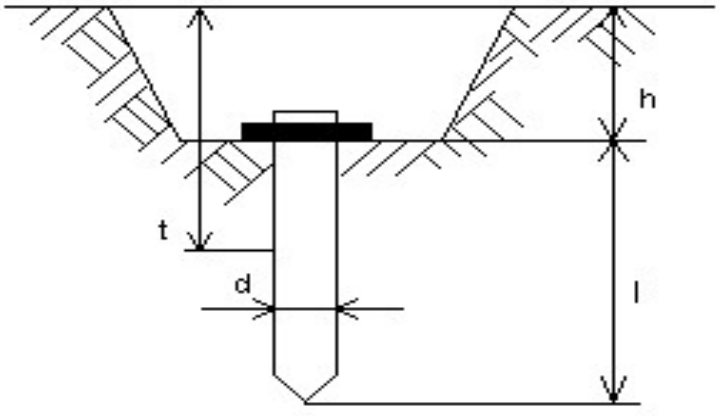

Рисунок 5.2 - Схема заземлення

$$R_в = (\rho_{рз} / 2\pi l * (\ln * (2l/d) + 0{,}5 * \ln ((4t + l)/4t - l)), \quad (5.2)$$

де $R_В$ – опір розтікання струму в землі вертикального заземлювача, Ом;

$\rho_{рз}$ – розрахунковий питомий опір землі, Ом –м;

l – довжина заземлювача, м ;

d – діаметр заземлювача, м.

t = h + 1/2=1 + 2/2 = 2 м.

$R_в = (58{,}5/(2*3{,}14*2)*(ln\ (2*2/0{,}05) + 0{,}5*ln(4*2 + 2)/(4*2\text{-}2)) = 21{,}6$ *(Ом)*

Визначаємо кількість заземлювачів за формулою:

$$n = R_В/R_{ЗН}, \quad (5.3)$$

$$n = 21{,}62/4 = 5{,}4;$$

Приймаємо $n = 6$.

Коефіцієнт використання заземлювачів $\eta_в$ залежно від відношення віддалі між електродами до довжини електроду $k = a_1/1$, де $a_1 = 3$ *м*. Оскільки $k = 3/2 = 1{,}5$, то $\eta_В$ становитиме *0,83*.

Кількість заземлювачів з врахуванням $\eta_в = 0{,}83$:

$$n = R_в/(R_{ЗН}*\eta_в) = 21{,}62/(4*0{,}83) = 6{,}51. \quad (5.4)$$

Отже, необхідна кількість заземлювачів становить 7 шт.

Знаходимо довжину горизонтальної смуги заземлювача, яка з'єднує вертикальні заземлювачі по контуру:

$$L = a_1 * n = 3*7 = 21\ (м). \quad (5.5)$$

Визначаємо опір горизонтального заземлювача $R_Г$, прокладеного на глибині h від поверхні землі:

$$R_г = (\rho_{рз}/(2\pi L)*ln*((2L^2)/(d*h), \quad (5.6)$$

де, $R_Г$ – опір розтікання струму в землі горизонтального заземлювача, Ом;

L – довжина горизонтального заземлювача, м;

d – діаметр заземлювача, м;

h – глибина розташування заземлювача, м.

$$R_г = (58{,}5/(2*3{,}14*21))\ ln\ (2*21*21/(0{,}05*1)) = 4{,}34\ (Ом).$$

Обчислюємо загальний опір заземлюючого пристрою:

$$R_З = R_в R_г / (n*R_г*\eta_в + R_в*\eta_2), \quad (5.7)$$

де, $R_З$ – загальний опір заземлюючого пристрою, Ом;

η_2 – коефіцієнт використання горизонтального заземлювача, який при $n = 7$ та $k = 1,5$, становитиме $\eta_2 = 0,53$.

$$R_З = 21,62*4,34/(7*4,34*0,83 + 21,62*0,53) = 2,56 \text{ (Ом)}.$$

Таким чином, $R_З < R_{Зн}$, що відповідає вимогам ПУЕ 1.7.65.

5.3 Безпека в надзвичайних ситуаціях

Пожежна безпека забезпечується шляхом проведення організаційних, технічних та інших заходів, спрямованих на попередження пожеж, забезпечення безпеки людей, зниження можливих майнових втрат і зменшення негативних екологічних наслідків у разі їх виникнення, створення умов для швидкого виклику пожежних підрозділів та успішного гасіння пожеж.

У районній державній адміністрації встановлений порядок (система) оповіщення людей про пожежу, з яким ознайомлюють всіх працівників. У приміщеннях на видних місцях розташовані таблички із зазначеними номерами телефонів для виклику пожежної охорони.

Працівники райдержадміністрації дотримуються встановленого протипожежного режиму, виконують вимоги правил та інших нормативних актів з питань пожежної безпеки.

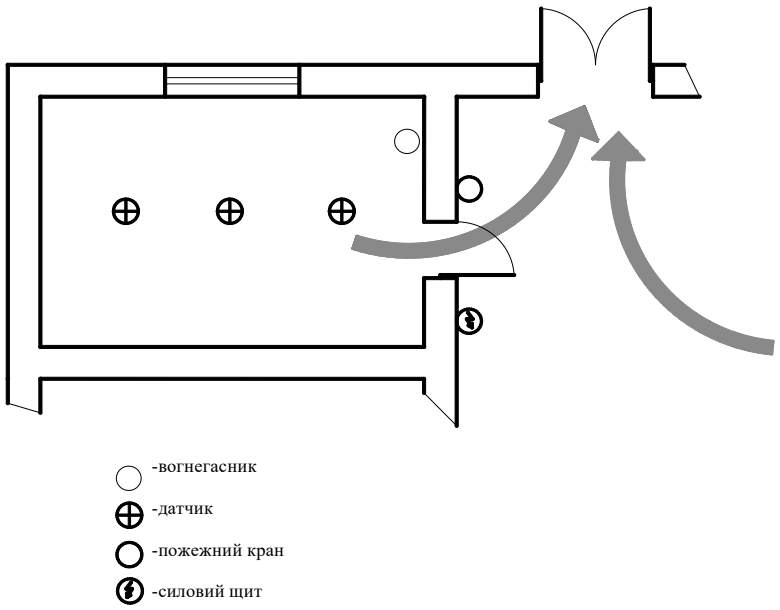

Рисунок 5.3 - Схема евакуації працівників на випадок пожежі

Приміщення оснащені системою автоматичної пожежної сигналізації, а також переносними вуглецево-кислотними вогнегасниками з розрахунку дві штуки на кожні 20 м.кв. площі приміщення. У приміщенні відділу програмного забезпечення, де розроблялась програма в рамках кваліфікаційної роботи, був встановлений вогнегасник ВВК-2, вогнегасна здатність — 3675-98.

5.4 Висновки до розділу

Отже, в даному розділі кваліфікаційної роботи були передбачені заходи по охороні праці (забезпечення ергономічних, санітарних та метеорологічних умов, відповідність природного та штучного освітлення вимогам СНиП II-4-79, дотримання вимог техніки безпеки та пожежної безпеки), що відповідають вимогам нормативних документів і актів та забезпечують нормальну, ефективну і безпечну для здоров'я людини життєдіяльність на виробництві. Рішення питань захисного заземлення підтверджено відповідними розрахунками. Враховані питання пожежної безпеки та складений план евакуації персоналу на випадок виникнення небезпечних ситуацій.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В ході роботи ми розібрали поняття 2D платформера та розглянули проекти, що заснували жанр. Також були проаналізовані сучасні комп'ютерні 2D платформери, що демонструють постійне покращення та еволюцію у різних аспектах. Було, визначено, що популярність та довговічність жанру пов'язана з великою фанатською базою, низьким порогом входження та незначними системними вимогами, що на фоні масштабних проектів виглядає куди привабливіше для сучасних геймерів. В сумі ці фактори підштовхнули до створення ще одного представника жанру 2D платформерів, що зацікавить своїм невибагливим, але досить веселим геймплеєм сучасну людину, що часто перебуває у постійних турботах.

Для створення гри ми проаналізували сучасні ігрові рушії. Вибір зупинили на двигуні Unity. Пов'язано це було з наявністю безкоштовної версії, невисокими системними вимогами та низькою оцінкою порогу входження, що була сформований на основі великої кількості документації для новачків та відносно простої мови програмування C#. Для більш зручного процесу програмування елементів гри підібрали оптимальне середовище розробки Visual Studio, що має всі необхідні нам функції та добру інтеграцію з Unity.

Після вибору оптимальних інструментів розробки ми розробили концепцію гри, а саме:

- Вказали основні механіки
- Обрали оптимальний візуальний стиль
- Визначили звуковий дизайн гри
- Вибрали додаткові геймплейні елементи

Після того як ми завантажили всі візуальні та аудіоелементи, приступили до програмування компонентів гри. Запрограмували кілька типів ворогів та пасток, реалізували механіку смерті та точок збереження для гравця, додали можливість збирати предмети на рівні та взаємодіяти з різноманітними елементами гри. Також

для зручності гравця було реалізовано можливість ставити гру на паузу, повертатися в головне меню, вимкнути музику або вийти з гри.

Також особлива увага була приділена створенню анімацій, які додають живості нашому платформеру. Для кожного рухомого об’єкта, як гравець, ворог чи прапор фінішної прямої створено свою анімацію, з оптимальними налаштуваннями для гармонічного вигляду.

Було створено п’ять захоплюючих рівнів, які випробовують навички гравця та забезпечують цікавий геймплей. Кожен рівень кидає нові та складніші виклики, а також містить свою стилізацію, що додає гравцю відчуття прогресу.

Наша робота над грою завершилася створенням виконуваного "exe" файлу, що дозволяє гравцям запускати гру на своїх комп’ютерах без необхідності скачувати додаткові файли. Це дозволяє нам поділитися результатами нашої роботи з широкою аудиторією, що переважно вибирає Windows як ігрову платформу.

Також проведено ряд економічних заходів, що підтверджує перспективну економічну сторону проекту, оскільки його вартість виявилась не значною. Крім цього були проведені заходи з аналізу стану охорони праці та безпеки в надзвичайних ситуаціях.

У дипломну проекті вдалося реалізувати всі поставлені цілі не використовуючи при цьому велику кількість ресурсів. Наша гра не відрізняється від аналогів по жанру і містить в собі всі ті елементи, за які її так люблять гравці. Таким чином, розроблений платформер містить всі переваги жанру та виконує головну ціль – надає приємне проведення дозвілля для людей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Godot [Електронний ресурс] — Режим доступу: https://godotengine.org/ (дата звернення 26.01.2023)
2. Rider [Електронний ресурс] — Режим доступу: https://www.jetbrains.com/rider/ (дата звернення 03.02.2023)
3. Unity [Електронний ресурс] — Режим доступу: https://unity.com/ (дата звернення 25.01.2023)
4. Unreal Engine [Електронний ресурс] — Режим доступу: https://www.unrealengine.com/ (дата звернення 24.01.2023)
5. Visual Studio and Visual studio code [Електронний ресурс] — Режим доступу: https://www.microsoft.com/uk-ua/ (дата звернення 03.02.2023)
6. Асети для Unity [Електронний ресурс] — Режим доступу: https://assetstore.unity.com/ (дата звернення 15.02.2023)
7. Документація по C# [Електронний ресурс] — Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення 25.02.2023)
8. Документація по Unity [Електронний ресурс] — Режим доступу: https://docs.unity3d.com/Manual/index.html (дата звернення 22.02.2023)
9. Історія розвитку платформерів [Електронний ресурс] — Режим доступу: https://www.redbull.com/in-en/evolution-of-platformers (дата звернення 10.01.2023)
10. Історія рушія Godot [Електронний ресурс] — Режим доступу: https://gamefromscratch.com/a-decade-of-godot/ (дата звернення 26.01.2023).
11. Історія серії Donkey Kong [Електронний ресурс] — Режим доступу: https://history-computer.com/donkey-kong-complete-history-every-game-in-order-and-more/ (дата звернення 10.01.2023)
12. Порівняння IDE для C# [Електронний ресурс] — Режим доступу: https://www.codeguru.com/tools/7-best-c-sharp-ide/ (дата звернення 04.02.2023)

13. Порівняння сучасних ігрових рушіїв [Електронний ресурс] — Режим доступу: https://www.incredibuild.com/blog/top-gaming-engines-you-should-consider (дата звернення 02.02.2023)

14. Статистика ігрової індустрії [Електронний ресурс] — Режим доступу: https://www.statista.com/topics/868/video-games/#topicOverview (дата звернення 11.01.2023).

ДОДАТКИ

ДОДАТОК А

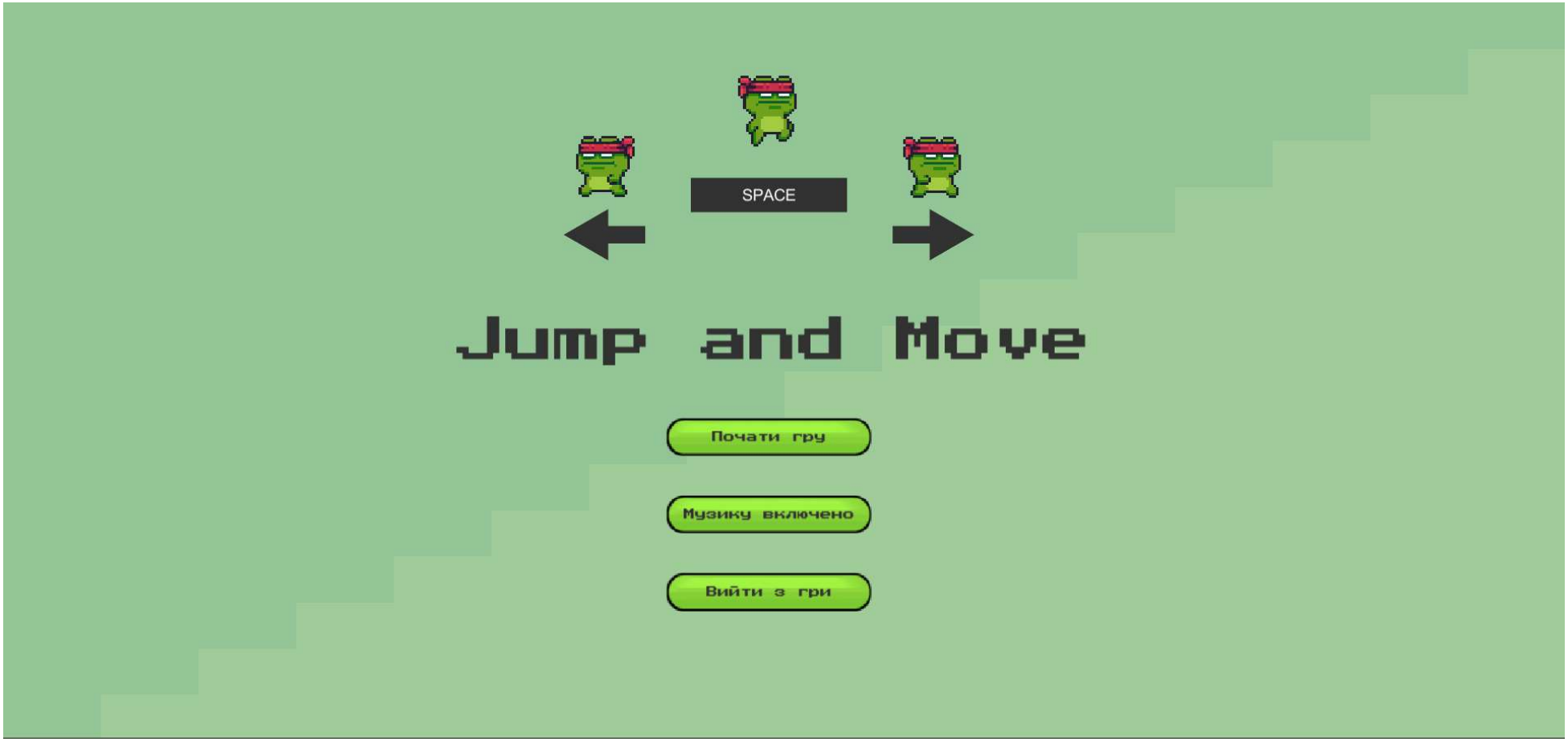

Рисунок А.1 – Головне меню гри

ДОДАТОК Б

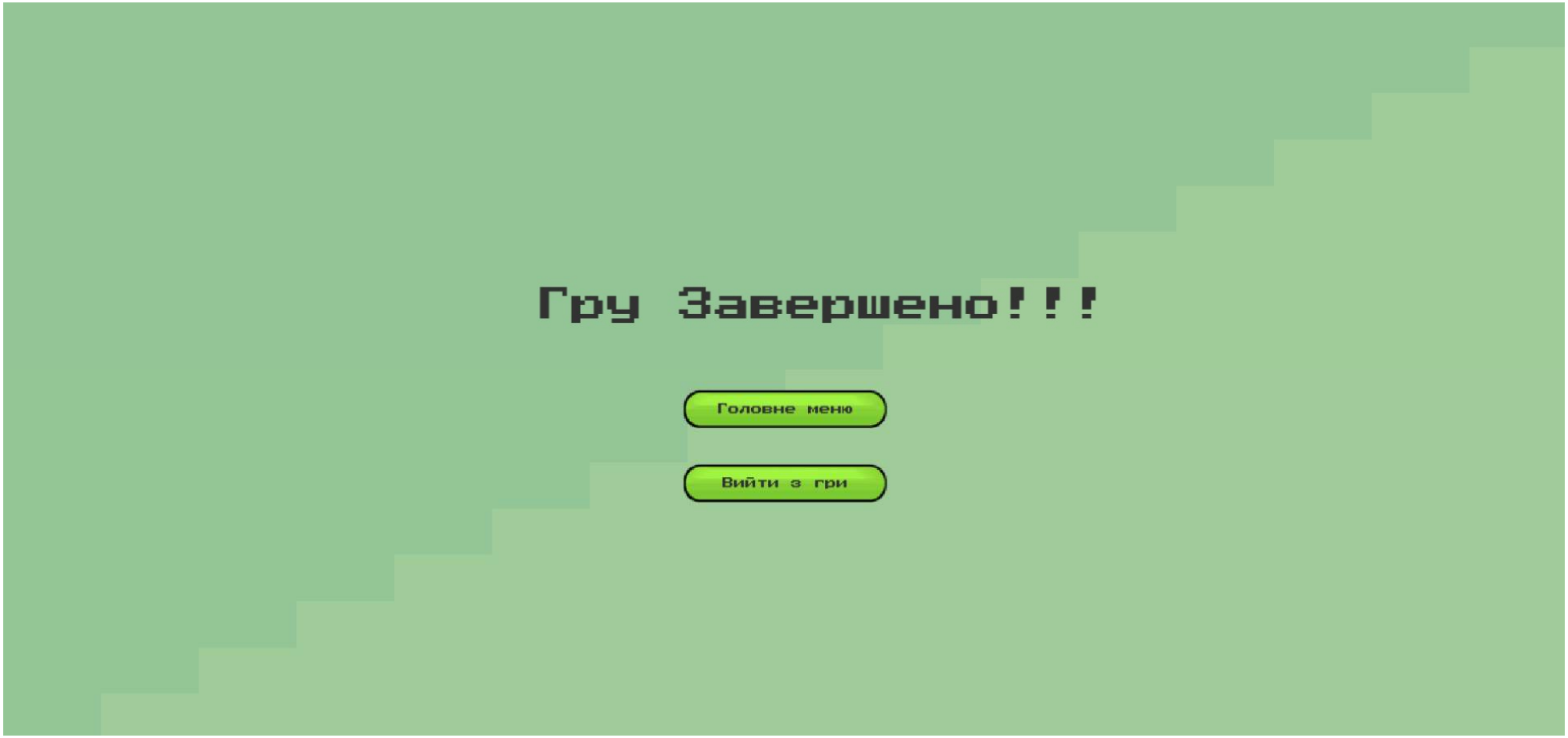

Рисунок Б.1 – Меню завершення гри

ДОДАТОК В

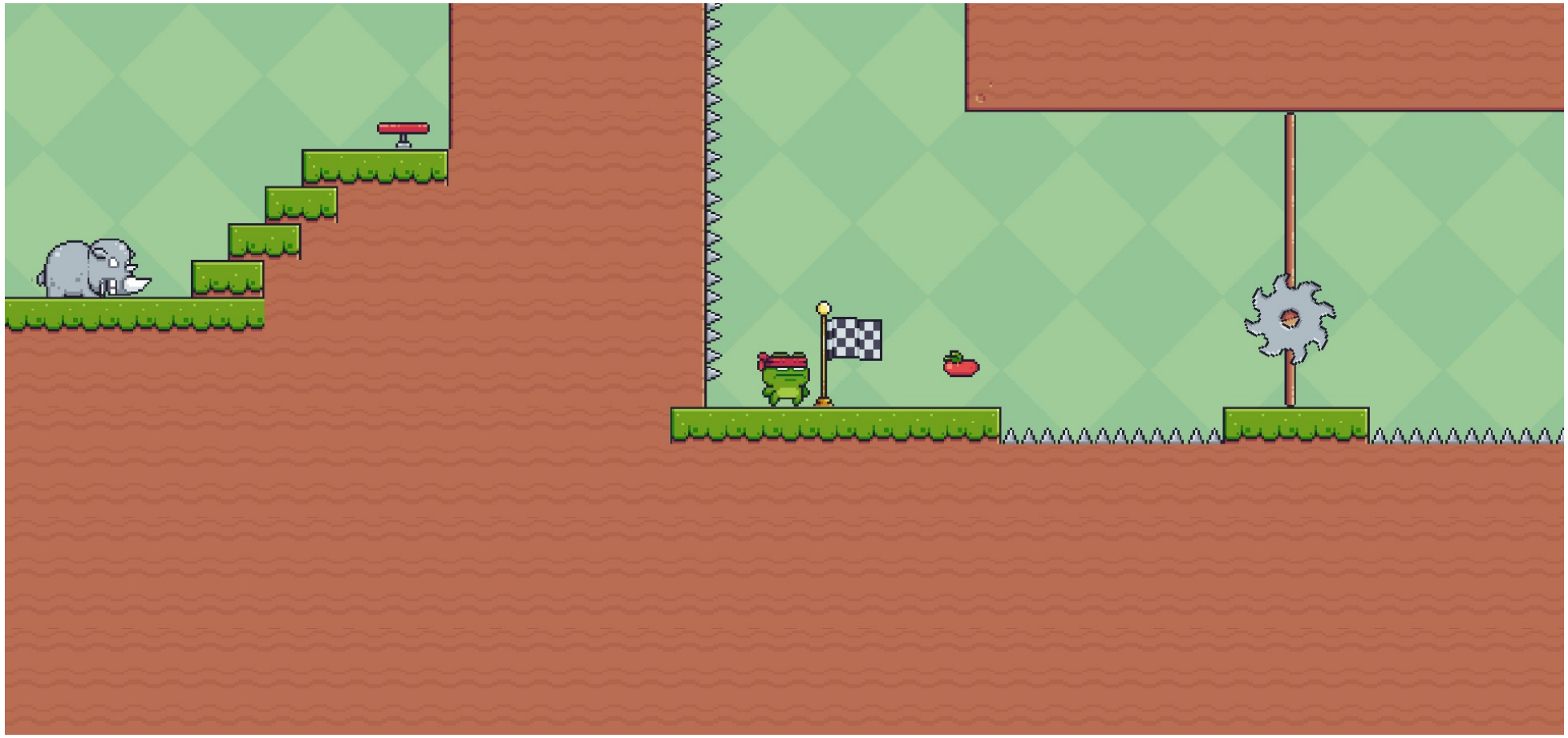

Рисунок В.1 – Ігровий процес