

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Розробка гри "On the edge" в жанрі (2D,
платформер)»**

Виконав студент групи ЗПЗк-22-1

БАТЮК Богдан Олександрович

Керівник роботи:

ЄФРЕМОВ Юрій Миколайович

Рецензент:

ЧИЖМОТРЯ Олексій Володимирович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»
зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БАТЮК Богдан Олександрович**
Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**
Тема роботи: **«Розробка гри "On the edge"»,**
затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №75/с**
Вихідні дані для роботи: **Розробка гри "On the edge" в жанрі (2D, платформер)**
Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Юрій ЄФРЕМОВ	02.03.2024	02.03.2024
2	Юрій ЄФРЕМОВ	04.04.2024	04.04.2024
3	Юрій ЄФРЕМОВ	02.05.2024	02.05.2024

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з комп'ютерної гри та пояснювальної записки. Пояснювальна записка до випускної роботи містить 63 сторінок, 41 ілюстрацію, 4 таблиці, 14 додаток та 20 використаних джерел.

Метою роботи є розробка комп'ютерної гри в жанрі 2D платформера. Гра має забезпечувати якісний ігровий процес із плавною роботою. Вона також повинна бути оптимізована і мати зручний користувацький інтерфейс.

У роботі були виконанні ключові завдання для розробки комп'ютерної гри, проведено аналіз функціональних вимог, вибрано відповідні технології та архітектурні рішення, розроблено та впроваджено алгоритми гри, а також здійснено тестування програмного продукту.

КЛЮЧОВІ СЛОВА: КОМП'ЮТЕРНА ГРА, ЮНІТІ, СКРИПТ, АССЕТ, ГЕЙМПЛЕЙ, ІНТЕРФЕЙС.

					ІПЗ.КР.Б – 121 – 24 - ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Батюк. Б.О.			Розробка гри "On the edge" в жанрі (2D, платформер)	Літ.	Арк.	Аркушів	
Керівник		Єфремов Ю. М.					3	92	
						Житомирська політехніка, група ЗІПЗк-22-1			
Н. контр.		Тичина О.П.							
Зав. каф.		Вакалюк Т.А.							

ABSTRACT

Bachelor's thesis consists of a computer game and an explanatory note. The explanatory note contains 63 pages, 41 illustrations, 4 tables, 14 appendix, and 20 sources used.

The aim of the thesis is to develop a 2D platformer computer game. The game aims to provide a high-quality gameplay experience with smooth operation. It should also be optimized and have a user-friendly interface.

The thesis addresses key tasks for the development of the computer game, including analysis of functional requirements, selection of appropriate technologies and architectural solutions, development and implementation of game algorithms, and testing of the software product.

KEYWORDS: COMPUTER GAME, UNITY, SCRIPT, ASSET, GAMEPLAY, INTERFACE.

					ПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ЗМІСТ	5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ КОМП'ЮТЕРНОЇ ГРИ	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту	10
1.3 Вибір та обґрунтування інструментальних засобів для розробки	19
Висновки до першого розділу	22
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ГРИ.....	23
2.1 Визначення варіантів використання та розробка ігрового додатку.....	23
2.2 Розробка дизайну рівня	28
2.3 Опис створення анімацій.....	32
2.4 Реалізація функціоналу гри.....	36
Висновки до другого розділу.....	47
РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ	48
3.1 Опис роботи з програмним додатком	48
3.2 Тестування роботи програмного забезпечення	55
Висновки до третього розділу	60
ВИСНОВКИ	61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	62
ДОДАТКИ.....	64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI - інтерфейс користувача (англ. User Interface)

GUI - графічний інтерфейс користувача (англ. Graphical User Interface)

HUD - інтерфейс відображення головної інформації (англ. heads-up display)

FPS - кадри на секунду (англ. frames per second)

ПЗ – програмне забезпечення

					ПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Створення комп'ютерної гри в жанрі 2D-платформера визначається кількома ключовими аспектами, що роблять її надзвичайно важливою і цікавою для сучасної індустрії.

Навіть не зважаючи що жанр 2D-платформера існує понад сорока років, він все ще є одним з найпопулярніших серед гравців усіх вікових груп. Гра має простий, але захоплюючий геймплей, який пропонує користувачам виклики в розв'язанні головоломок, швидкісний рух та стратегічні вирішення завдань. Такий формат сприяє популяризації гри серед широкого кола гравців і забезпечує стабільну аудиторію [1].

Розробка 2D-платформера дозволяє розробникам глибше вивчити ключові аспекти розробки ігор, такі як фізика руху персонажа, обробка колізій, анімація та взаємодія з об'єктами. Це дозволяє розширювати креативні можливості команди розробників і вдосконалювати їх навички в процесі роботи над проєктом [2].

Використання сучасних платформ для створення гри в жанрі 2D-платформера надає доступ до потужних інструментів та технологій, які сприяють зниженню витрат часу і зусиль на розробку. Ці платформи надають розширені можливості для створення анімацій, управління аудіо та візуальними ефектами, що сприяє створенню якісного інтерактивного досвіду для гравців.

Таким чином, створення комп'ютерної гри в жанрі 2D-платформера на основі сучасних інструментів є актуальним завданням. Сучасні методи розробки включають швидке прототипування, розробку дизайн-документу, алгоритми фізики та штучного інтелекту, створення інтуїтивного інтерфейсу, оптимізацію графіки та продуктивності коду, а також ручне і автоматизоване тестування. Використання цих методів дозволяє реалізувати креативні ідеї розробників.

Метою цієї роботи є розробка комп'ютерної гри в жанрі 2D-платформера.

Список завдань до мети данної роботи:

- Аналіз існуючих аналогів: огляд популярних продуктів в цьому жанрі для ідентифікації геймплейних механік і архітектурних рішень;

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

- Визначення функціональних вимог: складання детального списку функцій і можливостей, які має включати гра для забезпечення цікавого ігрового процесу та задоволення потреб користувачі;
- Вибір технологій та інструментів: вибір оптимальної платформи розробки, мови програмування та додаткових інструментів для реалізації поставлених завдань;
- Проектування геймплейних механік: розроблення концепції гри, включаючи фізику персонажів, систему управління, інтерактивні об'єкти та інші аспекти, що впливають на ігровий процес.
- Реалізація алгоритмів і програмування: написання програмного коду для реалізації визначених функціональних вимог і геймплейних механік.
- Тестування і відладка: проведення систематичного тестування гри для виявлення та виправлення помилок, а також оптимізація її продуктивності та стабільності.

Об'єктом дослідження є процес розробки комп'ютерної гри в жанрі 2D-платформера, включаючи всі етапи створення гри від концепції до фінального продукту.

Предметом дослідження є технології та методи розробки 2D-платформерів, зокрема алгоритми ігрової механіки, способи взаємодії користувача з інтерфейсом, а також методи оптимізації і тестування програмного забезпечення.

В роботі було використано методи аналізу, проектування, програмування, тестування, моделювання та емпіричний метод для оцінки якості гри.

Практичне значення отриманих результатів: створена комп'ютерна гра включає основні функціональні елементи 2D-платформера, які можуть бути використані для подальшого розроблення рівнів та готові до випуску й реалізації.

Структура пояснювальної записки: випускна робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел із 20 найменувань, 14 додатку, 4 таблиць та 41 ілюстрацій по тексту.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ КОМП'ЮТЕРНОЇ ГРИ

1.1 Постановка задачі

Індустрія відеоігор вже давно стала невід'ємною частиною сучасної культури та економіки. З розвитком технологій та зростаючими вимогами гравців, розробка ігор постійно вдосконалюється, пропонуючи нові ідеї та механіки. Одним з популярних жанрів є 2D платформери, які, незважаючи на свою відносну простоту, продовжують привертати увагу гравців завдяки своїй доступності та різноманітності [3].

2D платформери мають довгу історію, яка бере свій початок з кінця 1970-х років, коли з'явилися перші аркадні ігри. Вони стали основою для багатьох класичних ігор, таких як "Super Mario Bros.", "Sonic the Hedgehog" та "Mega Man". Ці ігри задали стандарти жанру, що включають стрибки, біг та вирішення головоломок у двовимірному середовищі. З часом, жанр 2D платформерів розширився, включаючи нові механіки та елементи, такі як бойові системи, дослідження світу та інтерактивні історії [3].

На сьогоднішній день існує безліч 2D платформерів, від класичних аркад до сучасних інді-ігор, що використовують найновіші технології для створення візуально привабливих та ігрових світів. Наприклад, такі ігри, як "Hollow Knight" та "Celeste", отримали високу оцінку за свої інноваційні підходи до жанру, глибоку сюжетну лінію та складність, яка забезпечує гравцям виклик і задоволення від гри.

Метою кваліфікаційної роботи бакалавра є розробка комп'ютерної гри в жанрі 2D платформера, яка слугуватиме не лише завершеним продуктом, але й базою для подальшого розвитку та створення нових рівнів і можливостей.

Розробка включає в себе:

1) проектування геймплею:

- Визначення основних механік гри, таких як рух персонажа, стрибки, взаємодія з об'єктами тощо.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		9

- Розробка концепції рівнів і розвитку сюжету, що впливає на геймплей.
- 2) створення рівнів:
- Проектування та реалізація ігрових карт, які включають в себе платформи, перешкоди, секрети та інші елементи.
 - Визначення структури рівнів, у тому числі прогресії складності і візуального дизайну.
- 3) дизайн поточення:
- Створення концепцій персонажів і їх візуального вигляду.
 - Розробка анімацій персонажів, включаючи рухи і взаємодію з оточенням.
 - Дизайн фонів, об'єктів та інших візуальних елементів гри.
- 4) реалізація ігрових механік:
- Програмування фізики і взаємодії об'єктів у грі.
 - Впровадження управління персонажем, інтерактивних елементів та ігрових правил.
 - розробка архітектури програмного забезпечення:
 - Впровадження модульності для забезпечення масштабованості і підтримки нових функцій.
- 5) тестування та оптимізація продукту:
- Виконання тестів для виявлення і усунення помилок і багів.
 - Оптимізація гри для забезпечення стабільної роботи.
 - Налаштування балансу складності, інтерфейсу користувача та ігрових ефектів.

1.2 Аналіз аналогів програмного продукту

2D платформери залишаються популярними у світі завдяки своїй простоті та стилю. Вони є важливим сегментом індустрії відеоігор, особливо серед інді-розробників, які часто обирають цей жанр через його відносну простоту в розробці.

Сучасні тенденції:

- Ретро-естетика: відродження інтересу до класичних 2D платформерів завдяки інді-іграм, таким як "Celeste" та "Hollow Knight".

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		10

- Інновації: впровадження нових механік та історій, що додають глибини ігровому процесу.
- Платформи: доступність на різних платформах, включаючи консолі, ПК та мобільні пристрої.

Популярність:

- Широка аудиторія: ігри у жанрі 2D платформера приваблюють як ветеранів відеоігор, які цінують класичні механіки, так і нових гравців, які шукають прості та захопливі ігрові досвіди.
- Ринковий вплив: багато відомих розробників починали з 2D платформерів, і цей жанр часто використовується для демонстрації нових ігрових концепцій та технологій.

Відмінні риси 2D платформерів :

1) Геймплей:

- Платформи: основний елемент гри — це платформи, на які гравець може стрибати та пересуватися.
- Стрибки: важлива механіка, яка зазвичай вимагає точного управління персонажем.
- Збирання об'єктів: гравці часто збирають різні предмети (монети, бонуси), які можуть впливати на прогрес у грі.
- Вороги: присутні вороги, з якими потрібно боротися або уникати.
- Завдання: виконання завдань або проходження рівнів для досягнення мети гри.

2) Управління:

- Просте управління: зазвичай обмежується кількома кнопками для переміщення, стрибків та атак.
- Інтуїтивність: гравці повинні легко розуміти, як керувати персонажем.

3) Дизайн рівнів:

- Прогресивна складність: рівні стають складнішими по мірі проходження гри.
- Секрети та бонуси: часто містять приховані області або бонусні рівні.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

4) Візуальний стиль:

- Піксель-арт: багато 2D платформерів використовують стиль піксель-арт для створення ретро атмосфери.
- Деталізація оточення: важливі деталі у фонах та анімаціях для створення атмосфери гри.

Отже, для аналізу з співпадаючими рисами було обрано такі ігри, як Dead Cells, Hollow Knight та Celeste.

1. Dead Cells (Рис 1.1):



Рисунок 1.1 – Комп'ютерна гра Dead Cells

"Dead Cells" - це рогаликвана метроїдванія з динамічним бойовим геймплеєм. Гравець вирушає в подорож через генеровані випадковим чином рівні, збираючи ресурси, здобуваючи нові зброї і здібності, і вступаючи в бої проти різноманітних ворогів. Успішне виконання завдань дозволяє гравцеві розблоковувати нові області і покращувати свого персонажа. Відсутність "постійної смерті" додає грі особливий елемент виживання. По факту чим далі гравець заходить і чим частіше він програє тим сильніше стає [4].

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

Була розроблена французькою компанією Motion Twin і видана у серпні 2018 року. Ця гра була реалізована на мові програмування C# та використовує рушій Unity для своєї реалізації.

Завдяки гнучкому геймплею, гра стала популярною серед шанувальників аркадних ігор та ретро-стилю. "Dead Cells" відображає унікальний підхід до жанру і підтримується активною спільнотою, що створює моди й розширення для подальшого насолодження від гри.

Завдяки поєднанню мови програмування C# та движка Unity, розробники Dead Cells могли створити захоплюючу комп'ютерну гру з вражаючими графічними ефектами, цікавою геймплейною механікою та захоплюючим ігровим світом. Використання C# дозволило легко реалізувати складні ігрові механіки та системи, тоді як потужний інструментарій Unity сприяв швидкому та ефективному створенню візуальних і аудіоелементів гри.

Переваги:

- Різноманітність геймплею: рандомізовані рівні та широкий вибір зброї і умінь надають глибину геймплею і високий рівень переігровуваності.
- Інтенсивний бойовий досвід: швидкий та експлуатуючий бойовий стиль, що нагадує ретро-ігри, викликає відчуття адреналіну.
- Естетика: унікальний мистецький стиль та прекрасно виконана анімація.
- Постійні оновлення: регулярні оновлення від розробників додають новий контент та виправляють помилки, що підтримує інтерес до гри.

Недоліки:

- Висока складність: для деяких гравців висока випадковість та складність гри можуть бути неприйнятними.
- Відсутність глибокої сюжетної лінії: гра фокусується на геймплеї, а не на сюжеті, що може розчарувати тих, хто шукає глибокий сюжет.
- Неявна інструкція: гравцям може бути важко зрозуміти механіки гри без додаткової інформації або тренування.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

- Обмежена варіативність оточення: деякі гравці можуть відчувати повторюваність оточення після тривалого ігрового часу.

2. Hollow Knight (Рис 1.2) :



Рисунок 1.2 – Комп'ютерна гра Hollow Knight

"Hollow Knight"— це метроїдванія-платформер, розроблений Team Cherry, що вийшов у лютому 2017 року. Гра поєднує в собі глибокий метроїдванійний геймплей з мистецтвом ручної малюнківної анімації і вражаючими художніми пейзажами [5].

У "Hollow Knight" гравці відправляються в подорож у таємниче підземне королівство Hallownest, де вони зустрічають різноманітних персонажів, розгадують загадки і борються з численними ворогами. Головний герой, безіменний лицар-клопотень, має можливість вдосконалювати свої здібності, збираючи валюту і виявляючи нові області.

Гра привернула увагу своїм мистецтвом і заглибленим геймплеєм, що вимагає від гравця вправності і стратегічного мислення. "Hollow Knight" отримав позитивні відгуки критиків і широко визнаний як одна з кращих метроїдваній від аудиторії.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		14

Переваги:

- Прекрасний світ: унікальний та захоплюючий світ, заповнений таємницями та прекрасним мистецтвом.
- Глибокий сюжет: інтригуючий сюжет та персонажі, які викликають емоції, створюють незабутній досвід.
- Геймплейна глибина: різноманітність геймплею, включаючи бої, головоломки та дослідження, надає грі багатогранність.

Недоліки:

- Висока складність: деякі гравці можуть відчувати виклик у подоланні складних боїв та загадок.
- Повільний темп: для деяких гравців може виявитися повільним темп гри, особливо в початкових рівнях.
- Завантаження: час завантаження між областями може бути тривожним для тих, хто шукає плавний потік гри.

3. Celeste (Рис 1.3):



Рисунок 1.3 – Комп'ютерна гра Celeste

"Celeste" (Рис 1.3) — це інді-платформер, розроблений і виданий Maddy Makes Games у січні 2018 року. Гра здобула популярність завдяки своїй унікальній сюжетній лінії і інтенсивній механіці гри [6].

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		15

У "Celeste" гравці керують головною героїнею по імені Медлін, яка прагне підкорити гору під назвою Celeste. Гра сполучає в собі класичні платформерні елементи зі складними головоломками та надзвичайно вимогливими рівнями.

Одним із ключових аспектів "Celeste" є його акцент на ментальному здоров'ї та особистих досягненнях. Гра використовує свою історію для відображення боротьби з депресією та самотійним піднесенням духу через перешкоди.

Гра вийшла на платформах PC, Nintendo Switch, PlayStation та Xbox, і вона була відзначена як одна з найкращих ігор 2018 року за її мистецьку варіацію та глибину сюжету.

Гра "Celeste" була розроблена на двигуні MonoGame/XNA з використанням мови програмування C#. MonoGame є відкритою реалізацією фреймворку XNA, який надає інструменти для створення інді-ігор на платформі Microsoft Windows, Xbox і Windows Phone.

MonoGame/XNA дозволяє розробникам створювати ігри з високою продуктивністю та кросплатформенною підтримкою, що дозволило студії Maddy Makes Games ефективно реалізувати свої творіння для різних платформ, включаючи PC, консолі Nintendo, PlayStation і Xbox.

Переваги:

- Відмінна історія: глибокий сюжет та добре розвинуті персонажі надають грі емоційну глибину.
- Геймплейна механіка: різноманітність геймплею та інтенсивний геймплей створюють захоплюючий досвід.
- Виклик: висока складність гри дозволяє гравцям відчувати задоволення від досягнень.

Недоліки:

- Висока складність: для деяких гравців висока складність гри може бути викликом.
- Монотонність: для деяких гравців гра може стати монотонною через постійне повторення випробувань.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		16

- Невеликий розмір гри: для деяких гравців гра може здатися занадто короткою в порівнянні з іншими іграми.
- Висока вимогливість до апаратного забезпечення: гра може вимагати потужного комп'ютера для плавної роботи.

В доповнення:

Unity — це кросплатформений рушій для створення ігор і програмного забезпечення, що підтримує створення ігор для різних платформ, таких як PC, консолі (PlayStation, Xbox, Nintendo Switch), мобільні пристрої (Android, iOS) та інші. Він відомий своєю гнучкістю, широким набором інструментів і зручним інтерфейсом, що дозволяє розробникам створювати як 2D, так і 3D ігри.

В Hollow Knight команда розробників використовувала Unity для створення його унікального арт-стилю, складної системи управління персонажем і інтерактивного світу гри. Unity дозволив студії створити деталізовані 2D анімації та взаємодію об'єктів, що відтворює глибину та іммерсивність "Hollow Knight".

А в Dead Cells за допомогою Unity гра поєднує в собі швидкі темпи, динамічні бої та випадково генеровані рівні. Unity дозволив реалізувати складну систему бою, анімацій та фізичних ефектів, які забезпечили глибокий і інтенсивний геймплей "Dead Cells".

C# — це мова програмування, розроблена Microsoft, яка поєднує в собі потужність C++ з простотою використання Java. Вона є однією з популярних мов для розробки ігор, додатків, веб-сайтів та різноманітних програмних рішень.

Основні особливості C#:

1. Синтаксис і ООП: C# базується на об'єктно-орієнтованому програмуванні (ООП), що дозволяє створювати класи, об'єкти, інтерфейси та інші структури для організації коду що дуже сильно спрощує розробку таких проектів як гра.
2. Манагемент пам'яті: C# має вбудовану систему збору сміття (Garbage Collection), яка автоматично вивільняє пам'ять від непотрібних об'єктів, що полегшує розробку і уникнення помилок з утриманням пам'яті.

3. Платформенна незалежність: мова C# підтримує кросплатформенність, що означає, що програми, написані на C#, можуть запускатися на різних операційних системах без значних змін в коді. Завдяки цій особливості «Dead Cells» без проблем змогла вийти на мобільний ринок.

У контексті розробки ігор, C# є популярним вибором завдяки своїй продуктивності, широким можливостям і підтримці таких інструментів, як Unity, що забезпечують ефективну розробку ігор для різних платформ.

Розглянемо загальні відомості про наведенні аналоги.

Таблиця 1.1

Таблиця порівняння платформерів

	Dead Cells	Hollow Knight	Celeste
Висока складність	8	7	9
Монотонність	7	4	6
Відчуття досягнень	9	9	7
Різноманітність геймплею	8	9	5
Інтенсивний геймплей	10	9	9
Сюжет	4	10	5
Цікавий світ	7	10	7
Саундтрек	5	10	8

В таблиці оцінювання (Таблиця 1.1) було використано 10-бальну шкалу де 1 це найгірший результат, а 10 це найкращий.

Аналізуючи всі аспекти ігор Dead Cells, Hollow Knight і Celeste, ми можемо ідентифікувати найкращі функції та можливості, щоб включити їх у власний продукт. Ось кілька варіантів:

- Різноманітність геймплею: включення різноманітних рівнів та завдань, які стимулюють дослідження та використання навичок гравців.

- Інтенсивний геймплей: забезпечення динамічної та енергійної гри зі швидкими відчуттями, які присутні в Dead Cells і Celeste.
- Висока складність: додавання викликів та вдалих досягнень, щоб стимулювати гравців та підвищувати повторний геймплей.

Краще зконцентруватися на окремих аспектах таких як різноманітність геймплею, його динамічність та складність.

1.3 Вибір та обґрунтування інструментальних засобів для розробки

При розробці будь-якого ігрового проекту ключовим є вибір технологічної платформи, яка забезпечить найвищу якість, швидкість розробки і підтримку для максимальної аудиторії гравців та розробників. Такою платформою є Unity.

Розглянемо всі ключові аспекти, необхідні в розробці нашого проекту:

- Візуальний редактор і компонентна архітектура: інтуїтивно зрозумілий інтерфейс для створення і керування об'єктами гри.
- Графічні можливості і підтримка фізичного моделювання: потужні інструменти для створення високоякісної графіки і реалістичних фізичних взаємодій.
- Аудіо-двигун і система анімацій: вбудовані засоби для інтеграції звуку та анімацій, що дозволяють створювати живі та інтерактивні ігрові світи.
- Широкий вибір ресурсів і підтримка спільноти: велика активна спільнота розробників і безліч ресурсів для навчання і підтримки.
- Швидкість розробки і ефективне управління ресурсами: прискорення процесу розробки завдяки готовим рішенням, таким як префаби, і інструментам для управління ресурсами.

Ці аспекти роблять Unity ідеальним вибором для створення як малих інді-проектів, так і великих комерційних ігор, надаючи всі необхідні інструменти для успішного розвитку ігрових проектів.

Unity має кілька додаткових особливих функцій, які роблять його ще привабливішим для розробників.

Додаткові особливі функції Unity:

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		19

- Unity Asset Store це ресурс, де розробники можуть купувати або завантажувати безкоштовні моделі, текстури, скрипти, плагіни та інші ресурси. Це значно прискорює процес розробки, дозволяючи використовувати готові рішення і зосередитися на аспектах гри.
- Cinemachine є інструментом для створення динамічних камер. Він дозволяє легко налаштовувати і керувати камерою в реальному часі.
- Інструмент Performance Reporting дозволяє відслідковувати і аналізувати продуктивність гри, виявляючи помилки і проблеми продуктивності в реальному часі.

Ці функції надають ще більший пріоритет у виборі, оскільки вони є доволі простими у використанні.

Мова програмування C# — це сучасна, об'єктно-орієнтована мова програмування. Вона була створена як частина платформи .NET і призначена для розробки різноманітних додатків, включаючи в це і комп'ютерні ігри.

Основні особливості C#

- Модернізовані синтаксичні конструкції: C# пропонує сучасний і зручний синтаксис, що полегшує написання коду. Мова включає в себе такі конструкції, як лямбда-вирази, анонімні методи, асинхронні методи (async/await) і багато іншого.
- Об'єктно-орієнтована: C# підтримує всі основні концепції об'єктно-орієнтованого програмування (ООП), такі як інкапсуляція, наслідування, поліморфізм і абстракція. Це дозволяє розробникам створювати гнучкі та масштабовані програми.
- Інтеграція з Unity: C# є основною мовою програмування для Unity, одного з найпопулярніших рушіїв для розробки ігор. Використання C# у Unity забезпечує високу продуктивність і зручність розробки ігрових логік, графічних інтерфейсів, анімацій і фізичних взаємодій.
- Розширюваність і модульність: завдяки можливостям C# дозволяє створювати модульні програми, які легко підтримувати та розширювати.

Використання C# у розробці ігор з Unity:

Unity використовує C# як основну мову для скриптингу. Це означає, що весь ігровий код, логіка поведінки персонажів, обробка подій і інтеракцій між об'єктами пишуться на C#. Ось кілька причин, чому C# ідеально підходить для розробки ігор на Unity:

- Простота і зручність: C# має зрозумілий і легкий для вивчення синтаксис, що дозволяє швидко освоїти основи і приступити до розробки.
- Потужні інструменти розробки: існує безліч інтегрованих середовищ розробки (IDE), таких як Visual Studio, які підтримують C# і надають розширені можливості для дебагінгу, тестування і оптимізації коду.
- Велика спільнота: спільнота розробників на C# дуже активна і надає безліч ресурсів для навчання, включаючи документацію, форуми, туторіали і приклади коду.
- Багата бібліотека функцій: C# має величезну стандартну бібліотеку класів, яка надає безліч готових рішень для обробки даних, роботи з мережею, створення графічних інтерфейсів та інших задач.

Таким чином, C# є потужною і гнучкою мовою програмування, яка забезпечує високу продуктивність і зручність розробки, що робить її ідеальним вибором для створення ігор на платформі Unity.

Вибір Visual Studio як середовищ розробки коду :

Visual Studio є однією з найпопулярніших інтегрованих середовищ розробки (IDE) для програмування на C#. Вибір Visual Studio для розробки ігор на Unity з використанням C# має багато переваг. Visual Studio має повну інтеграцію з Unity, що забезпечує зручний робочий процес. Завдяки плагіну Visual Studio Tools for Unity, розробники можуть легко налаштовувати та дебагувати свої проекти. Visual Studio надає розширені можливості для редагування коду, включаючи інтелектуальне автозавершення, підсвічування синтаксису та рефакторинг коду. Потужні інструменти для налагодження дозволяють легко знаходити і виправляти помилки в коді. Також підтримується розширюваність через плагіни, що дозволяє додавати нові функції та інструменти для задоволення специфічних потреб

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

проекту. Ці переваги роблять Visual Studio потужним і зручним інструментом для розробки ігор на Unity з використанням C#.

Висновки до першого розділу

У першому розділі було сформульовано основну мету цієї кваліфікаційної роботи: розробка комп'ютерної гри у жанрі 2D платформера. В рамках аналізу було оглянуто такі ігри, як Dead Cells, Hollow Knight та Celeste, для виявлення ключових особливостей і функціоналу, що визначили вимоги до розробки нашої гри, зокрема, динамічний та інтенсивний геймплей.

Для обґрунтування вибору інструментальних засобів та вимог до апаратного забезпечення було обрано платформу Unity для розробки гри з використанням мови програмування C# та інструменту Visual Studio. Цей вибір надає необхідні інструменти для створення і оптимізації гри, а також забезпечує ефективне використання ресурсів.

Загалом, перший розділ визначає основні параметри і напрямки подальшої роботи над проектом, створюючи основу для успішної реалізації 2D платформера.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ГРИ

2.1 Визначення варіантів використання та розробка ігрового додатку

Мета гри: дійти до кінця рівня уникаючи всіх пасток і по шляху зібрати всі кавуни.

Геймплей: основна ігрова механіка — це рух гравця, на якому зав'язана вся гра. Персонажу потрібно уникати всіх пасток на рівні, таких як статичні шипи або рухомі пили, і дістатися фінішу, щоб перейти на наступний рівень. Рівні спроектовані так, щоб максимально використовувати рух гравця, через що гра має високу складність. Перед кожною секцією з пастками знаходиться чекпоінт, і в разі невдачі рівень не буде починатися знову — ігровий персонаж просто буде респавнитись на останньому збереженому чекпоінті.

Рух головного персонажа прив'язаний до кнопок WASD, де кнопки A і D відповідають за напрямок руху або напрямок ривка. Клавіші W і S відповідають за напрямок ривка. За ривок відповідає кнопка LShift: при її натисканні персонаж буде робити різке переміщення в бік, який вказує клавіша WASD або комбінація з двох клавіш. Наприклад, якщо тримати клавіші W і D під час ривка, персонаж переміститься по діагоналі вправо.

Кнопка Space відповідає за стрибок. Якщо натиснути її в повітрі після першого стрибка, персонаж відштовхнеться в повітрі і стрибне ще раз. Якщо стрибнути на стіну і затримати клавішу A або D (залежно від того, з якого боку знаходиться стіна), персонаж буде ковзати по стіні. Якщо в цей момент натиснути стрибок, персонаж відштовхнеться від стіни.

Діаграма варіантів використання— це графічне зображення взаємодій між користувачами та системою, що відображає різні сценарії використання системи. Вона використовується для моделювання функціональних вимог, демонструючи, як різні типи користувачів взаємодіють з системою для досягнення певних цілей.

Основними елементами діаграми варіантів використання є:

1. Актори: зовнішні користувачі або інші системи, які взаємодіють з системою. Вони зображуються у вигляді фігурок.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		23

2. Варіанти використання: окремі функції або дії, які система виконує у відповідь на дії акторів. Вони зображуються у вигляді овалів або кіл.
3. Зв'язки: лінії, що з'єднують акторів з варіантами використання, показуючи, які актори використовують які функції системи.

Ці діаграми допомагають визначити та структурувати функціональні вимоги до системи, надаючи зручний спосіб для спілкування між розробниками та іншими зацікавленими сторонами.

Створену діаграму варіантів використання можна побачити на рисунку 2.1.

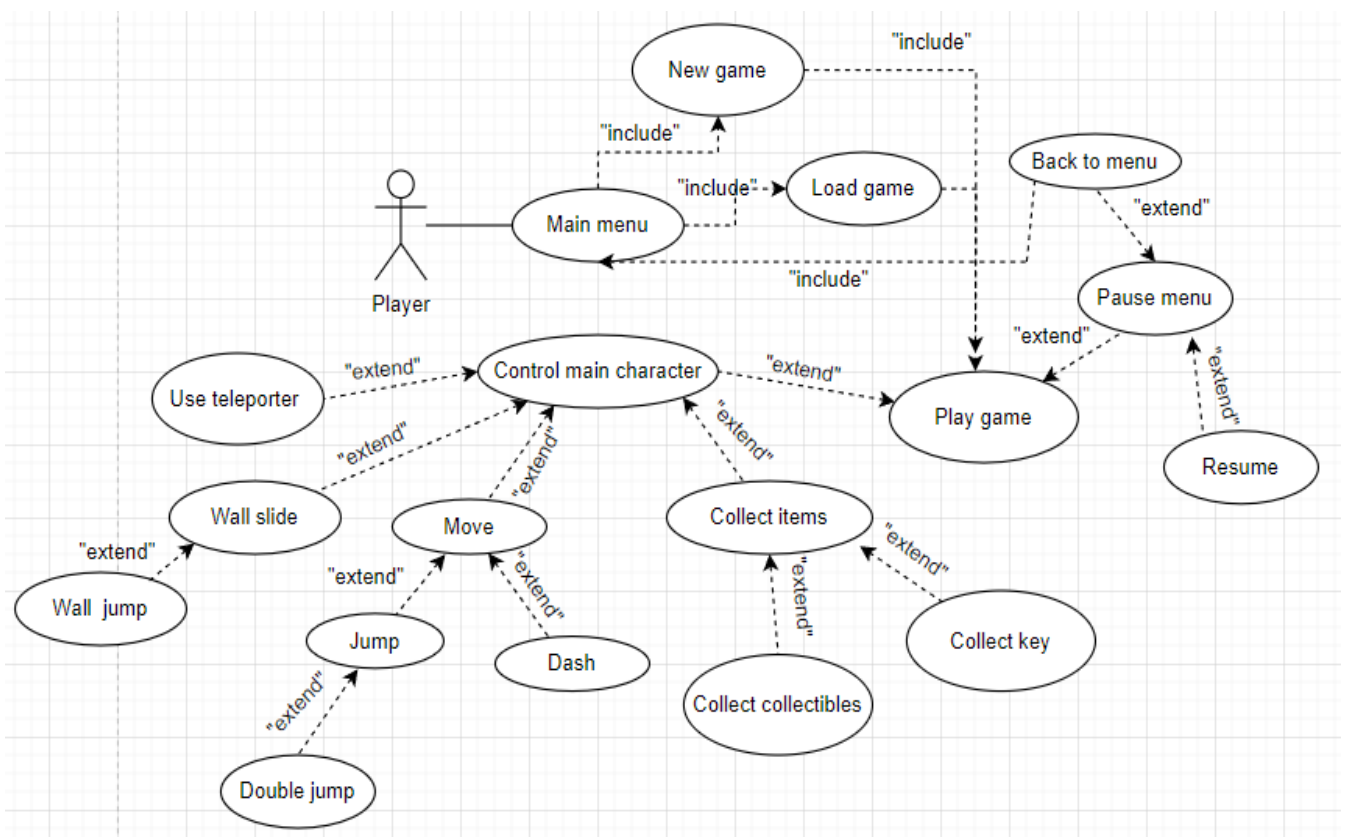


Рисунок 2.1 – Діаграма використання

У цій діаграмі зображено перелік дій, які може виконувати актор (Player). При запуску гри перше, що бачить користувач, це меню, в якому він може розпочати нову гру або продовжити стару, якщо така була. Після завантаження сцени з рівнем у гравця є можливість керувати персонажем, а саме: рухатися в сторони, стрибати, робити ривок та ковзати по стінах, робити подвійний стрибок. Також він може підбирати колекційні предмети, а на другому рівні для його проходження треба зібрати 3 ключі.

Діаграма класів (рис.2.2) є структурним діаграмою UML, яка описує структуру системи, показуючи її класи, атрибути, методи та взаємозв'язки між класами. Діаграма класів допомагає зрозуміти, як елементи програмного забезпечення взаємодіють між собою, а також які дані та поведінка зберігаються в кожному класі.

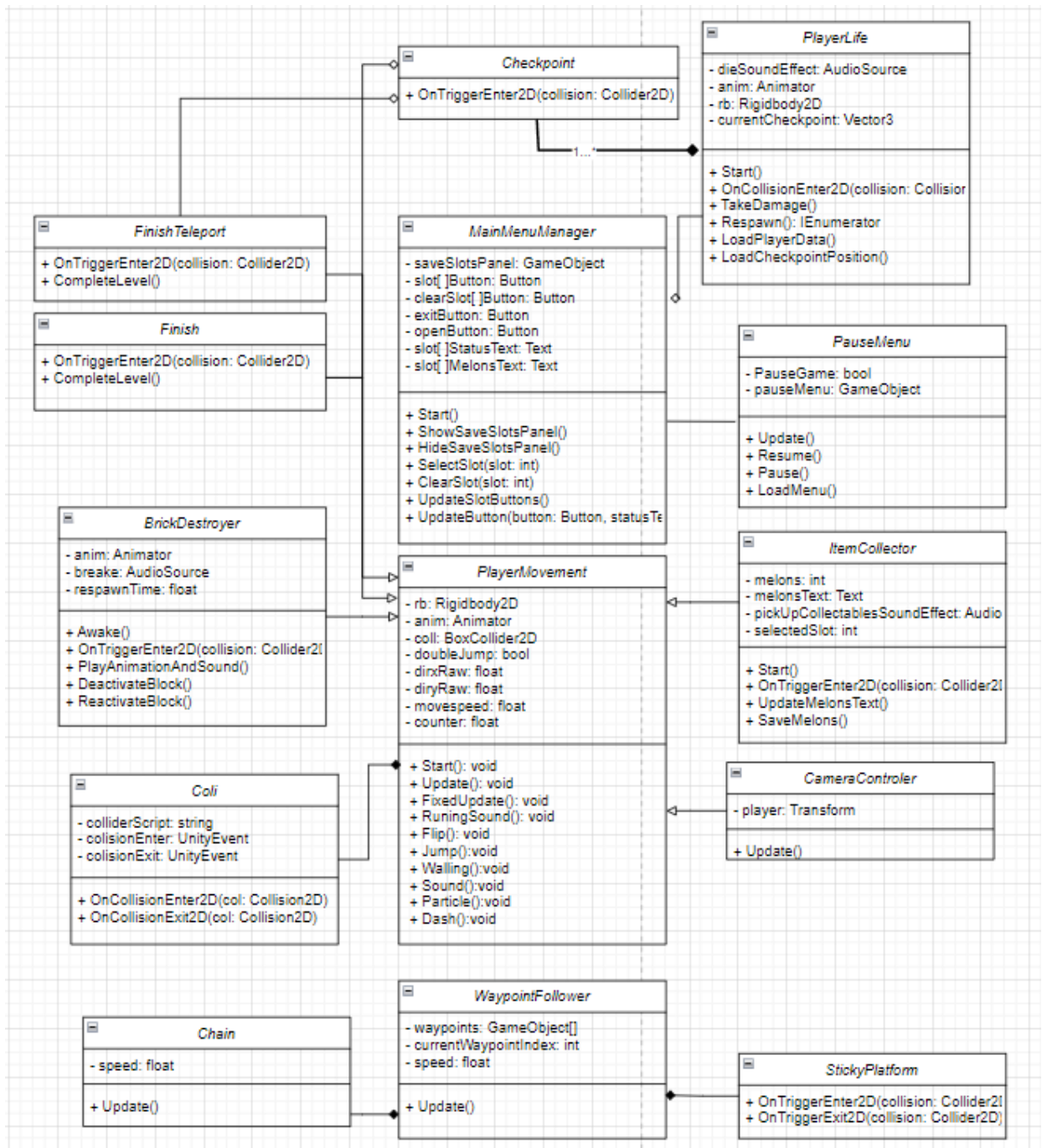


Рисунок. 2.2 – Діаграма класів

1. PlayerLife: клас відповідає за управління життям гравця. Він обробляє зіткнення з пастками, програвє звукові ефекти смерті та запускає

відповідну анімацію. Після смерті гравця ініціюється процес респауну на останній контрольній точці.

2. ItemCollector: клас відповідає за збирання колекційних предметів, таких як мелони. Він обробляє підрахунок зібраних предметів, відображає їх кількість на екрані та зберігає цю інформацію.
3. Teleporter: клас управляє телепортами у грі. Він зберігає цільову позицію для телепортації та надає доступ до цієї позиції.
4. BrickDestroyer: клас відповідає за знищення блоків при зіткненні з гравцем. Він програє відповідну анімацію та звуковий ефект, а потім деактивує блок на певний час перед його відновленням.
5. Checkpoint: клас зберігає позицію гравця при проходженні контрольних точок. Ця інформація зберігається для подальшого використання при респауні або завантаженні гри.
6. FinishTeleport: клас викликає завершення рівня і перехід до наступного при досягненні фінішної точки гравцем. Він завантажує наступну сцену через деякий час після досягнення фінішу.
7. Chain: клас управляє обертанням об'єкта (ланцюга). Він обертає об'єкт зі швидкістю, визначеною змінною speed, постійно в кожному кадрі, клас було створенно замість створення анімації обертання.
8. Finish: клас завершує рівень, очищує збережені дані контрольних точок та іншу інформацію, а потім завантажує наступний рівень. Він також обробляє досягнення гравцем фінішної точки.
9. WaypointFollower: клас відповідає за рух об'єкта між вказаними точками. Він переміщує об'єкт зі швидкістю до наступної точки і починає знову з першої точки після досягнення останньої, клас було створенно для рухаючихся платформ та пасток.
10. Coli: клас викликає події при зіткненні з об'єктами, які мають вказаний скрипт. Він викликає відкриття дверей відповідного кольору при зіткненні гравця з ключем при цьому ключ щезає.
11. CameraController: клас управляє позицією камери, слідкуючи за гравцем.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

Він постійно оновлює позицію камери, щоб вона відповідала позиції гравця з певним зсувом.

12. PauseMenu: клас управляє меню паузи гри. Він забезпечує функції для паузи, відновлення гри, а також повернення до головного меню .
13. StickyPlatform: клас забезпечує прикріплення гравця до платформи при зіткненні. Коли гравець взаємодіє з платформою, він прикріплюється до неї і рухається разом з нею до виходу з зони взаємодії.
14. MainMenuManager: клас управляє головним меню гри. Він відповідає за вибір та очищення слотів збереження, а також оновлення статусу слотів на екрані головного меню. Клас також обробляє переходи між головним меню та іншими частинами гри.
15. PlayerMovement: клас "PlayerMovement" відповідає за управління рухами гравця. Він обробляє введення користувача для переміщення гравця ліворуч і праворуч, здійснення стрибків, ривків та ковзання по стінах. Клас також реалізує подвійний стрибок і відстежує стан руху гравця, щоб коректно застосовувати анімації для різних дій.

Цей код демонструє кілька патернів і особливостей програмування в Unity, які варто відзначити. Спочатку це використання MonoBehaviour класів є типовою практикою в Unity для створення поведінки об'єктів у грі. Класи використовують різні компоненти, такі як Rigidbody2D, Animator, AudioSource та Transform, для забезпечення фізики, анімації та звукових ефектів.

Далі, патерн Observer (спостерігач) використовується в методах, що реагують на колізії та тригери, як у класах PlayerLife та BrickDestroyer, де методи викликаються у відповідь на події колізії з об'єктами з певними тегам.

І головне, використання PlayerPrefs для збереження даних гри, таких як положення контрольних точок, кількість зібраних об'єктів та вибрані слоти збереження, демонструє патерн Memento (знімок), який зберігає та відновлює стан гри. Завдяки цьому підходу гравець може безперервно продовжувати гру з того місця, де він завершив або зберіг проходження за власними уподобаннями.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		27

2.2 Розробка дизайну рівня

Гра буде розроблена у форматі лінійного рівня, де головному персонажу потрібно буде досягти кінця, розв'язуючи загадки, уникаючи пасток і збираючи колекційні предмети. Кожен рівень буде містити чекпоінти, на яких персонаж відроджуватиметься у разі смерті.

Перед створенням персонажа, керуваного гравцем, необхідно створити каркас рівня, що складатиметься з так званих плиток (рис. 2.3) . Всі асети в грі будуть 16 або 6 бітними .

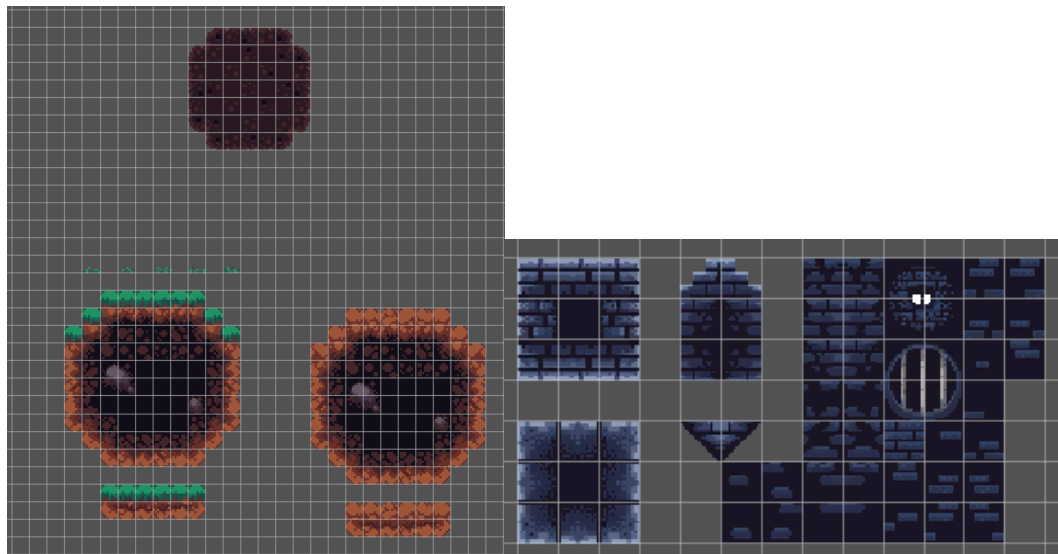


Рисунок. 2.3 - Плитка в вікні Tile Palette

З плиток буде створено основну частину рівня. При їх розміщенні кожна плитка буде встановлюватися на конкретному шарі рівня, таких як:

- Main — це основний шар, на якому розташована основна частина рівня, де гравець може ходити та взаємодіяти з об'єктами.
- MainNoneCollision — це ще один шар, з яким гравець може взаємодіяти, але на цьому шарі відсутня колізія. Наприклад, гравець може стояти на ньому, але не може пройти через нього або залізти вгору. Цей шар використовується як бар'єр, щоб обмежити рух гравця у певних напрямках.
- Background — це фоновий шар, з яким гравець не може взаємодіяти. Він завжди знаходиться позаду гравця та всіх інших шарів і використовується виключно для візуальної частини гри.

- Forntground — це шар, який завжди знаходиться перед усіма іншими шарами. Він також використовується тільки для візуального ефекту і не має впливу на механіку гри.

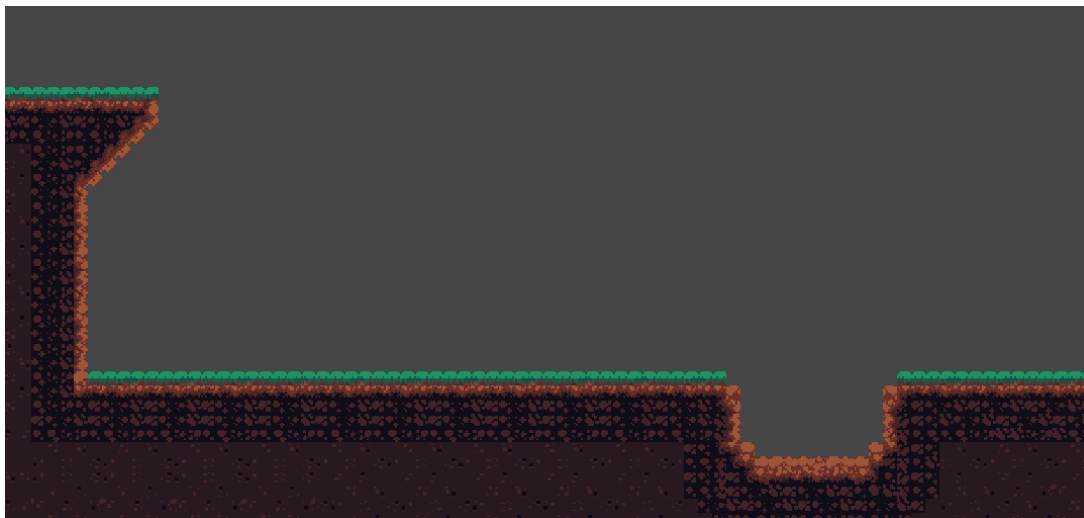


Рисунок. 2.4 – Рівень з розміщеними плитками

Після розміщення плиток на рівні (рис. 2.4), потрібно додати декорації та задній фон.

Декорації: для першого рівня були створені декорації (рис. 2.5), кожна з яких буде розміщуватися на рівні за допомогою префабів. Це дозволяє уникнути постійного копіювання одного і того ж спрайта, оскільки функція префабів забезпечує ефективне використання ресурсів і зручність у редагуванні.

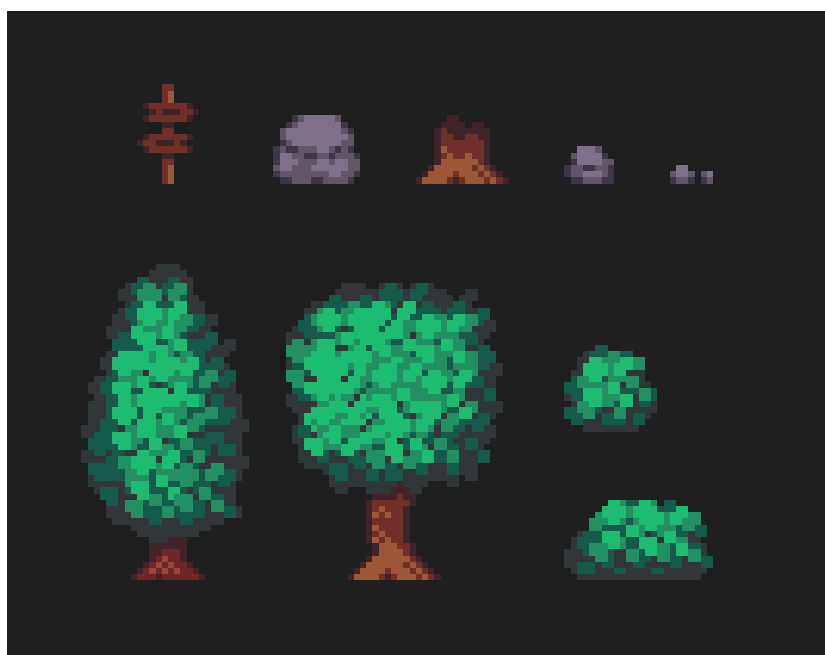


Рисунок. 2.5 – Асети декорацій першого рівня

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

Задній фон буде створений з високоякісної градієнтної картинки (рис. 2.6), яка буде розтягнута на весь рівень.



Рисунок. 2.6 – фонова картинка

Після розміщення всіх декорацій на рівні він виглядає так, як показано на рисунку нижче (рис. 2.7). Після цього можна перетворити використані декорації в префаби, які можна буде постійно використовувати, замість того щоб постійно копіювати об'єкти.

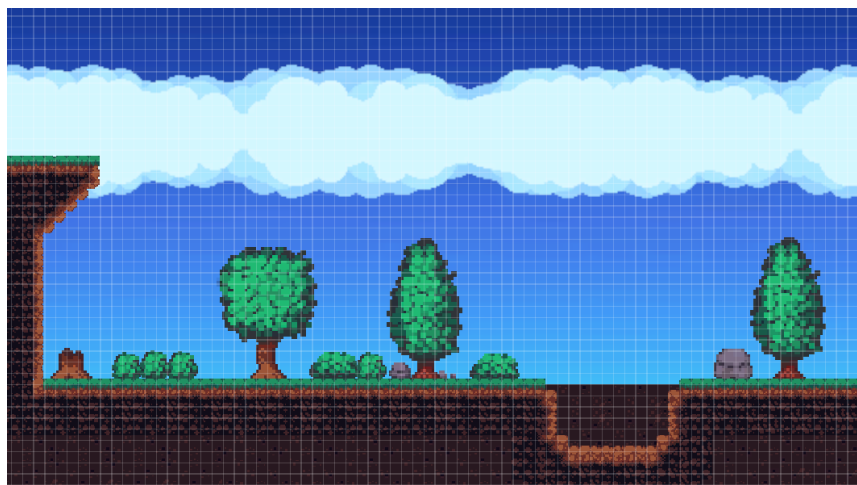


Рисунок. 2.7 – Вигляд першого рівня з розміщеними декораціями

Надалі було створено дизайн персонажу. Спрайт персонажа був доданий на рівень (рис. 2.8).

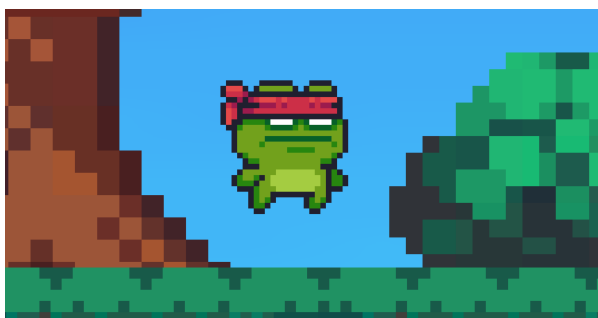


Рисунок. 2.8 – Зовнішній вигляд ігрового персонажу

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		30

Був обраний дизайн та створені спрайти пасток (рис. 2.9), які будуть використовуватися на рівнях. Це будуть єдині об'єкти на карті, які зможуть завдати шкоди головному герою.

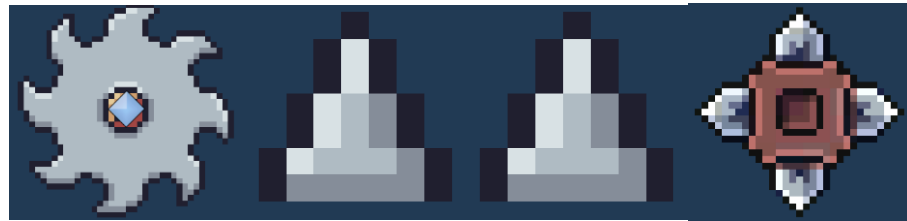


Рисунок. 2.9 – Спрайти пасток

Були створені спрайти рухомих платформ (рис. 2.8), які будуть рухатися від однієї точки до іншої, а гравець зможе рухатися разом з ними. Також були створені спрайти колекційних предметів (рис. 2.10), які гравець зможе збирати за бажанням, але зазвичай вони будуть сховані або розміщені у важкодоступних місцях.



Рисунок. 2.10 – Спрайт платформи та предмета

Для другого рівня були створені ще три об'єкти: телепорти, падаючі блоки (рис. 2.11) та важкі двері (рис. 2.12) з ключами у вигляді дорогоцінного каміння.



Рисунок. 2.11 – Спрайт телепорту та падаючого блоку

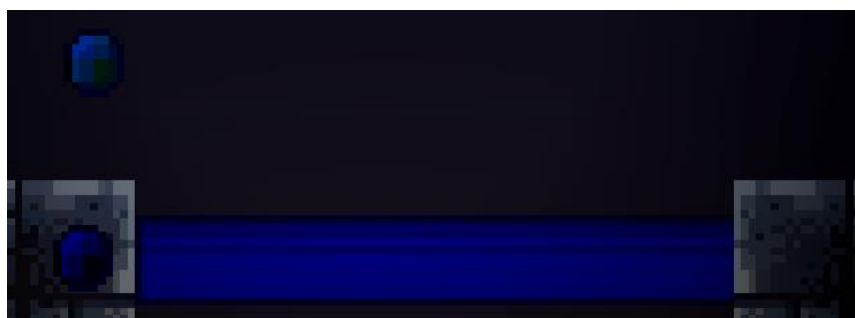


Рисунок. 2.12 – Спрайт тяжких дверей

Для другого рівня було створено освітлення (рис. 2.13), яке було налаштовано та розташовано по всьому рівні. Також було створено невелику кількість спрайтів декорацій (рис. 2.14), які були розташовані по рівню..

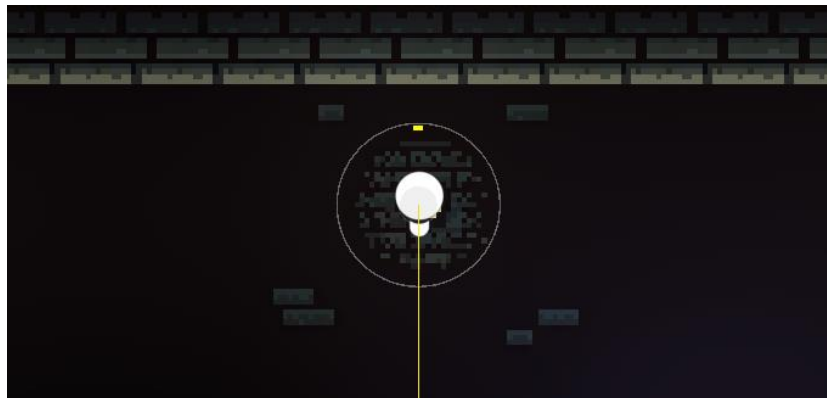


Рисунок. 2.13 – Відображення лампи в редакторі юніті



Рисунок. 2.14 – Декорації другого рівня

Всі ассети було збережено по папкам в проекті, і до деяких, наприклад, пасток, буде додано скрипти. В цілому, завдяки використанню префабів, створення інших рівнів буде набагато легше.

2.3 Опис створення анімацій

Анімації в Unity є важливою складовою будь-якої ігрової чи інтерактивної додаткової розробки. Вони не лише додають реалізму і живості ваших ігор, але й дозволяють керувати поведінкою об'єктів, створюючи плавні переходи між різними станами та взаємодіючими елементами [7].

Анімації в Unity є механізмом, який дозволяє змінювати параметри об'єктів з часом. Це може включати переміщення, обертання, зміну розміру, зміну колірних властивостей і навіть звукові ефекти. Основна ідея полягає в тому, щоб створити плавні переходи між різними станами об'єкта, щоб відтворити природний рух або інші ефекти [7].

Для створення анімацій для двовимірних об'єктів в Unity використовуються наступні ключові компоненти:

1. Спрайти (Sprites): зображення, які відображають кожен кадр анімації. Спрайти можуть представляти різні позиції, вигляди або стани об'єкта.
2. Аніматори (Animator): компоненти, які керують логікою анімації об'єктів. Аніматори встановлюють, які спрайти відображати в певний момент часу і які переходи виконувати між різними станами.
3. Анімаційні контролери (Animation Controllers): файли, які містять логіку переходів між анімаційними станами та управлінням анімаційними параметрами. Вони об'єднують різні анімаційні стани в один потік керування.
4. Ключові кадри (Keyframes): спеціальні точки в часі, де визначені значення параметрів об'єктів для створення ефектів анімації. Ключові кадри вказують, як змінюються властивості об'єкта з часом, і Unity інтерполює між цими кадрами для створення плавного ефекту руху або зміни.

Анімації в Unity дозволяють створювати динамічні та привабливі ігрові інтерфейси, анімовані персонажі, спеціальні ефекти і багато іншого. Їх створення є важливою складовою для розробки як двовимірних, так і тривимірних ігор.

Першим і головним завданням було розроблення анімацій для гравця. Для нього було створено сім різних анімацій: анімацію спокою, анімацію бігу, ковзання по стіні, стрибка, падіння та смерті. Всі ці анімації були об'єднані в анімаційному контролері. Після цього в аніматор контролері були налаштовані переходи між різними анімаціями (рис.2.15).

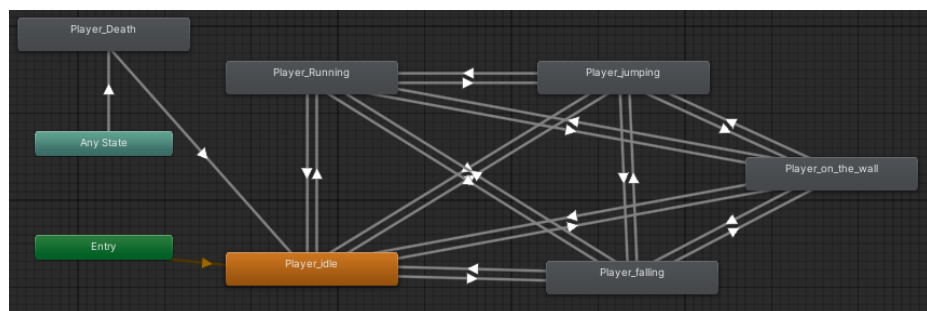


Рисунок. 2.15 – Відображення всіх анімацій персонажу в вікні аніматора

У коді використовується перерахування MovmentState, яке відображає різні стани анімацій гравця (рис.2.16). Коли відбувається певна дія, така як стрибок, метод Jump() передає відповідну цифру для стану стрибка в метод UpdateAnimationState().

Ось як виглядає код методу оновлення стану анімації:

```
private void UpdateAnimationState()
{
    MovmentState state;
    if (dirxRaw > 0f || dirxRaw < 0f || rb.velocity.x !=0)
    {
        state = MovmentState.running;
    }
    else
    {
        state = MovmentState.idle;
    }
    if (rb.velocity.y > .1f)
    {
        state = MovmentState.jumping;
    }
    else if (rb.velocity.y < -.1f)
    {
        state = MovmentState.falling;
    }
    if (isWalltouch() && isWallSliding)
    {
        state = MovmentState.on_the_wall;
    }
    if (isWallJumping)
    {

```

```

state = MovmentState.jumping;
anim.SetInteger("state", (int)state);
}

```

Всі анімації окрім смерті постійно передаються через цей метод.



Рисунок. 2.16 – Анімація падання, стрибка, бігу та покою

До анімацій гравця були додані частинки (рис.2.17)., які значно збагатили візуальну привабливість гри. Ці частинки використовувалися для різних ефектів, таких як ривок, біг, ковзання по стіні та подвійний стрибок. Кожен з цих ефектів мав свої власні часточки, які були створені з одного спрайту частинок, що відтворювалися кілька разів для створення відчуття руху і динаміки.

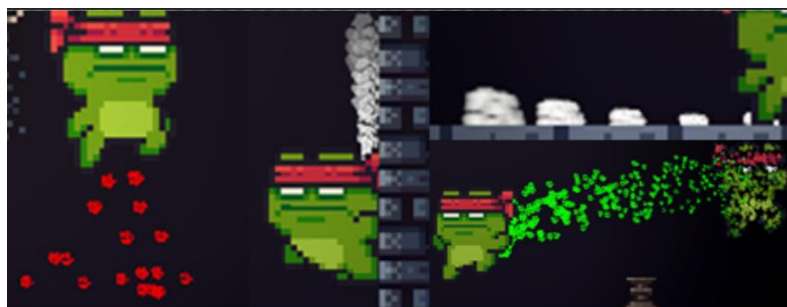


Рисунок. 2.17 – Часточки які супроводжують дії гравця

Окрім анімацій гравця, були створені інші анімації, такі як руйнування падаючих блоків (рис.2.18), відкриття важких дверей з ключем, а також звичайних дверей. Маленькі деталі, такі як обертання гайки під час руху платформи, також були візуалізовані за допомогою анімацій.

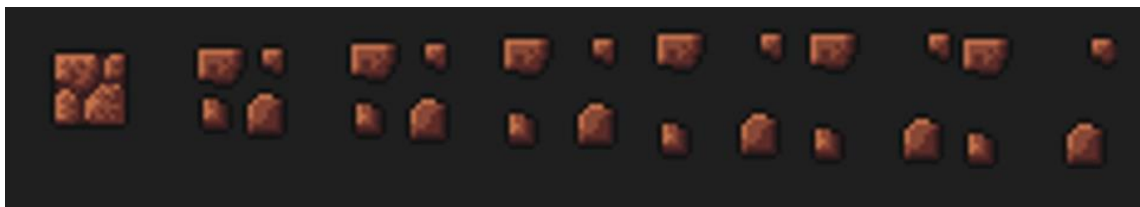


Рисунок. 2.18 –Покрокова анімація падаючого блоку

Ці ефекти додають ігровому середовищу глибину і реалізму, роблячи ігровий світ більш живим і взаємодійним для гравця.

2.4 Реалізація функціоналу гри

Головною метою цього розділу є виявлення та розкриття технічних аспектів розробки гри, зокрема управління персонажем, реалізація взаємодій з ігровим оточенням, обробка колізій, управління станами гравця та ігрових об'єктів, а також збереження та відновлення прогресу гри.

Основною механікою гри є переміщення персонажу, в проекті є клас для реалізації руху гравця “PlayerMovement” (рис.2.19) він відповідає за все переміщення персонажу тобто: стрибок, подвійний стрибок рух в сторони ривок та ковзання по стіні .

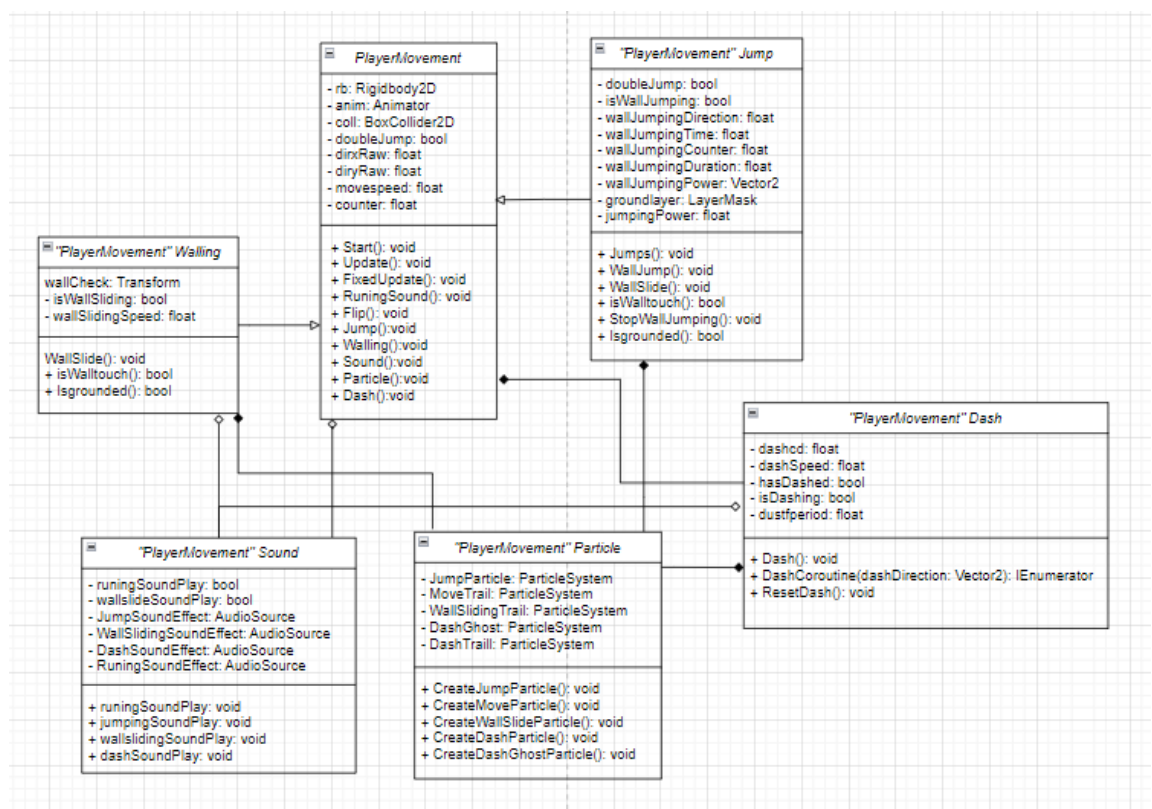


Рисунок. 2.19 – Діаграма методів класу “PlayerMovement”

Далі розглянемо опис кожного використаного методу в класі “PlayerMovement”.

1) Start()

Метод Start викликається один раз під час ініціалізації скрипта. Він використовується для налаштування початкового стану компонентів:

- `rb = GetComponent<Rigidbody2D>():` отримує компонент `Rigidbody2D` персонажа, який використовується для керування фізикою.

- `coll = GetComponent<BoxCollider2D>():` отримує компонент `BoxCollider2D`, що використовується для виявлення колізій з іншими об'єктами.
- `anim = GetComponent<Animator>():` отримує компонент `Animator`, який керує анімаціями персонажа.

2) Update()

Метод `Update` викликається кожен кадр і містить основну логіку для керування рухом, стрибками, ривками та оновленням анімаційного стану персонажа:

- `float x = Input.GetAxis("Horizontal");` отримує значення осі "Horizontal" для керування горизонтальним рухом.
- `float y = Input.GetAxis("Vertical");` отримує значення осі "Vertical" для керування вертикальним рухом.
- `dirxRaw = Input.GetAxisRaw("Horizontal");` отримує сире значення осі "Horizontal".
- `diryRaw = Input.GetAxisRaw("Vertical");` отримує сире значення осі "Vertical".
- Виклик методів `Jumps()`, `RuningSound()`, `WallSlide()`, `WallJump()`, `UpdateAnimationState()` для відповідних дій.
- `Flip():` змінює напрямок, в якому дивиться персонаж, якщо необхідно.
- Перевірка, чи був натиснутий ривок і чи можливо виконати ривок, з подальшим викликом методу `Dash()`.
- Перевірка, чи персонаж знаходиться на землі і чи було використано ривок для скидання прапорця `hasDashed`.

3) FixedUpdate()

Метод `FixedUpdate` викликається з фіксованою частотою, що робить його підходящим для фізичних обчислень:

- Керування горизонтальним рухом персонажа, якщо він не стрибає зі стіни і не робить ривок.
- Виклик методів для створення частинок руху, ковзання по стіні та ривка.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		37

4) isWalltouch()

Метод isWalltouch перевіряє, чи торкається персонаж стіни:

- Використовує Physics2D.OverlapBox для перевірки наявності об'єкта з шаром groundlayer поруч з персонажем.

5) WallSlide()

Метод WallSlide керує ковзанням по стіні:

- Перевіряє, чи торкається персонаж стіни і чи не знаходиться він на землі, та чи є рух у горизонтальному напрямку.
- Встановлює прапорець isWallSliding та обмежує вертикальну швидкість персонажа.
- Відтворює або зупиняє звук ковзання по стіні залежно від стану.

6) Jumps()

Метод Jumps керує стрибками та подвійними стрибками:

- Перевіряє, чи персонаж знаходиться на землі та чи була натиснута кнопка стрибка.
- Встановлює вертикальну швидкість персонажа для виконання стрибка.
- Відтворює частинки стрибка та звук стрибка.
- Керує логікою подвійного стрибка.

7) UpdateAnimationState()

Метод UpdateAnimationState оновлює стан анімації персонажа:

- Визначає стан персонажа на основі його горизонтального та вертикального руху, а також взаємодії з стінами та землею.
- Встановлює відповідний стан анімації у компоненті Animator.

8) Isgrounded()

Метод Isgrounded перевіряє, чи знаходиться персонаж на землі:

- Використовує Physics2D.BoxCast для перевірки наявності об'єкта з шаром groundlayer під персонажем.

9) WallJump()

Метод WallJump керує стрибками зі стін:

- Перевіряє, чи персонаж ковзає по стіні.

- Встановлює напрямок та силу стрибка зі стіни.
- Викликає метод StopWallJumping через певний час для зупинки стрибка зі стіни.

10) StopWallJumping()

Метод StopWallJumping зупиняє стрибок зі стіни:

- Встановлює прапорець isWallJumping у false.

11) Flip()

Метод Flip змінює напрямок, в якому дивиться персонаж:

- Перевіряє, чи необхідно змінити напрямок на основі горизонтального руху.
- Змінює масштаб персонажа по осі X для відображення у відповідному напрямку.

12) Dash()

Метод Dash керує виконанням ривка персонажа:

- Відтворює звук ривка.
- Встановлює прапорець hasDashed у true.
- Викликає метод DashCoroutine для керування ривком.

13) DashCoroutine(Vector2 dashDirection)

Метод DashCoroutine є корутиною, яка керує ривком персонажа:

- Встановлює прапорець isDashing у true.
- Встановлює швидкість персонажа для ривка.
- Виконує паузу на час відкату ривка.
- Викликає метод ResetDash для скидання стану ривка.

14) ResetDash()

Метод ResetDash скидає стан ривка:

- Встановлює прапорець isDashing у false.
- Скидає прапорець hasDashed, якщо персонаж знаходиться на землі.

15) CreateJumpParticle()

Метод CreateJumpParticle створює частинки для стрибка:

- Відтворює частинки стрибка, якщо персонаж не знаходиться на землі.

16) CreateMoveParticle()

Метод CreateMoveParticle створює частинки для руху:

- Відтворює частинки руху з певним інтервалом часу, якщо персонаж рухається і знаходиться на землі.

17) CreateWallSlideParticle()

Метод CreateWallSlideParticle створює частинки для ковзання по стіні:

- Відтворює частинки ковзання по стіні з певним інтервалом часу, якщо персонаж торкається стіни.

18) CreateDashParticle()

Метод CreateDashParticle створює частинки для ривка:

- Відтворює частинки ривка з певним інтервалом часу, якщо персонаж виконує ривок.

19) CreateDashGhostParticle()

Метод CreateDashGhostParticle створює частинки для ефекту під час ривка:

- Відтворює частинки ефекту примари під час ривка.

20) RuningSound()

Метод RuningSound відтворює звуковий ефект для бігу:

- Відтворює звук бігу, якщо персонаж рухається горизонтально, знаходиться на землі і не торкається стіни.
- Зупиняє звук бігу, якщо персонаж перестав рухатись.

Клас "PlayerMovement" реалізує комплексну систему керування рухом персонажа у 2D платформері, включаючи різні механіки руху, стрибків, ривків та взаємодії зі стінами. Основні аспекти роботи класу включають ініціалізацію та налаштування компонентів, керування рухом, обробку колізій та взаємодій, анімацію та візуальні ефекти, а також звукові ефекти.

Завдяки добре структурованому коду, клас "PlayerMovement" забезпечує гравцю чітке та плавне керування персонажем. Клас також легко розширюється та модифікується, що дозволяє додавати нові механіки або змінювати існуючі без значних зусиль.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		40

В проєкті окрім руху гравця є три класи які відповідають за збереження, та відродження гравця всі вони пов'язані з класом Player prefs (Рис. 2.20).

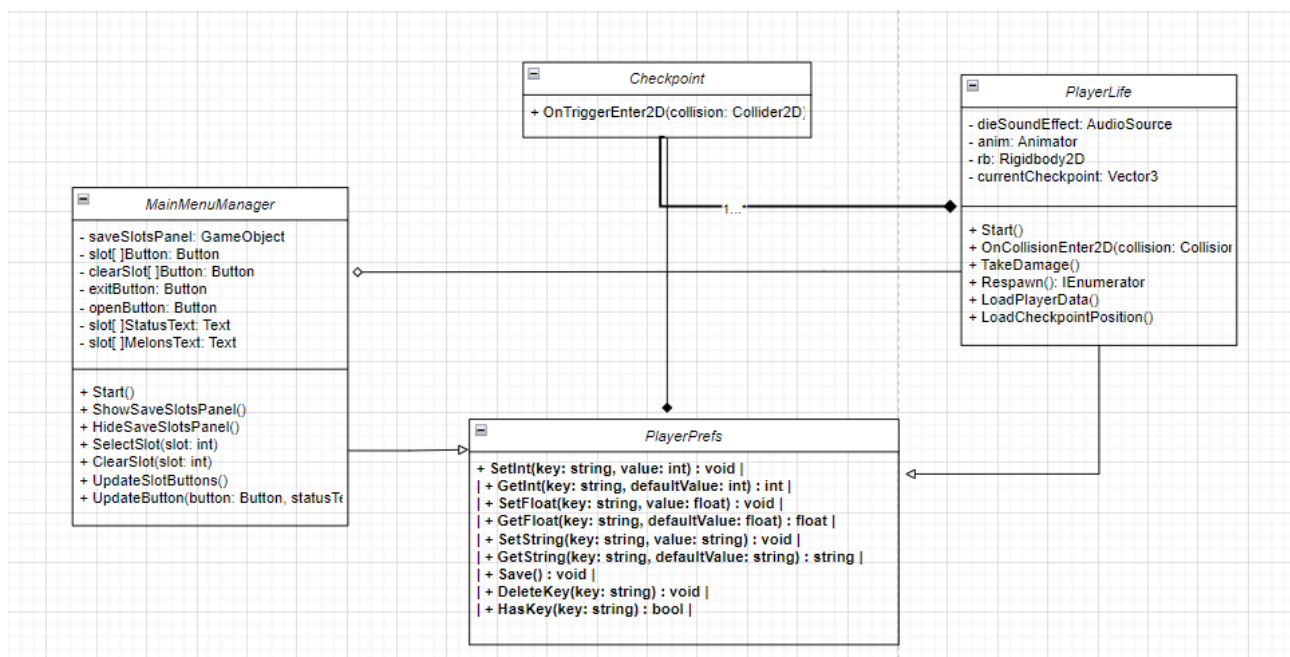


Рисунок. 2.20 – Діаграма класів пов'язаних з "PlayerPrefs"

Таблиця 2.1

Опис методів класу MainMenuManage

Назва	Опис	Дії
Start()	Ініціалізує компонент, прив'язуючи кнопки до відповідних методів та оновлюючи стан кнопок збереження.	Перевіряє, чи призначено панель слотів збереження. Якщо ні, виводить помилку. Прив'язує функції до кнопок та викликає UpdateSlotButtons().
HideSaveSlotsPanel()	Приховує панель слотів збереження.	Встановлює властивість active для панелі слотів збереження у false.

ShowSaveSlotsPanel()	Відображає панель слотів збереження.	Встановлює властивість active для панелі слотів збереження у true.
ClearSlot(int slot)	Очищає дані для вибраного слота.	Видаляє всі ключі збережених даних для вибраного слота у PlayerPrefs та викликає UpdateSlotButtons().
UpdateSlotButtons()	Оновлює стан кнопок слотів збереження.	Викликає UpdateButton() для кожного слота, передаючи відповідні кнопки та тексти стану.
UpdateButton(Button button, Text statusText, Text melonsText, int slot)	Оновлює текст кнопки відповідного слота збереження.	Якщо для слота є збережені дані, оновлює тексти стану та кількості кавунів, інакше встановлює тексти як "Melons: 0".
SelectSlot(int slot)	Вибирає слот збереження і завантажує відповідну сцену.	Встановлює вибраний слот у PlayerPrefs. Якщо для вибраного слота є збережені дані, завантажує відповідну сцену, інакше завантажує "Level 1".

Клас “MainMenuManager” відповідає за керування слотами збереження в грі та забезпечує зручний інтерфейс для гравців, дозволяючи їм вибирати, очищати та переглядати стан своїх збережень. Цей клас відображає панель слотів збереження, де гравці можуть вибрати один із трьох слотів, кожен з яких може зберігати поточний прогрес гравця, включаючи позицію гравця, кількість зібраних предметів та назву сцени. Коли гравець обирає слот, клас завантажує відповідні дані з ‘PlayerPrefs’ і переміщує гравця до відповідної сцени або початкового рівня, якщо збережень немає. Також ‘MainMenuManager’ надає можливість очищення слотів, видаляючи всі пов'язані дані з “PlayerPrefs”, і відповідно оновлює відображення кнопок інтерфейсу для показу поточного стану кожного слота (збережено чи порожній). Крім того, цей клас відповідає за показ та приховування панелі слотів збереження, що дозволяє гравцям легко взаємодіяти з системою збережень.

Таблиця 2.2

Опис методів класу MainMenuManager

Назва	Опис	Дії
Start()	Ініціалізує компонент, завантажуючи необхідні компоненти та дані гравця.	Прив'язує Rigidbody2D та Animator до об'єкта гравця та викликає LoadPlayerData() для завантаження позиції гравця.
OnCollisionEnter2D(Collision2D collision)	Обробляє зіткнення гравця з іншими об'єктами.	Перевіряє, чи об'єкт зіткнення має тег "Trap". Якщо так, викликає метод TakeDamage() для обробки пошкоджень гравця.

TakeDamage()	Виконує дії, пов'язані з пошкодженням гравця.	Відтворює звуковий ефект смерті, запускає анімацію смерті та викликає корутину Respawn() для респауну гравця.
Respawn()	Корутіна, яка обробляє респаун гравця.	Зупиняє рух гравця, чекає одну секунду, завантажує позицію останньої контрольної точки, переміщує гравця до цієї позиції, знову дозволяє рух гравця.
LoadPlayerData()	Завантажує дані про позицію гравця.	Викликає LoadCheckpointPosition(), щоб завантажити позицію контрольної точки та встановити позицію гравця.
LoadCheckpointPosition()	Завантажує координати контрольної точки для обраного слота збереження.	Перевіряє наявність даних контрольної точки у PlayerPrefs. Якщо дані є, завантажує координати; якщо ні, використовує поточну позицію.

Клас “PlayerLife” відповідає за управління життям гравця в грі. Він має методи для обробки пошкоджень “TakeDamage()”, які відтворюють звук смерті гравця та запускають анімацію смерті через аніматор. Після смерті гравця

викликається метод “Respawn()”, який забезпечує плавний відновлення гравця на останньому чекпоінті після короткої затримки. Крім того, клас зберігає позицію гравця на чекпоінті в “PlayerPrefs” та завантажує ці дані під час старту сцени, щоб гравець міг почати гру з останнього збереженого місця.

В класі є спеціальна функція в Unity — Coroutine, яка дозволяє виконувати код у кілька етапів або в певних проміжках часу. Це дуже корисно для реалізації асинхронних операцій, таких як затримки, поступове оновлення або інші дії, які потрібно виконувати протягом кількох кадрів або секунд.

Coroutine запускається методом StartCoroutine() і працює у фоновому режимі паралельно з основним потоком програми. Вона може бути призупинена в будь-який момент за допомогою ключового слова yield, а потім відновлена з того ж місця в наступний кадр або через певний проміжок часу.

У класі PlayerLife є метод Respawn(), який є Coroutine. Його ключові моменти:

```
StartCoroutine Respawn();
```

Coroutine Respawn() запускається у методі TakeDamage() після того, як гравець отримує пошкодження.

```
private IEnumerator Respawn()
{
    rb.bodyType = RigidbodyType2D.Static;
    yield return new WaitForSeconds(1);
    rb.bodyType = RigidbodyType2D.Dynamic;
    LoadCheckpointPosition();
    transform.position = currentCheckpoint;
    rb.bodyType = RigidbodyType2D.Static;
    yield return new WaitForSeconds(1);
    rb.bodyType = RigidbodyType2D.Dynamic;
```

Використання yield return new WaitForSeconds(1); створює затримку на одну секунду, після чого Coroutine відновлює свою роботу. Це дозволяє зробити респаун гравця плавним і наглядним для користувача.

Клас Checkpoint відповідає за реалізацію логіки перевірки чекпоінтів у грі. Основний метод OnTriggerEnter2D(Collider2D collision) активується, коли гравець потрапляє на чекпоінт. Він перевіряє, чи це дійсно гравець, і якщо так, то зберігає його поточну позицію (transform.position) та назву активної сцени у PlayerPrefs. Це дозволяє гравцеві розпочати гру з цього чекпоінту після смерті або перезавантаження сцени.

В другому рівні був реалізований клас Teleporter відповідає за реалізацію телепортів у грі. Він має поле destination, яке визначає точку призначення для телепортації. Основний метод GetDestination() повертає цю точку, щоб інші об'єкти могли отримати доступ до неї і використовувати для переміщення гравця або іншого об'єкта у відповідному напрямку.

```
public class Teleporter : MonoBehaviour
{
    [SerializeField] private Transform destination;
    public Transform GetDestination()
    {
        return destination;
    }
}
```

Клас “Coli” відповідає за обробку подій зіткнень у грі. У нього є три серіалізовані поля: “colliderScript”, “colisionEnter” та “colisionExit”. Поле `colliderScript` визначає назву скрипту, який повинен бути прикріплений до об'єкту, що може стикатися з іншими об'єктами.

Метод “OnCollisionEnter2D(Collision2D col)” викликається, коли об'єкт зіткнується з іншим об'єктом. Він перевіряє наявність компонента, вказаного у “colliderScript”, і якщо такий присутній, активує подію “colisionEnter”.

Метод “OnCollisionExit2D(Collision2D col)” викликається, коли об'єкт припиняє стикатися з іншим об'єктом. Він також перевіряє наявність відповідного компонента і, якщо умова виконується, ініціює подію “colisionExit”.

Цей клас спеціально створений для роботи з ключами та дверима, де взаємодія зіткнень визначає відповідні дії у грі:

					ІПЗ.КР.Б – 121 – 24 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		46

```

public class Coli : MonoBehaviour
{
    [SerializeField] private string colliderScript;
    [SerializeField] private UnityEvent colisionEnter;
    [SerializeField] private UnityEvent colisionExit;
    private void OnCollisionEnter2D(Collision2D col)
    {
        if (col.gameObject.GetComponent(colliderScript))
        {
            colisionEnter?.Invoke();
        }
    }
    private void OnCollisionExit2D(Collision2D col)
    {
        if (col.gameObject.GetComponent(colliderScript))
        {
            colisionExit?.Invoke();
        }
    }
}

```

Висновки до другого розділу

В результаті роботи над другим розділом було визначено та створено основну архітектуру проекту. Зокрема, була розроблена структура гри, включаючи дизайн рівнів з декораціями та пастками. Кожен рівень було детально пророблено з урахуванням геймплейних особливостей та візуальної привабливості.

Для керування персонажем були розроблені відповідні класи, які включали у себе логіку переміщення, обробку взаємодії з об'єктами на рівні, а також взаємодію з пастками та іншими елементами середовища.

Окрім цього, були створені класи, відповідальні за збереження прогресу гри. Це включало в себе реалізацію системи чекпоінтів та збереження поточного стану гравця, що дозволяло йому відновлювати гру після смерті або виходу з неї. Анімаційні ефекти були створені та інтегровані з відповідними класами.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ

3.1 Опис роботи з програмним додатком

При запуску гри гравця зустрічає стартове меню (рис. 3.1) з назвою гри та кнопкою "Старт". Єдина дія користувача, доступна йому, - це натиснути кнопку "Старт", після чого з'являється панель з трьома слотами для збережень (рис. 3.2).



Рисунок. 3.1 – Стартове меню гри



Рисунок. 3.2 – Панель збережень в стартовому меню гри

					ІПЗ.КР.Б – 121 – 24 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		48

У панелі збережень відображається інформація про кожен з трьох слотів збережень. Кожен слот може бути пустим або містити інформацію про збережений рівень гри. Для кожного слоту відображається наступна інформація:

1. Статус слота: показується чи слот порожній ("Порожній") або містить збережений рівень ("Збережено").
2. Кількість кавунів: якщо в слоті є збережений рівень, то також відображається кількість зібраних кавунів у цьому рівні.
3. Дії зі слотом: для кожного слоту є можливість видалити збережену інформацію. Це забезпечується кнопками або іншими відповідними елементами інтерфейсу, які дозволяють користувачеві очистити слот і знову використовувати його для нового збереження.

Ця інформація дозволяє гравцеві керувати своїми збереженнями, вибирати потрібний слот для завантаження гри або очистки слоту для нового збереження.

Після запуску пустого слота збереження запускається перший рівень (рис. 3.3)



Рисунок. 3.3 – Початок першого рівня

Сцена першого рівня включає наступні елементи:

1. Рухаюча платформа(рис. 3.4): на початку рівня гравець зустрічає рухаючу платформу. Це може бути частина гри, де гравець повинен взаємодіяти з платформою, щоб пересуватися або долати перешкоди. Платформа може рухатися горизонтально або вертикально, створюючи виклики для гравця.

2. Підбираємий предмет – кавун (рис. 3.5): у процесі проходження рівня гравець зустрічає підбираємий предмет, яким є кавун. Підбирання кавунів може бути однією з головних мет рівня, що мотивує гравця до дослідження та збору.



Рисунок. 3.4 – Рухаючися платформа з ямою під нею

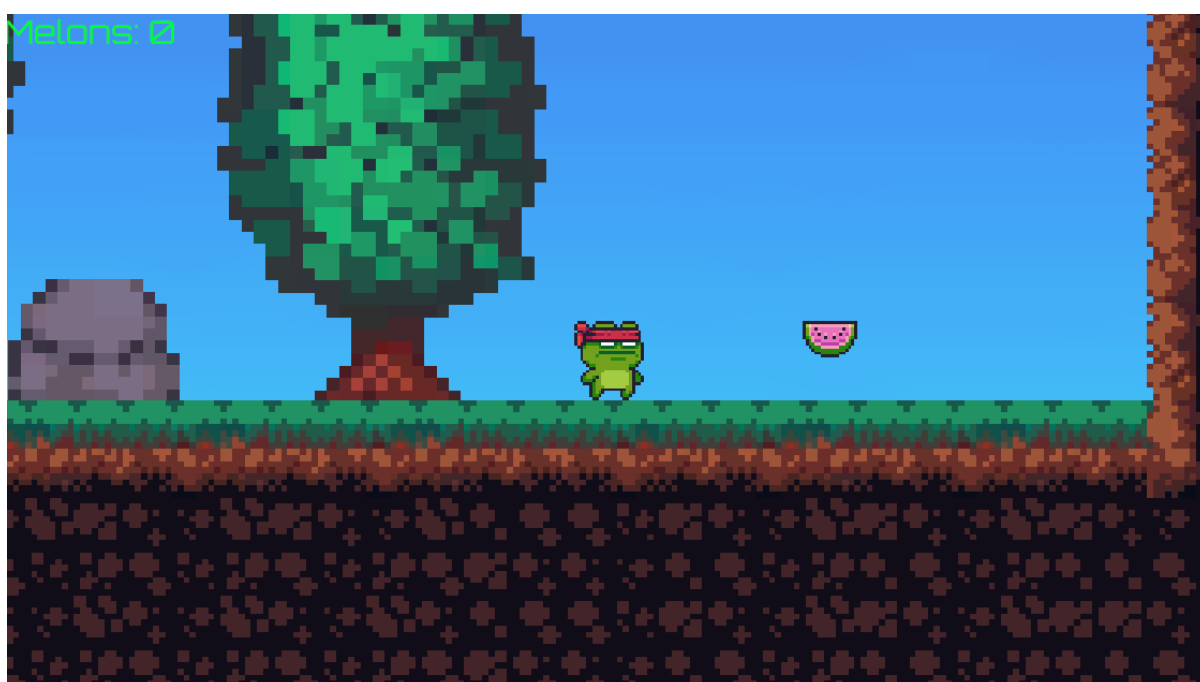


Рисунок. 3.5 – Підбираємий предмет

Після проходження підземного сегменту рівня гравець повинен дійти до входу у замок (рис. 3.6) і завершити рівень. В Рисунку 3.7 зображено зовнішній вигляд всього рівня в редакторі.

					ІПЗ.КР.Б – 121 – 24 - ІЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

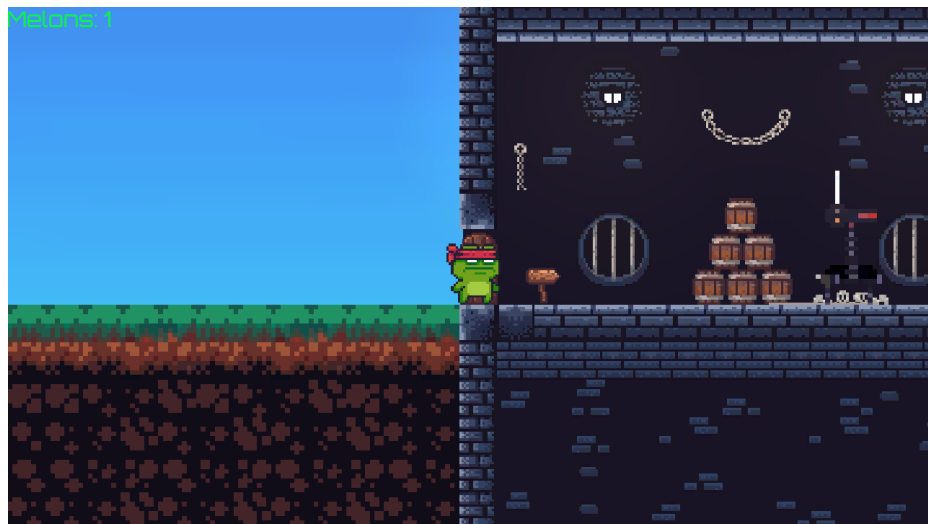


Рисунок. 3.6 – Кінець рівня



Рисунок. 3.7 – Загальний вигляд першого рівня

Після переходу на наступний рівень ми бачимо початок першого (рис. 3.8), де вже змінилось освітлення. Цей рівень ділиться на дві частини, розділені закритими дверима, які відчиняються при підбиранні ключів (рис. 3.9). Гравець має знайти та зібрати ключі, що відкривають доступ до нових ділянок рівня, забезпечуючи таким чином прогрес у грі.



Рисунок. 3.8 – Початок другого рівня

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		51

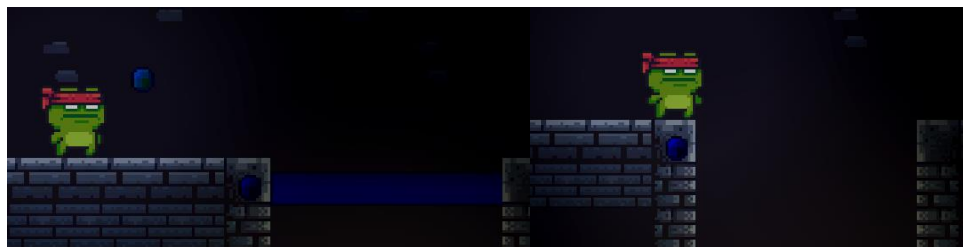


Рисунок. 3.9 – Відкривання дверей при підбиранні ключа

Після відкриття першої дверей настає перший сегмент рівня (рис. 3.10), по середині якого знаходиться захований кавун (рис. 3.11)., який гравець може підібрати, але це не обов'язково для проходження рівня.



Рисунок. 3.10 – Загальна вигляд першого сегменту

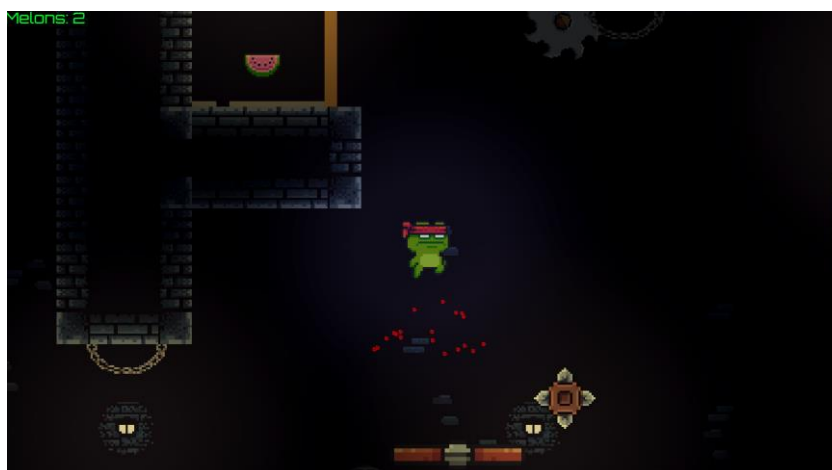


Рисунок. 3.11 – Прихований кавун

В кінці сегмента знаходиться кімната, яка повністю обставлена шипами і має рухомі платформи, що обертаються навколо неї. Якщо гравець стане на таку платформу, він може обійти кімнату навколо і побачити вхід. Пройшовши сегмент з пастками, гравець дістається до телепорта та наступного ключа (рис. 3.12). Після підбору ключа гравець чує відповідний звуковий ефект, що сигналізує про відкриття дверей. Натиснувши кнопку "Е", гравець переноситься на початок рівня, де він бачить відкриті двері (рис. 3.13) відповідного кольору ключа .

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		52



Рисунок. 3.12 – Телепорт та зелений ключ

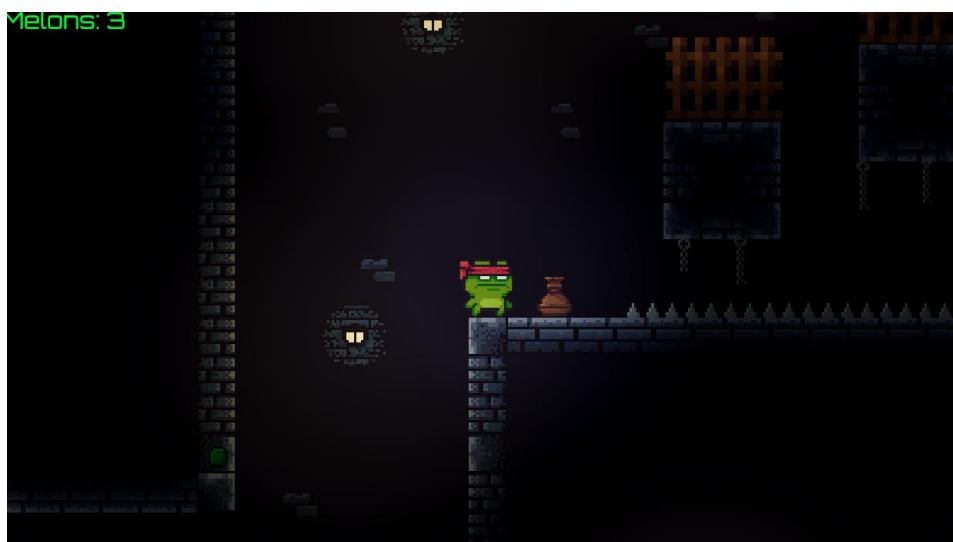


Рисунок. 3.13 – Сегмент рівня за відкритими дверями.

На другому сегменті рівня, зображеному на рисунку 3.14, гравця очікують нові випробування. Тут присутні нові пастки в вигляді падаючих блоків(рис. 3.15). Гравцеві потрібно пробігти через цей сегмент до того, як блоки зникнуть. Але через певний час блоки відновлюються, тому у гравця є шанс знайти вільні і безпечні проміжки для проходження.



Рисунок. 3.14 – Загальний вигляд другої частини рівня.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		53



Рисунок. 3.15 – Падаючі блоки

У кінці другого сегменту рівня знаходиться багато телепортів (рис. 3.16), які гравець повинен пройти. Серед них є один, який веде до ключа і телепорта на початок рівня. Інший телепорт веде до кавуна, що є необов'язковим предметом для збору. Усі інші телепорти призводять до закритої ями з шипами.



Рисунок. 3.16 – Кінцевий сегмент з телепортами

Після підбору останнього ключа і повернення на початок рівня, гравець вирушає до третьої відкритої кімнати. У цій кімнаті знаходиться телепорт (рис. 3.17), який завершує гру і викликає кінцеве меню (рис. 3.18).

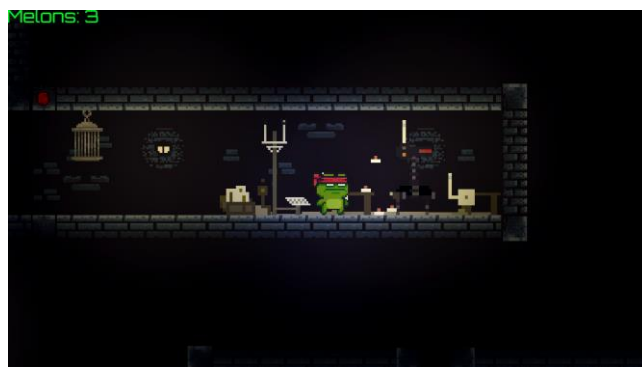


Рисунок. 3.17 – Кінець рівня в вигляді телепорту

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		



Рисунок. 3.18 – Кінцеве меню

3.2 Тестування роботи програмного забезпечення

Було проведено тестування гри для виявлення та усунення багів (дефектів).

Таблиця 3.1

Результати тестування

Сцена	Вхідні дані	Очікувані дані	Спостережені дані
Start Sreen	Натискання кнопки “Start”	Відкривається панель слотів збереження	Відкривається панель слотів збереження
	Натискання пустої кнопки видалення збереження	Збереження видаляється	Збереження видаляється
	Натискання пустої кнопки завантаження збереження	Запускається сцена першого рівню	Запускається сцена першого рівню

Меню паузи	Натиск кнопки продовження гри	Гра продовжується а меню паузи закривається	Гра продовжується а меню паузи закривається
	Натиск кнопки повернення до головного меню	Відбувається перехід на сцену з меню	Відбувається перехід на сцену з меню
Сцена першого рівня	Натискання на кнопки AD	Персонаж рухається в ліво та право	Персонаж рухається в ліво та право
	Натискання на кнопку Spase	Персонаж пригає	Персонаж пригає
	Натискання на кнопку LShift	Нічого не відбувається	Нічого не відбувається
	Натискання на кнопку Lshift рухаючись	Персонаж швидко переміщується в сторону в яку рухався до натискання клавiшi	Персонаж швидко переміщується в сторону в яку рухався до натискання клавiшi
	Спроба доторкнутися до шипів	Персонаж помирає та відроджується на останньому чекпоінті	Персонаж помирає та відроджується на останньому чекпоінті

	Наступаємо на рухаючуся платформу	Персонаж їде разом з платформою	Персонаж їде разом з платформою
	Зіштовхуємося з рухаючоюся пилою	Персонаж помирає та відроджується на останньому чекпоінті	Персонаж помирає та відроджується на останньому чекпоінті
	Підбираємо кавун	В трекер зліва зверху додається плюс один кавун	В трекер зліва зверху додається плюс один кавун
	Підходимо до стіни і впираємось в неї	Персонаж впирається в неї і не рухається	Персонаж впирається в неї і не рухається
	Пригаємо на стіну та тиснемо кнопку руху в її сторону	Прилипаємо до стіни і починаємо по ній ковзати	Прилипаємо до стіни і починаємо по ній ковзати
	Коли ковзаємо на стіні натискаємо на стрибок	Відпригуємо від стіни	Відпригуємо від стіни
	Після натискання на Spase натискаємо ще раз	Персонаж пригає другий раз в повітрі	Персонаж пригає другий раз в повітрі
	Заходимо в двері	Двері відкриваються	Двері відкриваються

	Заходимо в замок	Переносимось на другий рівень	Переносимось на другий рівень
Сцена другого рівня	Підбираємо ключ	Відкривається дверь відповідного кольору до ключа	Відкривається дверь відповідного кольору до ключа
	Стоячи біля телепорту натискаємо E	Телепорт переносить на початок рівня	Телепорт переносить на початок рівня

При тестуванні було виявлено ряд помилок:

Виявлена помилка: під час тестування було виявлено, що персонаж може проходити крізь текстури при наборі дуже великої швидкості.

Причина виникнення: ця проблема виникає через недостатню частоту перевірки зіткнень фізичним двигуном гри. У більшості ігрових рушіїв, таких як Unity, фізика об'єктів обчислюється з певною частотою (фізичні кроки). Якщо персонаж рухається дуже швидко, він може переміщуватися настільки далеко між фізичними кроками, що фізичний двигун пропускає момент, коли персонаж повинен зіткнутися з іншими об'єктами. Це явище відоме як "тунелювання".

Можливі вирішення:

1. Збільшення частоти фізичних кроків: збільшення частоти фізичних кроків дозволить фізичному двигуну частіше перевіряти зіткнення, що зменшить ймовірність тунелювання. У Unity це можна зробити, змінивши значення Fixed Timestep у налаштуваннях Time (Edit > Project Settings > Time).
2. Використання коллайдерів з безперервною детекцією зіткнень: Unity має режим безперервної детекції зіткнень (Continuous Collision Detection) для Rigidbody2D. Цей режим спеціально розроблений для об'єктів, що рухаються з великою швидкістю. Він дозволяє фізичному двигуну

коректно обробляти зіткнення, навіть якщо об'єкти переміщуються на великі відстані між фізичними кроками. Це можна налаштувати через інспектор, встановивши параметр Collision Detection у значення Continuous для Rigidbody2D.

3. Обмеження максимальної швидкості: ще один простий спосіб запобігти тунелюванню – обмежити максимальну швидкість персонажа. Це можна зробити програмно, перевіряючи і коректуючи швидкість Rigidbody2D у кожному кадрі.
4. Додавання додаткових коллайдерів: інший підхід полягає в додаванні додаткових коллайдерів або хитбоксів навколо об'єктів, що збільшує шанс виявлення зіткнення.

Було обрано Обмеження максимальної швидкості:

```
void FixedUpdate()
{
    float maxSpeed = 10f;
    if (rb.velocity.magnitude > maxSpeed)
    {
        rb.velocity = rb.velocity.normalized * maxSpeed;
    }
}
```

Помилка, при якій гравець при переході на другий рівень спавнився за мапою, була пов'язана з тим, що при завершенні першого рівня не стиралися чекпоінти. Це могло спричинити те, що гравець зберігав позицію з попереднього рівня, яка не відповідала розташуванню на новій карті.

Причини проблеми:

Залишені чекпоінти:

- При переході на новий рівень залишалися збережені координати чекпоінтів з попереднього рівня.
- Ці координати не відповідали новій мапі, тому гравець з'являвся за межами ігрового поля.

Для вирішення було оновлено скрипт фінішу:

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		59

```

public class Finish : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            Invoke("CompleteLevel", 2f);
        }
    }
    private void CompleteLevel()
    {
        int selectedSlot = PlayerPrefs.GetInt("SelectedSlot", 1);
        PlayerPrefs.SetString($"SceneName_{selectedSlot}",
            SceneManager.GetActiveScene().name);
        PlayerPrefs.DeleteKey($"CheckpointX_{selectedSlot}");
        PlayerPrefs.DeleteKey($"CheckpointY_{selectedSlot}");
        PlayerPrefs.DeleteKey($"CheckpointZ_{selectedSlot}");
        PlayerPrefs.Save();
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex+1);
    }
}

```

Висновки до третього розділу

При написанні третього розділу було детально описано роботу гри, включаючи керування персонажем, особливі механіки, а також проведено тестування всієї гри. У цьому розділі розглянуто, як гравець взаємодіє з ігровим середовищем, використовуючи елементи керування, і як реалізовано ключові механіки, такі як збирання предметів, активація пасток, проходження рівнів і використання телепортів. Завдяки проведеному тестуванню вдалося ідентифікувати та виправити декілька помилок.

					ІІЗ.КР.Б – 121 – 24 - ІІЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розроблено та реалізовано ігровий додаток, що відповідає визначеним вимогам і функціональним характеристикам. Проєкт охоплює всі основні етапи розробки програмного продукту, включаючи визначення варіантів використання, розробку архітектури, створення дизайну рівнів, анімацій та реалізацію ігрових механік.

Перший розділ роботи був присвячений аналізу та визначенню вимог до програмного продукту. Було сформульовано функціональні та нефункціональні вимоги. На основі цих вимог було створено діаграми варіантів використання та діаграми класів, що дозволило структурувати проєкт та забезпечити його послідовну реалізацію.

У другому розділі було розроблено дизайн рівнів з використанням різних декорацій та пасток. Крім того, було описано створення анімацій для персонажа та інших об'єктів гри. Були розроблені та впроваджені класи для роботи з анімацією, керування діями головного героя, збереженням прогресу гри та інших важливих механік.

Третій розділ зосереджувався на реалізації функціоналу гри. Було описано та протестовано всі ігрові механіки, включаючи рух, збирання предметів, активацію пасток, проходження рівнів і використання телепортів. Завдяки проведеному тестуванню вдалося виявити та виправити низку помилок. Наприклад, було вирішено проблему, коли гравець при переході на другий рівень спавнився за мапою через те, що при завершенні першого рівня не стиралися чекпоінти.

У підсумку, під час виконання кваліфікаційної роботи було створено повнофункціональний ігровий додаток, що відповідає визначеним вимогам. У майбутньому додаток може змінюватися або доповнюватися новими функціями та елементами.

					ІПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		61

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Game Design Document: A Guide. Gamasutra. 2023. [Електронний ресурс] – Режим доступу до ресурсу:
https://www.gamasutra.com/view/feature/132649/creating_a_great_design_document.php
2. Unity Learn. Your First Game Jam. Unity Technologies. 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.unity.com/course/your-first-game-jam>
3. Best Platformers of All Time. Culture of Gaming. 2023. [Електронний ресурс] – Режим доступу до ресурсу: <https://cultureofgaming.com/25-best-platformers/>
4. Dead Cells – Game. Motion Twin official. 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://dead-cells.com>
5. Hollow Knight official. Team Cherry. 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.hollowknight.com>
6. Celeste - Help Madeline. Official Game Maddy Makes Games. 2024. [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.celestegame.com>
7. Unity Animation and Rigging. Unity Learn. 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.unity.com/course/animation-and-rigging>
8. Designing Levels for Games. Game Developer. 2023. [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.gamedeveloper.com/design/designing-levels-for-games>
9. Unity Scripting API. Unity Technologies. 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/ScriptReference/>
10. Game Development Patterns. Game Programming Patterns. 2023. [Електронний ресурс] – Режим доступу до ресурсу:
<https://gameprogrammingpatterns.com/>

					ІІЗ.КР.Б – 121 – 24 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		62

11. Implementing Save Systems in Unity. Ray Wenderlich. 2023. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.raywenderlich.com/913-save-and-load-data-in-unity>
12. Unity Documentation. Animation Overview. Unity Technologies. 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/Manual/AnimationOverview.html>
13. Unity - Manual: Canvas. Unity User Manual 2022.3 (LTS). [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/ScriptReference/Canvas.html>
14. UNITY БАЗОВИЙ: курс. Андрій Олійник. 2023. [Електронний ресурс] – Режим доступу до ресурсу: <https://itvdn.com/ua/video/unity-essential>
15. Unity - Manual: Using Animation Events. Unity User Manual 2023.2 (LTS). [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/Manual/AnimationOverview.html>
16. Unity - Manual: UsesLuminPlatformLevelAttribute. Unity User Manual 2023.2 (LTS). [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/ScriptReference/UsesLuminPlatformLevelAttribute.html>
17. Unity - Manual: Behaviour. Unity User Manual 2022.3 (LTS). [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/ScriptReference/Behaviour.html>
18. Unity - Manual: BoxCollider2D. Unity User Manual 2022.3 (LTS). [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/ScriptReference/BoxCollider2D.html>
19. Unity - Manual: Collider2D. Unity User Manual 2022.3 (LTS). [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/ScriptReference/Collider2D.html>
20. Unity - Manual: GameObject. Unity User Manual 2022.3 (LTS). [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/ScriptReference/GameObject.html>

ДОДАТКИ

					ПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг файлу “MainMenuManager.cs”

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class MainMenuManager : MonoBehaviour
{
    public GameObject saveSlotsPanel;
    public Button slot1Button;
    public Button slot2Button;
    public Button slot3Button;
    public Button clearSlot1Button;
    public Button clearSlot2Button;
    public Button clearSlot3Button;
    public Button exitButton;
    public Button openButton;
    public Text slot1StatusText;
    public Text slot1MelonsText;
    public Text slot2StatusText;
    public Text slot2MelonsText;
    public Text slot3StatusText;
    public Text slot3MelonsText;

    private void Start()
    {
        if (saveSlotsPanel == null)
        {
            Debug.LogError("SaveSlotsPanel не призначено в інспекторі.");
            return;
        }
    }
}
```

					ІІЗ.КР.Б – 121 – 24 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		65

```
}
```

```
slot1Button.onClick.AddListener(() => SelectSlot(1));  
slot2Button.onClick.AddListener(() => SelectSlot(2));  
slot3Button.onClick.AddListener(() => SelectSlot(3));  
clearSlot1Button.onClick.AddListener(() => ClearSlot(1));  
clearSlot2Button.onClick.AddListener(() => ClearSlot(2));  
clearSlot3Button.onClick.AddListener(() => ClearSlot(3));  
exitButton.onClick.AddListener(HideSaveSlotsPanel);  
openButton.onClick.AddListener>ShowSaveSlotsPanel);
```

```
UpdateSlotButtons();
```

```
}
```

```
public void ShowSaveSlotsPanel()
```

```
{
```

```
    saveSlotsPanel.SetActive(true);
```

```
}
```

```
public void HideSaveSlotsPanel()
```

```
{
```

```
    saveSlotsPanel.SetActive(false);
```

```
}
```

```
private void SelectSlot(int slot)
```

```
{
```

```
    PlayerPrefs.SetInt("SelectedSlot", slot);
```

```
    if (PlayerPrefs.HasKey($"CheckpointX_{slot}"))
```

```
{
```

					ІІЗ.КР.Б – 121 – 24 - ІІЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        string sceneName = PlayerPrefs.GetString($"SceneName_{slot}", "Level 1");
        SceneManager.LoadScene(sceneName);
    }
    else
    {
        SceneManager.LoadScene("Level 1");
    }
}

private void ClearSlot(int slot)
{
    PlayerPrefs.DeleteKey($"CheckpointX_{slot}");
    PlayerPrefs.DeleteKey($"CheckpointY_{slot}");
    PlayerPrefs.DeleteKey($"CheckpointZ_{slot}");
    PlayerPrefs.DeleteKey($"SceneName_{slot}");
    PlayerPrefs.DeleteKey($"Melons_{slot}");
    PlayerPrefs.Save();
    UpdateSlotButtons();
}

private void UpdateSlotButtons()
{
    UpdateButton(slot1Button, slot1StatusText, slot1MelonsText, 1);
    UpdateButton(slot2Button, slot2StatusText, slot2MelonsText, 2);
    UpdateButton(slot3Button, slot3StatusText, slot3MelonsText, 3);
}

private void UpdateButton(Button button, Text statusText, Text melonsText, int slot)
{
    if (PlayerPrefs.HasKey($"CheckpointX_{slot}"))

```

```

{
    int melons = PlayerPrefs.GetInt($"Melons_{slot}", 0);
    statusText.text = $"Слот {slot}: Збережено";
    melonsText.text = $"Melons: {melons}";
}
else
{
    statusText.text = $"Слот {slot}: Порожній";
    melonsText.text = "Melons: 0";
}
}
}

```

Лістинг файлу “PlayerMovement.cs”

```

public class PlayerMovement : MonoBehaviour
{
    private Rigidbody2D rb;
    private Animator anim;
    private BoxCollider2D coll;

    [SerializeField] public Transform wallCheck;
    private bool isWallSliding;
    [SerializeField] private float wallSlidingSpeed = 2f;

    private bool doubleJump;

    private float dirxRaw;
    private float diryRaw;

    private bool isFacingRight = true;

```

```

private bool isWallJumping;
private float wallJumpingDirection;
[SerializeField] private float wallJumpingTime = 0.1f;
private float wallJumpingCounter;
[SerializeField] private float wallJumpingDuration = 0.2f;
private Vector2 wallJumpingPower = new Vector2(8f, 16f);

```

```

[SerializeField] private LayerMask groundlayer;
[SerializeField] private float movespeed = 10f;
[SerializeField] private float jumpingPower = 14f;
private float dashcd = 0.8f;
[SerializeField] private float dashSpeed = 20;
private bool hasDashed;
private bool isDashing = false;

```

```

private bool wallslideSoundPlay = false;
private bool runingSoundPlay = false;

```

```

public ParticleSystem JumpParticle;
public ParticleSystem MoveTrail;
public ParticleSystem WallSlidingTrail;
public ParticleSystem DashGhost;
public ParticleSystem DashTraill;

```

```

[SerializeField] private AudioSource JumpSoundEffect;
[SerializeField] private AudioSource WallSlidingSoundEffect;
[SerializeField] private AudioSource DashSoundEffect;
[SerializeField] private AudioSource RuningSoundEffect;

```

```
[SerializeField] private float dustfperiod;
```

```
private float counter;
```

```
private enum MovmentState { idle, running, jumping, falling, on_the_wall }
```

```
private void Start()
```

```
{
```

```
    rb = GetComponent<Rigidbody2D>();
```

```
    coll = GetComponent<BoxCollider2D>();
```

```
    anim = GetComponent<Animator>();
```

```
}
```

```
private void Update()
```

```
{
```

```
    float x = Input.GetAxis("Horizontal");
```

```
    float y = Input.GetAxis("Vertical");
```

```
    dirxRaw = Input.GetAxisRaw("Horizontal");
```

```
    diryRaw = Input.GetAxisRaw("Vertical");
```

```
    Jumps();
```

```
    RuningSound();
```

```
    WallSlide();
```

```
    WallJump();
```

					ІІЗ.КР.Б – 121 – 24 - ІІЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

UpdateAnimationState();

if (!isWallJumping && !isWalltouch())
{
    Flip();
}
if (Input.GetButtonDown("Fire3") && !hasDashed)
{
    if (dirxRaw != 0 || diryRaw != 0)

        Dash();

}
if (Isgrounded() && !Input.GetButton("Fire3"))
{
    hasDashed = false;
}
}
private void FixedUpdate()
{
    if (!isWallJumping && !isDashing )
    {
        rb.velocity = new Vector2(dirxRaw * movespeed, rb.velocity.y);
    }

    CreateMoveParticle();
    CreateWallSlideParticle();
    CreateDashParticle();
}

```

```

private bool isWalltouch()
{
    return Physics2D.OverlapBox(wallCheck.position, new Vector2(0.15f, 0.5f), 0f,
groundlayer);
}
private void WallSlide()
{
    if (isWalltouch() && !Isgrounded() && dirxRaw != 0f)
    {
        isWallSliding = true;
        rb.velocity = new Vector2(rb.velocity.x, Mathf.Clamp(rb.velocity.y, -
wallSlidingSpeed, float.MaxValue));
    }
    else
    {
        isWallSliding = false;
    }

    if (isWallSliding && !wallslideSoundPlay)
    {
        WallSlidingSoundEffect.Play();
        wallslideSoundPlay = true;
    }
    else if (!isWallSliding)
    {
        WallSlidingSoundEffect.Stop();
        wallslideSoundPlay = false;
    }
}

```

```

}
private void Jumps()
{
    if (Isgrounded() && !Input.GetButton("Jump"))
    {
        doubleJump = false;
    }

    if (Input.GetButtonDown("Jump"))
    {
        if (Isgrounded() || doubleJump || isWallSliding)
        {
            CreateJumpParticle();
            rb.velocity = new Vector2(rb.velocity.x, jumpingPower);
            JumpSoundEffect.Play();

            if (!Isgrounded() && !doubleJump && isWallSliding)
            {
                doubleJump = true;
            }
            else
            {
                doubleJump = !doubleJump;
            }
        }
    }

    if (Input.GetButtonUp("Jump") && rb.velocity.y > 0f)
    {

```

					ІІЗ.КР.Б – 121 – 24 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		73

```

        rb.velocity = new Vector2(rb.velocity.x, rb.velocity.y * 0.5f);
    }
}
private void UpdateAnimationState()
{
    MovmentState state;

    if (dirxRaw > 0f || dirxRaw < 0f || rb.velocity.x != 0)
    {
        state = MovmentState.running;
    }
    else
    {
        state = MovmentState.idle;
    }

    if (rb.velocity.y > .1f)
    {
        state = MovmentState.jumping;
    }
    else if (rb.velocity.y < -.1f)
    {
        state = MovmentState.falling;
    }

    if (isWalltouch() && isWallSliding)
    {
        state = MovmentState.on_the_wall;
    }
}

```

```

    }
    if (isWallJumping)
    {
        state = MovmentState.jumping;
    }
    anim.SetInteger("state", (int)state);
}
private bool Isgrounded()
{
    return Physics2D.BoxCast(coll.bounds.center, coll.bounds.size, 0f, Vector2.down,
.1f, groundlayer);
}
private void WallJump()
{
    if (isWallSliding)
    {
        isWallJumping = false;
        wallJumpingDirection = -transform.localScale.x;
        wallJumpingCounter = wallJumpingTime;

        CancelInvoke(nameof(StopWallJumping));
    }
    else
    {
        wallJumpingCounter -= Time.deltaTime;
    }

    if (Input.GetButtonDown("Jump") && wallJumpingCounter > 0f)
    {
        isWallJumping = true;
    }

```

```

        rb.velocity = new Vector2(wallJumpingDirection * wallJumpingPower.x,
wallJumpingPower.y);
        wallJumpingCounter = 0f;

        if (transform.localScale.x != wallJumpingDirection)
        {
            isFacingRight = !isFacingRight;
            Vector3 localScale = transform.localScale;
            localScale.x *= -1f;
            transform.localScale = localScale;
        }

        Invoke(nameof(StopWallJumping), wallJumpingDuration);
    }

}

private void StopWallJumping()
{
    isWallJumping = false;
}

private void Flip()
{
    if (isFacingRight && dirxRaw < 0f || !isFacingRight && dirxRaw > 0f)
    {
        isFacingRight = !isFacingRight;
        Vector3 localScale = transform.localScale;
        localScale.x *= -1f;
        transform.localScale = localScale;
    }
}

```

```

}
private void Dash()
{
    DashSoundEffect.Play();
    hasDashed = true;
    CreateDashGhostParticle();
    Vector2 dashDirection = new Vector2(dirxRaw, diryRaw).normalized;

    if (dashDirection != Vector2.zero)
    {
        StartCoroutine(DashCoroutine(dashDirection));
    }
    else
    {
        ResetDash();
    }
}
private IEnumerator DashCoroutine(Vector2 dashDirection)
{
    isDashing = true;
    rb.velocity = dashDirection * dashSpeed;

    yield return new WaitForSeconds(dashcd);

    ResetDash();
}
private void ResetDash()
{
    isDashing = false;

```

```

    if (Isgrounded())
    {
        hasDashed = false;
    }

}

private void CreateJumpParticle()
{
    if (!Isgrounded()) {
        JumpParticle.Play();
    }
}

private void CreateMoveParticle()
{
    counter += Time.deltaTime;
    if (Isgrounded() && dirxRaw < 0f || Isgrounded() && dirxRaw > 0f)
    {
        if (counter > 0.16)
        {
            MoveTrail.Play();
            counter = 0;
        }
    }
}

private void CreateWallSlideParticle()
{
    counter += Time.deltaTime;

```

```

if (isWalltouch() )
{
    if (counter > 0.3f)
    {
        WallSlidingTrail.Play();
        counter = 0;
    }
}
}

```

```

private void CreateDashParticle()
{
    counter += Time.deltaTime;
    if (isDashing)
    {
        if (counter > 0.1f)
        {
            DashTraill.Play();
            counter = 0;
        }
    }
}

```

```

private void CreateDashGhostParticle()
{
    DashGhost.Play();
}

```

```

private void RuningSound()
{

```

					ПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						79
Змн.	Арк.	№ докум.	Підпис	Дата		

```

if (dirxRaw != 0 && !runingSoundPlay && !isWalltouch() && Isgrounded())
{
    RuningSoundEffect.Play();
    runingSoundPlay = true;
}
else if (dirxRaw == 0 || !Isgrounded())
{
    RuningSoundEffect.Stop();
    runingSoundPlay = false;
}
}
}

```

Лістинг файлу “PlayerLife.cs”

```

public class PlayerLife : MonoBehaviour
{
    [SerializeField] private AudioSource dieSoundEffect;
    [SerializeField] private Animator anim;
    [SerializeField] private Rigidbody2D rb;

    private Vector3 currentCheckpoint;

    private void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        LoadPlayerData();
    }
}

```

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Trap"))
    {
        TakeDamage();
    }
}
```

```
private void TakeDamage()
{
    dieSoundEffect.Play();
    anim.SetTrigger("death");
    StartCoroutine(Respawn());
}
```

```
private IEnumerator Respawn()
{
    rb.bodyType = RigidbodyType2D.Static;
    yield return new WaitForSeconds(1);
    rb.bodyType = RigidbodyType2D.Dynamic;

    LoadCheckpointPosition();
    transform.position = currentCheckpoint;
    rb.bodyType = RigidbodyType2D.Static;

    yield return new WaitForSeconds(1);
    rb.bodyType = RigidbodyType2D.Dynamic;
}
```

```
private void LoadPlayerData()
```

					ІІЗ.КР.Б – 121 – 24 - ІІЗ	Арк.
						81
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{

    LoadCheckpointPosition();
    transform.position = currentCheckpoint;
    Debug.Log("Player spawned at: " + currentCheckpoint);
}

private void LoadCheckpointPosition()
{
    int selectedSlot = PlayerPrefs.GetInt("SelectedSlot", 1);
    if (PlayerPrefs.HasKey($"CheckpointX_{selectedSlot}"))
    {
        currentCheckpoint = new Vector3(
            PlayerPrefs.GetFloat($"CheckpointX_{selectedSlot}"),
            PlayerPrefs.GetFloat($"CheckpointY_{selectedSlot}"),
            PlayerPrefs.GetFloat($"CheckpointZ_{selectedSlot}"));
    }
    else
    {
        currentCheckpoint = transform.position;
    }
}
}

```

Лістинг файлу “Finish.cs”

```

public class Finish : MonoBehaviour
{
    void Start()
    {

```

```

    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            Invoke("CompleteLevel", 2f);
        }
    }

    private void CompleteLevel()
    {
        int selectedSlot = PlayerPrefs.GetInt("SelectedSlot", 1);
        PlayerPrefs.SetString($"SceneName_{selectedSlot}",
SceneManager.GetActiveScene().name);

        PlayerPrefs.DeleteKey($"CheckpointX_{selectedSlot}");
        PlayerPrefs.DeleteKey($"CheckpointY_{selectedSlot}");
        PlayerPrefs.DeleteKey($"CheckpointZ_{selectedSlot}");
        PlayerPrefs.Save();

        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}

Лістинг файлу “Teleporter.cs”
public class Teleporter : MonoBehaviour
{
    [SerializeField] private Transform destination;

    public Transform GetDestination()

```

```

{
    return destination;
}
}

```

Лістинг файлу “BrickDestroyer.cs”

```

public class BrickDestroyer : MonoBehaviour
{
    private Animator anim;
    [SerializeField] private AudioSource breake;
    [SerializeField] private float respawnTime = 5f;

    private void Awake()
    {
        anim = GetComponent<Animator>();
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            Invoke("PlayAnimationAndSound", 1f);
            Invoke("DeactivateBlock", 1.2f);
        }
    }

    private void PlayAnimationAndSound()
    {
        anim.Play("Brick");
        breake.Play();
    }
}

```

```
}
```

```
private void DeactivateBlock()
{
    gameObject.SetActive(false);
    Invoke("ReactivateBlock", respawnTime);
}
```

```
private void ReactivateBlock()
{
    gameObject.SetActive(true);
}
}
```

Лістинг файлу “WallDoor.cs”

```
public class WallDoor : MonoBehaviour
{
    private Animator anim;
    [SerializeField] private AudioSource OpenSound;
    [SerializeField] private AudioSource TakeSound;
    private void Awake()
    {
        anim = GetComponent<Animator>();
    }

    [ContextMenu("Open")]
    public void Open()
    {
        anim.SetTrigger("Open");
        OpenSound.Play();
        TakeSound.Play();
    }
}
```

```
}
```

```
}
```

Лістинг файлу “itemCollector.cs”

```
public class itemCollector : MonoBehaviour
```

```
{
```

```
    private int melons = 0;
```

```
    [SerializeField] private Text melonsText;
```

```
    [SerializeField] private AudioSource pickUpCollectablesSoundEffect;
```

```
    private int selectedSlot;
```

```
    private void Start()
```

```
{
```

```
        selectedSlot = PlayerPrefs.GetInt("SelectedSlot", 1);
```

```
        melons = PlayerPrefs.GetInt($"Melons_{selectedSlot}", 0);
```

```
        UpdateMelonsText();
```

```
}
```

```
    private void OnTriggerEnter2D(Collider2D collision)
```

```
{
```

```
        if (collision.gameObject.CompareTag("Melon"))
```

```
{
```

```
            Destroy(collision.gameObject);
```

```
            melons++;
```

```
            Debug.Log("Melons: " + melons);
```

```
            UpdateMelonsText();
```

```
            pickUpCollectablesSoundEffect.Play();
```

```
            SaveMelons();
```

```
        }
```

```
}
```

					ІІЗ.КР.Б – 121 – 24 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		86

```
private void UpdateMelonsText()
{
    melonsText.text = "Melons: " + melons;
}
```

```
private void SaveMelons()
{
    PlayerPrefs.SetInt($"Melons_{selectedSlot}", melons);
    PlayerPrefs.Save();
}
}
```

Лістинг файлу “Checkpoint.cs”

```
public class Checkpoint : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            int selectedSlot = PlayerPrefs.GetInt("SelectedSlot", 1);
            PlayerPrefs.SetFloat($"CheckpointX_{selectedSlot}", transform.position.x);
            PlayerPrefs.SetFloat($"CheckpointY_{selectedSlot}", transform.position.y);
            PlayerPrefs.SetFloat($"CheckpointZ_{selectedSlot}", transform.position.z);
            PlayerPrefs.SetString($"SceneName_{selectedSlot}",
SceneManager.GetActiveScene().name);
            PlayerPrefs.Save();
            Debug.Log("Checkpoint saved at: " + transform.position);
        }
    }
}
```

Лістинг файлу “FinishTeleport.cs”

					ІІЗ.КР.Б – 121 – 24 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		87

```

public class FinishTeleport : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            Invoke("CompleteLevel", 2f);
        }
    }

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

```

Лістинг файлу “Chain.cs”

```

public class Chain : MonoBehaviour
{
    [SerializeField] private float speed = 2f;
    void Update()
    {
        transform.Rotate(0, 0, 360 * speed * Time.deltaTime);
    }
}

```

Лістинг файлу “PlayerTeleport.cs”

```
public class PlayerTeleport : MonoBehaviour
{
    private GameObject currentTeleporter;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            if (currentTeleporter != null)
            {
                transform.position =
currentTeleporter.GetComponent<Teleporter>().GetDestination().position;
            }
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Teleporter"))
        {
            currentTeleporter = collision.gameObject;
        }
    }

    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.CompareTag("Teleporter"))
        {
            if (collision.gameObject == currentTeleporter)
```

					ІІЗ.КР.Б – 121 – 24 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		89

```

    {
        currentTeleporter = null;
    }
}
}
}

```

Лістинг файлу “PauseMenu.cs”

```

public class PauseMenu : MonoBehaviour

```

```

{
    public static bool PauseGame;
    public GameObject pauseMenu;

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

```

```

public void Resume()

```

```

{
    pauseMenu.SetActive(false);
}

```

					ІІЗ.КР.Б – 121 – 24 - ІІЗ	Арк.
						90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        Time.timeScale = 1f;
        PauseGame = false;
    }

    public void Pause()
    {
        pauseMenu.SetActive(true);
        Time.timeScale = 0f;
        PauseGame = true;
    }

```

```

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

```

Лістинг файлу “CameraControler.cs”

```

public class CameraControler : MonoBehaviour

```

```

{
    [SerializeField] private Transform player;
    private void Update()
    {
        transform.position = new Vector3 (player.position.x, player.position.y,
transform.position.z);
    }
}

```

Лістинг файлу “.cs”

```

public class Coli : MonoBehaviour

```

					ПЗ.КР.Б – 121 – 24 - ПЗ	Арк.
						91
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{
    [SerializeField] private string colliderScript;
    [SerializeField] private UnityEvent colisionEnter;
    [SerializeField] private UnityEvent colisionExit;

    private void OnCollisionEnter2D(Collision2D col)
    {
        if (col.gameObject.GetComponent(colliderScript))
        {
            colisionEnter?.Invoke();
        }
    }

    private void OnCollisionExit2D(Collision2D col)
    {
        if (col.gameObject.GetComponent(colliderScript))
        {
            colisionExit?.Invoke();
        }
    }
}

```