ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка ігрового додатку в середовищі Unity»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Дмитро КОВАЛЕНКО
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти Дмитро КОВАЛЕНКО
група ІСДМ-62

Керівник: Ольга ПОЛОНЕВИЧ
науковий ступінь, вчене звання к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС

_____ Каміла СТОРЧАК
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Коваленко Дмитро Богданович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка ігрового додатку в середовищі Unity

керівник кваліфікаційної роботи Ольга ПОЛОНЕВИЧ к.т.н., доцент,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, документація Unity, вимоги до ігор сучасності .

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження процесу розробки
Аналіз інструментів які надає Unity
Реалізація ігрових механік
5. Перелік графічного матеріалу: *презентація*

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Планування розробки гри	19.10-05.11.23	
2	Пошук та аналіз тематики гри	05.11-12.11.23	
3	Аналіз основного інструментарію	13.11-19.11.23	
4	Збір ресурсів та підготовчі процеси	20.11-25.11.23	
5	Розробка основних механік	27.11-03.12.23	
6	Розробка рівнів та компіляція	04.12-10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Дмитро КОВАЛЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Ольга ПОЛОНЕВИЧ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра (магістра): 65 стор., 84 рис., 0 табл., 6 джерел.

Мета роботи – метою даної дипломної роботи є розробка та реалізація платформера в середовищі Unity з метою дослідження оптимальних методів розв'язання проблем, що виникають під час розробки ігор цього жанру.

Об'єкт дослідження – процес розробки гри.

Предмет дослідження – інструменти Unity.

Короткий зміст роботи: ця робота спрямована на практичне втілення теоретичних концепцій, вивчених у теоретичній частині, та представлення результатів у вигляді функціонального продукту, який має бути прикладом, для подальших досліджень, чи проектів подібного типу.

КЛЮЧОВІ СЛОВА: ЮНІТІ, ГЕМПЛЕЙ, ІГРОВІ МЕХАНІКИ, РОЗРОБКА, ДИЗАЙН РІВНІВ

ABSTRACT

Text part of the master's qualification work: 65 pages, 84 pictures, 0 table, 6 sources.

The purpose of the work - the purpose of this thesis is the development and implementation of a platformer in the Unity environment in order not only to demonstrate technical skills in creating games of this genre, but also to research optimal methods of solving problems that arise during the development of games of this genre.

The object of research is the process of game development on Unity and its ways to solve issues that arise during game development.

The subject of research is the use of Unity tools in game development.

Summary of the work: this work is aimed at the practical implementation of the theoretical concepts studied in the theoretical part and the presentation of the results in the form of a functional product that should be an example for further research or projects of a similar type.

KEYWORDS: UNITY, GAMEPLAY, GAME MECHANICS, DEVELOPMENT, LEVEL DESIGN.

ЗМІСТ

ВСТУП

Актуальність. У сучасному цифровому світі ігрова індустрія визначає нові тренди в розвитку розважальних технологій та стає ключовим фактором формування сучасної культури. В цьому контексті розробка ігор на популярних геймдевельних платформах визначає необхідність пошуку та впровадження новаторських рішень, щоб задовольнити постійно зростаючі вимоги геймерської аудиторії. Однією з найбільш поширених та ефективних платформ для створення відмінних ігрових виробів є Unity.Актуальність даної роботи полягає в необхідності подальшого розширення можливостей геймдевельного інструментарію та створення високоякісних ігор, які відповідають високим стандартам геймерського співтовариства. Зосереджуючись на розробці ігор на Unity, дослідження відділене на вивчення найновіших технологій, підходів та методів, що сприяють високопродуктивній та творчій роботі в цьому середовищі.Результати даного дослідження мають потенціал вплинути на подальший розвиток галузі геймдевелопменту та сприяти створенню ігор, які вразять своєю якістю, оригінальністю та емоційною взаємодією з гравцями.

Мета роботи – метою даної дипломної роботи є розробка та реалізація платформера в середовищі Unity з метою дослідження оптимальних методів розв'язання проблем, що виникають під час розробки ігор цього жанру.

Для виконання поставленої мети, у магістрській роботі розроблено та виконано наступні завдання:

Дослідження процесу розробки
Аналіз інструментів які надає Unity
Реалізація ігрових механік

Об’єкт дослідження – процес розробки гри.

Предмет дослідження – інструменти Unity.

Методи дослідження - Під час виконання завдань магістерської кваліфікаційної роботи були використані методи розробки ігрових додатків, а саме платформерів.

Наукова новизна одержаних результатів. Наукова новизна магістерської кваліфікаційної роботи, полягає у розробці практичних рекомендацій щодо реалізації ігри на основі Unity, додатково сам проект може бути чудовим прикладом для початківців в цій сфері.

Практична значущість одержаних результатів. Практична значимість дослідження полягає у можливості застосування запропонованого рішення на практиці при розробці більш складних проектів в майбутньому.

1 ПЛАНУВАННЯ ТА АНАЛІТИКА

1.1 Планування гри

Створення плану розробки гри - це важливий етап, який допомагає організувати роботу та забезпечити систематичний прогрес у процесі створення гри.

Першим етапом в розробці це аналіз та підготовка. В цьому етапі ми маємо визначити основну основну концепцію гри, та зібрати як омога більше матеріалів. Підготувати інструментарій, зрозуміти що саме ми взагалі будемо проектувати. В цьому етапі, додатково для зручності, при зборі ресурсів потрібно їх інтегрувати в середовище розробки, що на етапі програмування та проектування рівнів одразу використовувати готові рішення.

Наступний етап це збір основних механік які будуть використовуватись в грі, потрібно зрозуміти як теоретичні відомості перенести а практичну. Відносно від того як саме будуть реалізовані технічна частина проекту залежить, як саме ми будемо проектувати наступні рівні, які рішення будуть прийматись в ході розробки

1.2 Пошук та аналіз основної тематики до гри

Тематика гри - це основна ідея або концепція, навколо якої будується весь геймплей та візуальний стиль. Вона визначає обрану тему, атмосферу, сюжет та стиль гри. Тематика гри може бути широкою і різноманітною, включаючи фантастичні світи, історичні події, наукову фантастику, пригодницькі історії, війну, фентезі та багато іншого.

В контексті розробки платформера на базі Unity, тематика гри може впливати на обрані візуальні елементи, анімацію персонажів, архітектуру рівнів та

загальний настрій гри. Важливо обрати тематику, яка буде цікавою для цільової аудиторії та відображатиме ваші творчі задуми та концепцію гри.

За основу було вирішено взяти сьогоднішню тематику кібер панку, яка обрала популярність після виходу в світ одно іменої гри. Обрання такої набагато полегшить збір ресурсів до даного проекту, та спростить ряд рішень які будуть використані під час розробки.

Сама гра, буде описувати подорож персонажу, по постіндустріальному заводу, який буде кидати йому виклик, вигляді складного рівневого дизайну, ворогів, та перепон. Гра буде називатись «Втеча від індустріалізації». В ній головний герой матиме завдання втечі з промислового підприємства, великих масштабів. Головна причина в течі не є важливою, тому що основний акцент буде звернуто на механіки гри та дизайн рівнів. Підчас подорожі на гравця будуть чекати виклики, які потребуватимуть від нього реакції, моторики, адаптації, та кординації.

Тематика промислового підприємства підприємства в стилі кібер панку обрана додатково для, створення пригніченої атмосфери, замету якої взято підштовхування гравця до зміни положення ігрового середовища, та руху в перед до кінця

Дана тематика також надає змогу використовувати більш зрозумілі терміни для сьогоденного гравця, та спростити занурення гравця в ігровий процес. Поєднуючи терміни сьогодення, та фантастичного майбутнього, яке не є веселим, тематика гри надасть змогу відкинути в гравця зайві запитання, та полегшити ігровий процес для гравця.

Додатково, обрана тема промислового підприємства, на дає можливість реалізації платформеру. Гравцю не потрібно буде пояснювати велику кількість рішень, при створенні дизайну рівнів. Гравець буде чітко розуміти чому саме деякі елементи ігрового простору знаходяться в тому чи іншому місці. Додатково візуальна частина заводу, поєднується з сього деням, тобто візуально фантастичні заводи, темного, майбутнього не дуже сильно відрізняються від сьогодення, що значно полегшує збір ресурсів для розробки гри. Плюсом є ще те, що багато

проектів до цього вже використовували подібний стиль, тому є можливість проаналізувати інші роботи, та створити більш коректний ігровий досвіт для гравця.

2 РОЗБІР ТЕХНІЧНИХ МОЖЛИВОСТЕЙ ТА ПІДГОТОЧА ФАЗА

2.1 Unity та основні інструменти для розробки

Unity — це інтегроване середовище розробки ігор, яке надає розробникам широкий спектр інструментів для створення ігор різних жанрів.

Unity внесла значний вклад у світову розробку ігор, забезпечуючи розробникам потужний інструментарій для створення якісних ігор різних жанрів. Основні аспекти внеску Unity в галузь розробки ігор включають:

Доступність та Зручність:

Unity надає інтуїтивний інтерфейс та простий спосіб роботи з різноманітними аспектами гри, що робить його доступним для широкого кола розробників, включаючи новачків.

Мультиплатформеність:

Unity дозволяє розробляти ігри для різних платформ, таких як PC, мобільні пристрої, консолі, віртуальна реальність та інші. Це спрощує процес портування та розширює аудиторію гравців.

Спільнота та Ресурси:

Є активна спільнота розробників Unity, яка обмінюється знаннями, досвідом та ресурсами. Unity Asset Store надає доступ до різноманітних ресурсів, таких як готові активи, інструменти та розширення, що полегшує розробку.

Гнучкість та Розширюваність:

Розробники можуть легко розширювати функціонал Unity за допомогою власних скриптів та розширень. Це надає велику гнучкість у реалізації різноманітних ідей та концепцій гри.

Візуальні Ефекти та Графіка:

Unity підтримує високий рівень графічних можливостей, що дозволяє розробникам створювати вражаючі візуальні ефекти та реалістичні графічні об'єкти.

Анімація та Звук:

Unity має потужні інструменти для створення реалістичної анімації та роботи зі звуком, що додає глибину та іммерсію в ігри.

Ігрова Фізика:

Unity включає вбудовану систему фізики, яка дозволяє реалістично моделювати рух та взаємодію об'єктів у грі.

Підтримка VR та AR:

Unity добре підтримує розробку віртуальної та розширеної реальності, що робить його ідеальним вибором для ігор з використанням цих технологій.

Інновації та Експерименти:

Unity надає розробникам можливість експериментувати з ідеями та швидко реалізовувати новаторські концепції гри, дозволяючи їм концентруватися на творчому процесі.

Усі ці фактори роблять Unity потужним інструментом у світі геймдеву, сприяючи розвитку як комерційних, так і незалежних проектів.

В цьому розділі ми розглянемо основні інструменти Unity, спрямовані на розробку платформерів.

Редактор сцен та об'єктів в Unity є ключовим інструментом для розробки ігор. Цей інтегрований інструмент дозволяє розробникам візуально створювати та редагувати вміст гри. Розглянемо основні функції редактора сцен та об'єктів в Unity:

Редактор сцен (Scene View):

Відображення сцени: Редактор сцен відображає вміст вашої гри у вигляді 3D-простору, де ви можете редагувати об'єкти та розміщувати їх у просторі.

Навігація: Ви можете переміщатися, обертати та масштабувати сцену для зручності редагування.

Огляд об'єктів: Ви бачите всі об'єкти, які присутні у сцені, включаючи персонажі, об'єкти оточення, світло та інші елементи.

Ієрархія об'єктів (Hierarchy):

Список об'єктів: Ієрархія показує список всіх об'єктів у поточній сцені. Вона відображає структуру об'єктів за їхніми батьківськими та дочірніми зв'язками.

Управління об'єктами: Ви можете вибирати, переміщати, обертати та змінювати властивості об'єктів, просто перетягуючи їх у ієрархії.

Інспектор об'єкта (Inspector):

Редагування властивостей: Інспектор дозволяє вам переглядати та редагувати властивості об'єктів, включаючи їхню позицію, розмір, матеріали, анімацію та інше.

Додавання компонентів: Ви можете додавати різноманітні компоненти до об'єктів, такі як колайдери, скрипти, світло, звукові джерела та багато іншого.

Створення та Розміщення Об'єктів:

Створення нових об'єктів: Розробники можуть створювати нові об'єкти, перетягуючи їх у сцену з розділу Assets або шляхом використання контекстного меню.

Розміщення об'єктів: Об'єкти можна просто перетягувати в сцені для розміщення їх у бажаному місці.

Огляд Сцени в Реальному Часі:

Програвання сцени: Розробники можуть перевіряти гру в реальному часі, просто натисканням кнопки "Play", щоб переконатися, як буде виглядати гра під час виконання.

2D-анімація в Unity надає можливість створювати живу та привабливу анімацію для персонажів, об'єктів та інших елементів у 2D-грі. Основні елементи 2D-анімації в Unity включають в себе використання спрайтів та Animator Controller. Розглянемо основні кроки для створення 2D-анімації:

Створення Спрайтів:

Спрайти: Спрайти - це 2D-графічні об'єкти, які використовуються для створення анімації. Ви можете використовувати графічні програми для створення спрайтів або імпортувати готові зображення у форматах, таких як PNG чи JPEG.

Створення Animator Controller:

Animator Controller: Це об'єкт, який дозволяє визначити та керувати різними анімаційними станами персонажа чи об'єкта. Для створення Animator Controller ви можете використовувати вікно Animator в Unity.

Створення Анімаційних Станів:

Анімаційні стани: Створюйте анімаційні стани для різних дій персонажа (наприклад, ходьба, стрибок, атака). Кожен стан представляє конкретну анімацію.

Налаштування Переходів:

Переходи між станами: Задайте умови, за яких персонаж буде переходити між анімаційними станами. Наприклад, перехід від стану "Ходьба" до "Стрибок" може відбуватися при натисканні клавіші стрибка.

Створення Ключових Кадрів (Keyframes):

Ключові кадри: Визначте ключові кадри для кожного анімаційного стану. Ключові кадри вказують на конкретний момент часу, де змінюються властивості спрайта (позиція, обертання, масштаб).

Використання Спрайтових Анімацій:

Спрайтові анімації: Unity дозволяє створювати анімації, перебуваючи в режимі редагування сцени. Ви можете переміщати та обертати спрайти на різних кадрах для створення плавної анімації.

Проходження Анімації:

Перевірка анімації: Запустіть гру або перевірте анімацію безпосередньо в редакторі, використовуючи інструмент відтворення анімацій.

2D-анімація в Unity надає гнучкість та зручність у розробці живої та захоплюючої графіки для 2D-ігор.

Unity має ряд потужних інструментів для реалізації фізики в 2D іграх. Нижче описано основні елементи та інструменти фізичного моделювання в Unity для 2D-проектів:

Rigidbody 2D:

Опис: Rigidbody 2D - це компонент, який додає фізичні властивості об'єкту в 2D-просторі. Він визначає масу, висоту, густину тіла та інші параметри, які впливають на його фізичну поведінку.

Використання: Рекомендується застосовувати Rigidbody 2D до об'єктів, які повинні рухатися або реагувати на фізичні сили.

Collider 2D:

Опис: Collider 2D - це компонент, який визначає форму об'єкта для обробки фізичних стикувань з іншими об'єктами.

Використання: Щоб забезпечити взаємодію об'єктів у просторі, до них застосовують Collider 2D.

Physics Materials 2D:

Опис: Матеріали фізики (Physics Materials) дозволяють налаштовувати параметри тертя та відскоку для об'єктів, які зіштовхуються.

Використання: Це корисно для створення різних ефектів стикувань, таких як затримка або гладкі відскоки.

Joint 2D:

Опис: Joint 2D - це компонент, який дозволяє з'єднувати два або більше Rigidbody 2D разом.

Використання: Використовується для створення різноманітних з'єднань між об'єктами, таких як поковзання, пружини, кріплення тощо.

Effector 2D:

Опис: Effector 2D - це компонент, який застосовує спеціальні фізичні впливи, такі як гравітація, обмеження об'єктів, зміна масштабу тощо.

Використання: Застосовується для створення унікальних фізичних ефектів на об'єктах.

Tilemap Collider 2D:

Опис: Tilemap Collider 2D дозволяє використовувати тайловані картки (Tilemaps) як Collider для об'єктів.

Використання: Зручний для створення фізично взаємодіючих об'єктів на основі тайлованих карт.

Ці компоненти і інструменти узгоджено працюють у 2D-середовищі Unity, надаючи розробникам багато можливостей для створення реалістичної та захоплюючої фізики в своїх іграх.

2.2 Збір ресурсів та підготовча фаза

Для розробки даного проекту потрібно зібрати велику кількість асетів які будуть використовуватись, для подальшого використання. Збір асетів, таких як графічні ресурси, звукові ефекти та анімації, є ключовим етапом у розробці гри. Сама гра має стати візуально привабливою та захопливою завдяки правильному вибору та використанню асетів.

Основним джерелом яким я буду користуватись для візуальних асетів буде сайт «https://craftpix.net/sets/cyberpunk-platformer-asset-pixel-art/», тут я візьму Tilemaps, спрайти головних персонажей, та ворогів.

Першим набором ресурсів який буду використовувати це буде набір спрайтів, для мапи. Набір ресурсів було взято з «https://craftpix.net/freebies/free-industrial-zone-tileset-pixel-art/» та називається, «INDUSTRIAL ZONE TILESET PIXEL ART». В цьому наборі міститься, загальний обсяг візуального декору, анімацій, та фонів.

Візуально сам Tilemaps відображений нижче.

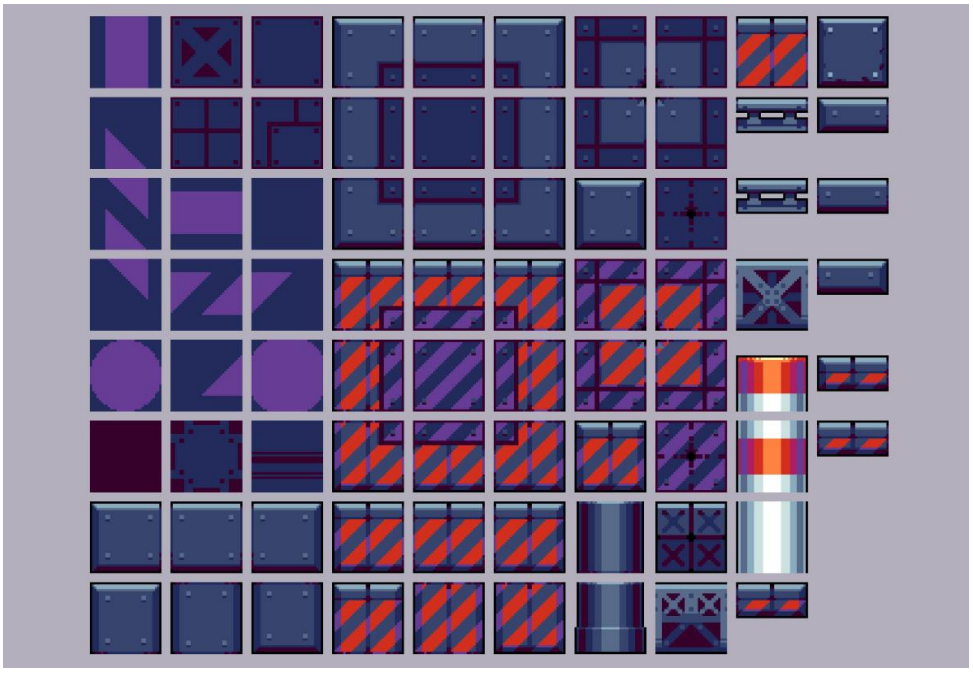


Рис. 2.1. Сітка для мапи

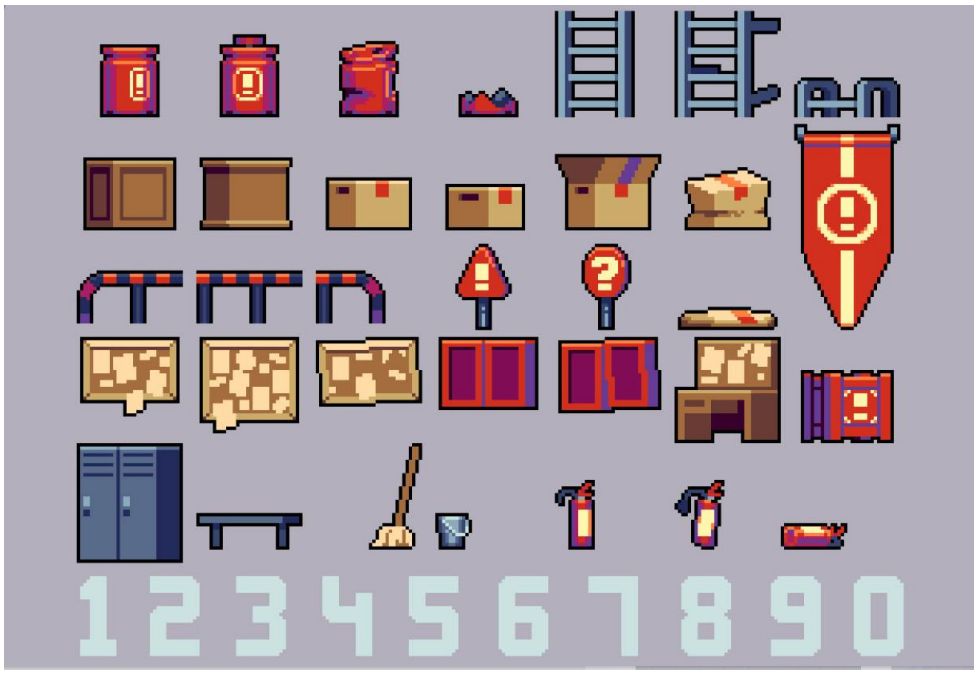


Рис. 2.2. Декоративні об’єкти

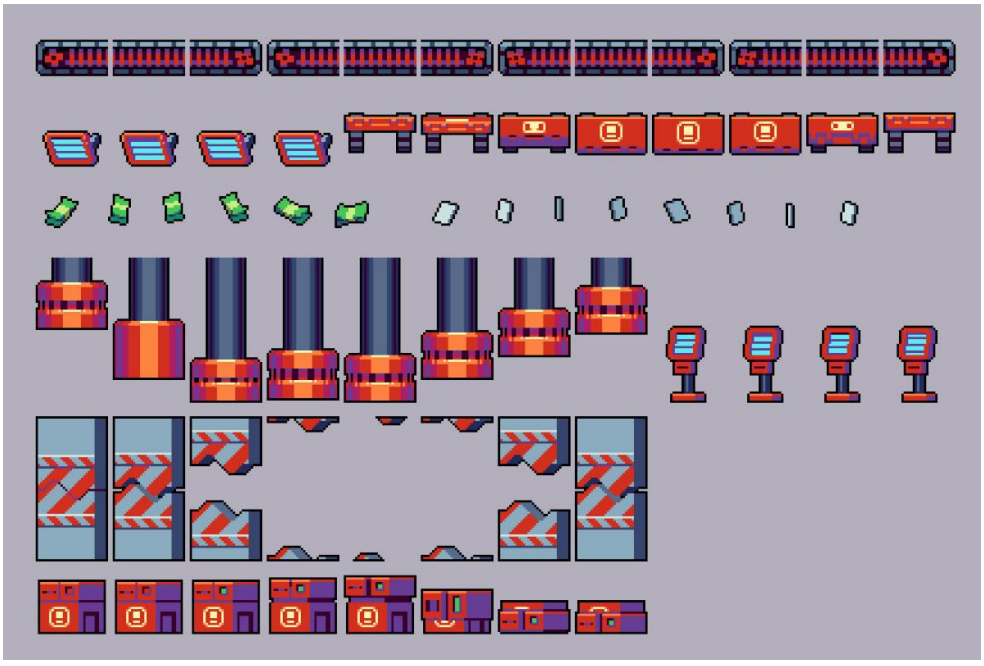


Рис. 2.3. Об’єкти які мають анімації, та є функціональними.

Для коректної роботи з цим набором ресурсів потрібно коректно налаштувати інструменти Unity. Першим чином потрібно інтегрувати ці спрайти в середовище розробника, для цього потрібно у кожного спрайта встановити набір правил:

1. Texture Type - Sprite(2d and UI),

2. Sprite Mode – Single ,
3. Pixels Per Unit – 32,
4. Mesh Type – Tight,
5. Extrude Edges – 1,
6. Pivot – Center,
7. sRGB – true,
8. Alpha Source – Input Texture Alpha,
9. Alpha is Transparency – True,
10. Read/Write – false,
11. Generate Mipmap – false,
12. Warp mode - Clamp,
13. Filter Mod – no filter,

Наступним кроком розробки є створенням зручного інструменту для, розробки рівнів. Для початку потрібно створити сітку для подальшої зручності, для цього у верхньому меню потрібно обрати "Window" -> "2D" -> "Tilemap" після чого буде створено об`єкт Grid в якому буде одразу Tilemap який я перейменую в «Platform», та додатково ще створю Tilemap та назву «Backgraund». Ці дві сітки матемуть основну роль в подальшому створенні рівня.

До сітки Platform я встановив такий перелік параметрі:

1. Transform
    1.1 Position (xyz) – (0,0,0),
    1.2 Rotation (xyz) – (0,0,0),
    1.3 Scale (xyz) – (1,1,1),
2. Tilemap ,
    2.1 Animation Frame Rate – 1
    2.2 Tile Anchor (xyz) – (0.5,0.5,0.5),
3. TileMap Renderer ,
    3.1 Order in Layer – 10,
4. Tilemap Collider ,
5. Rigidbody 2D ,

5.1 Simulated – true,

5.2 Mass – 1,

5.3 Gravity Scale – 0,

5.4 Freeze Position and Rotation (xyz) – all True,

6. Composite Collider

До сітки Backgraund встановлено такий перелік параметрів

1. Transform
   1.1 Position (xyz) – (0,0,0),
   1.2 Rotation (xyz) – (0,0,0),
   1.3 Scale (xyz) – (1,1,1),
2. Tilemap ,
   2.1 Animation Frame Rate – 1
   2.2 Tile Anchor (xyz) – (0.5,0.5,0.5),
3. TileMap Renderer ,
   3.1 Order in Layer – -1,

Наступним кроком було створення зручної палітри спрайтів для подальшого користування, для цього було створено нову палітру, з назвою «PromZone», до неї було створено додатково папки «PromeZoneTitle», та «Titles».

Візуально палітра виглядає так.

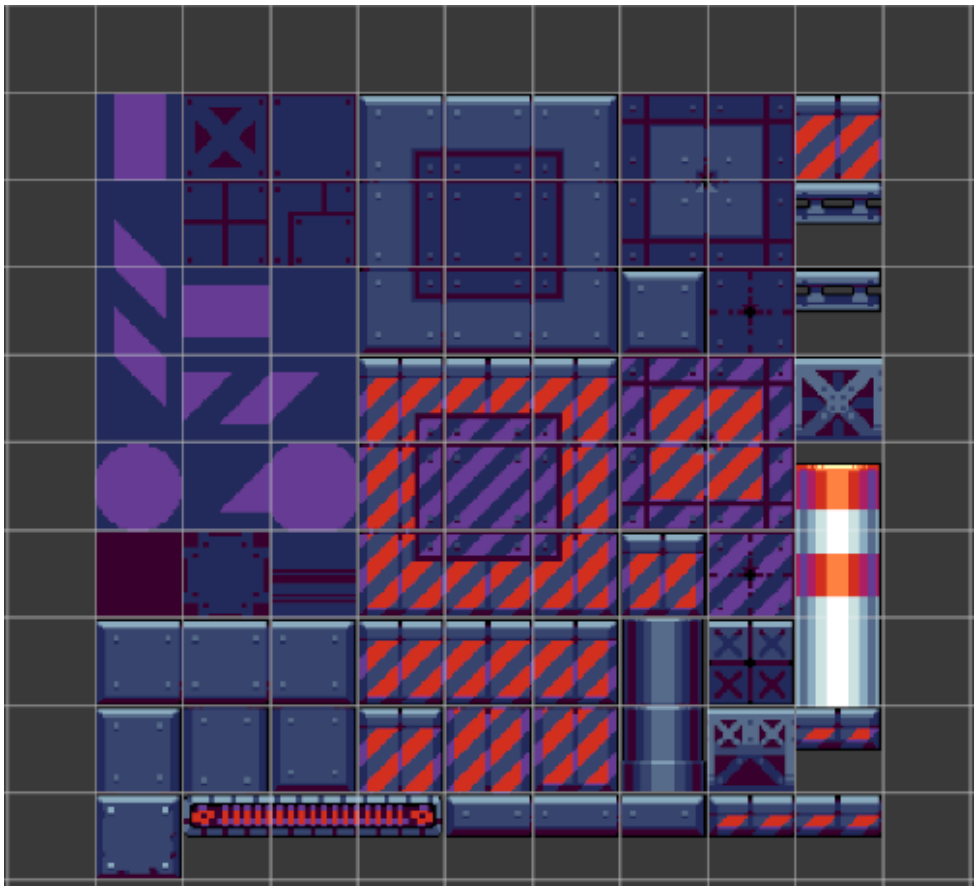


Рис. 2.4. Палітра спрайтів

Наступним в списку ресурсів має бути перелік головних персонажів для гри, асети було вирішено взяти з того самого сайту що і до цього і від тих самих авторів. Для використання було вирішено взяти «3 CYBERPUNK CHARACTERS PIXEL ART» з ресурсу «https://craftpix.net/freebies/free-3-cyberpunk-characters-pixel-art/». Сам пак в собі містить три персонажі, з набором анімацій, а саме чоловік, жінка і кіборг. Для початку можна зосередитись тільки на чоловікові.

Сам набір візуально виглядає так .

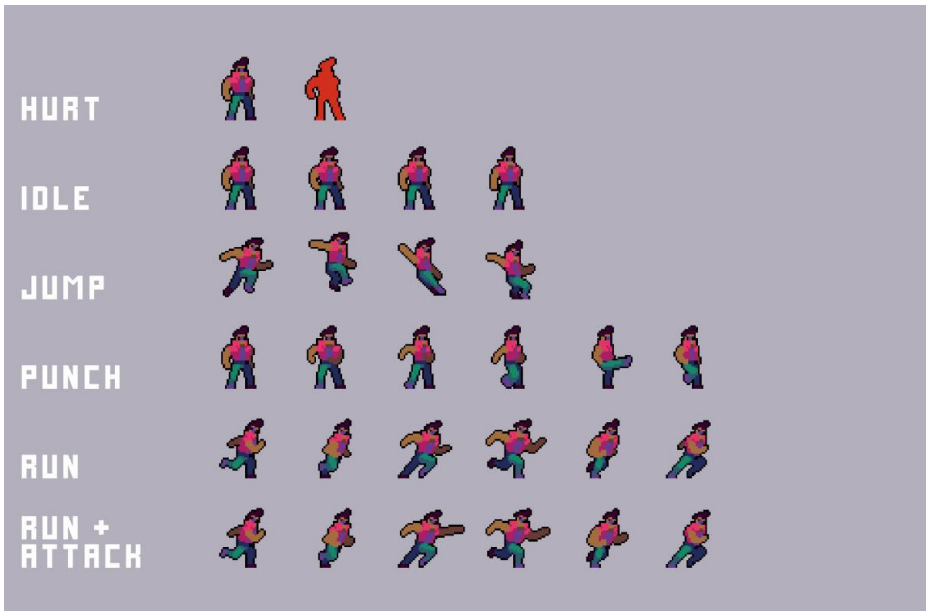


Рис. 2.5. Рух головного персонажа

Рис. 2.6. Інші дії головного персонажу

Для основного набору спрайтів головного персонажу потрібно налаштувати коректно параметри спрайтів, а після створити набір анімацій.

Для початку потрібно прописати параметри для всіх анімацій а саме вони мають бути:

1. Texture Type - Sprite(2d and UI),

2. Sprite Mode – Single ,
3. Pixels Per Unit – 32,
4. Mesh Type – Tight,
5. Extrude Edges – 1,
6. Pivot – Center,
7. sRGB – true,
8. Alpha Source – Input Texture Alpha,
9. Alpha is Transparency – True,
10. Read/Write – false,
11. Generate Mipmap – false,
12. Warp mode - Clamp,
13. Filter Mod – no filter,

Також потрібно, завантажити ресурси для ворогів, їх також було вирішено взяти з того самого ресурсу, та від тих самих авторів. Посилання на ресурс – «https://craftpix.net/freebies/free-city-enemies-pixel-art-sprite-sheets/», назва паку «CITY ENEMIES PIXEL ART SPRITE SHEETS». Сам пак має великий перелік персонажів але для подальшого використання було вирішено обрати одного, а саме персонажу ближнього бою.

Візуальне відображення нижче.

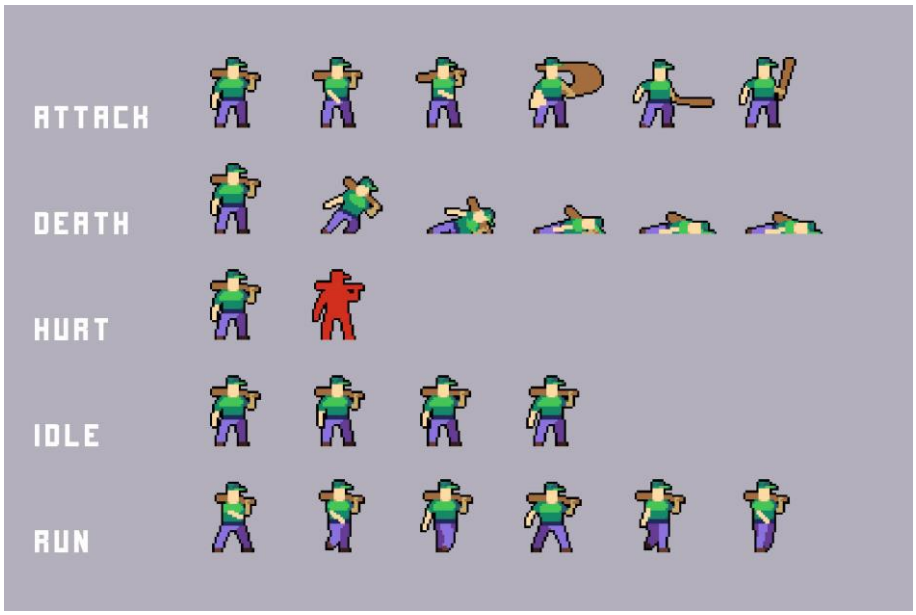


Рис. 2.7. Анімації до ворогів

Для основного набору спрайтів ворогів потрібно налаштувати коректно параметри спрайтів, а після створити набір анімацій.

Для початку потрібно прописати параметри для всіх анімацій а саме вони мають бути:

1. Texture Type - Sprite(2d and UI),
2. Sprite Mode – Single ,
3. Pixels Per Unit – 32,
4. Mesh Type – Tight,
5. Extrude Edges – 1,
6. Pivot – Center,
7. sRGB – true,
8. Alpha Source – Input Texture Alpha,
9. Alpha is Transparency – True,

10. Read/Write – false,
11. Generate Mipmap – false,
12. Warp mode - Clamp,
13. Filter Mod – no filter,

На цьому можна закінчити основний збір ресурсі, далі можна вже розпочинати розробку механік та рівнів. При потребі додаткових ресурсів, є можливість повернутися до нього та додавати нові деталі в ході розробки, на разі цього достатньо.

3 РОЗРОБКА МЕХАНІК ТА КОНСТРУЮВАННЯ РІВНІВ

3.1 Розробка основних механік

Розробка основних механік гри - це ключовий етап в створенні власної гри. Слід ретельно продумати та реалізувати функціонал, який забезпечить унікальний та захоплюючий геймплей. Ось деякі загальні механіки, які мають бути розроблені під час цього етапу розробки:

1. Рух персонажа:

1.1 Додайте можливість рухатися вліво та вправо.

1.2 Реалізуйте механіку стрибків для подолання перешкод та прогалин.

2. Фізика та колізії:

2.1 Забезпечте реалістичну фізику руху та стрибків.

2.2 Використовуйте колізії для взаємодії персонажа з платформами, перешкодами та іншими об'єктами.

2.3 Реалізуйте систему уникнення та обходу перешкод.

3. Анімації:

3.1 Створіть анімації для різних дій персонажа, таких як ходьба, стрибки, опинки та атаки.

3.2 Використовуйте анімації для вираження різних станів персонажа (наприклад, біль при травмі, радість при досягненні мети).

4. Вороги та боротьба:

4.1 Додайте ворогів, які перешкоджають гравцеві.

4.2 Реалізуйте систему боротьби або уникання ворогів.

5. Рівні:

5.1 Створіть різноманітні рівні з цікавими ландшафтами та перешкодами.

5.2 Використовуйте промисловий стиль заводського середовища для створення різноманітних та захоплюючих рівнів.

Для самого початку потрібно створити головного персонажа, для цього в менеджері об’єктів створимо пусту сущність, та дамо їй назву «Player». Далі до «Player» додамо такі атрибути як: Animator, Sprite Render, Box Collider 2d, Rigid Body 2d, та створимо файл Player.cs який додамо також до атрибутів, та вході розробки будемо модифікувати його, для прописування взаємодій з навколишнім середовищем.

Для подальшої розробки нам потрібно створити хоч якесь поле для тестування. На етапі підготовки було підготовлено сітку для створення мапи тому можна вже намалювати хоч якийсь ландшафт. Було намальовано полоску на якій ми будемо випробовувати основні механіки руху.

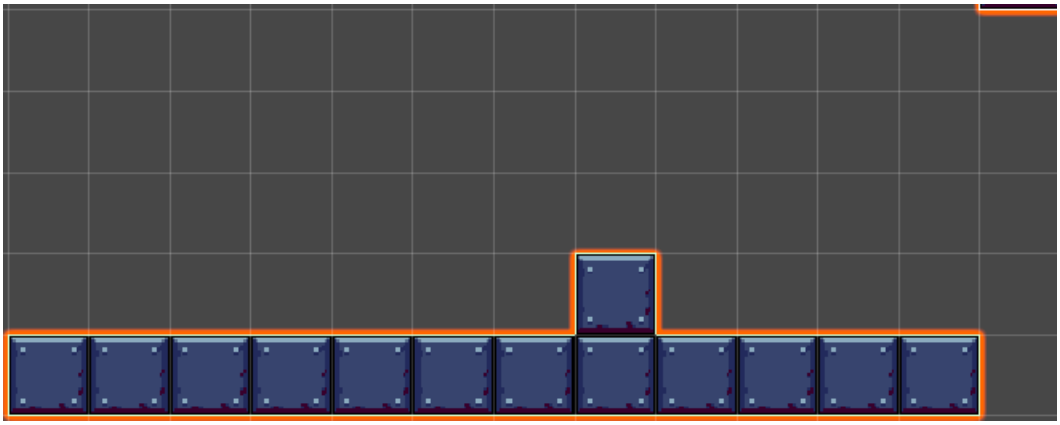


Рис. 3.1. Початкове поле для тестувань

Механіка руху в грі є ключовою частиною геймплею і може бути різноманітною в залежності від задумувань та концепції вашої гри. Її ми маємо реалізувати першою, але для більш зручної розробки було вирішено, спочатку реалізувати початкові анімації, щоб потім активувати їх з скриптів, при виконанні різних дій персонажу.

Перша анімація яка буде розроблена це анімація покою.

Рис. 3.2 Анімація покою

Для анімації покою в самому юніті ми створимо нову анімацію, та назвемо її «idel», сама анімація буде прив’язана до об’єкту «Player», та матиме зацикленість ця анімація буде доданою до Аніматор «Player» і буде початковою і основною анімацією для «Player», після ініціалізації рівня.

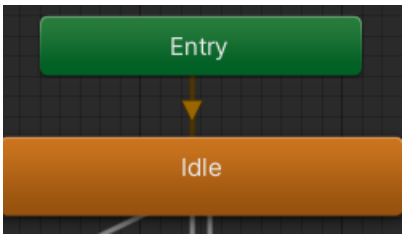


Рис. 3.3. Початкова ієрархія аніматору

Ми вже маємо початкову візуалізацію нашого головного героя, виглядає вона приблизно так.

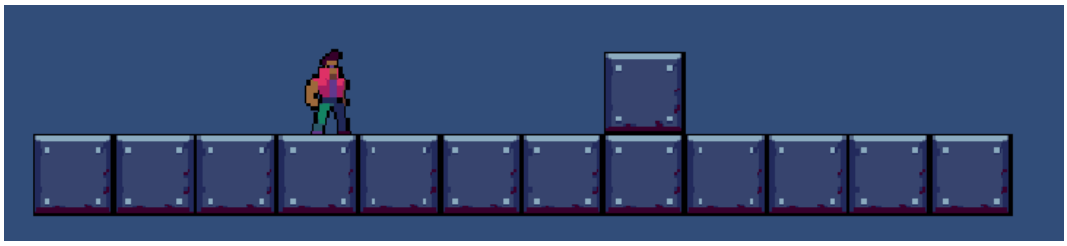


Рис. 3.4. Візуалізація Головного персонажу в стані покою.

Наступна анімація це анімація бігу персонажу.

Рис. 3.5. Анімація бігу

Наступна анімація яка буде опрацьована це анімація руху, для неї як і для анімації покою, ми створимо в юніті нову анімацію «Run», та матиме

зацикленість ця анімація буде доданою до Аніматор «Player» і використовуватись при переміщені персонажу, в разі його зупинки анімація переключиться на «idel». Для цієї анімації потрібні особливі умови вокиристання, тому в аніматорі потрібно створити умови «Run», та «Jump». В собі вони мають зберігати особливі значення «Run = true», та «Jump = false», при таких параметрах параметрах буде активуватись анімація бігу.

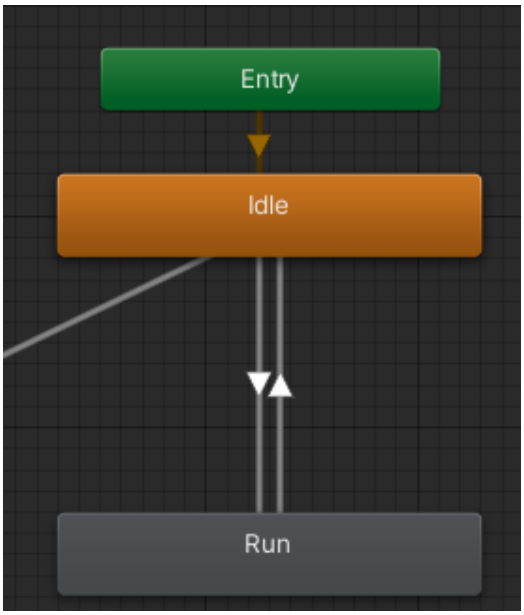


Рис. 3.6. Ієрархія аніматору з анімацією бігу.

На тестовому полі наш персонаж вже має візуалізацію переміщення, далі потрібно додати анімацію стрибку, та після вже прописати скрипт який надав би можливість для персонажу змінювати позицію в просторі.

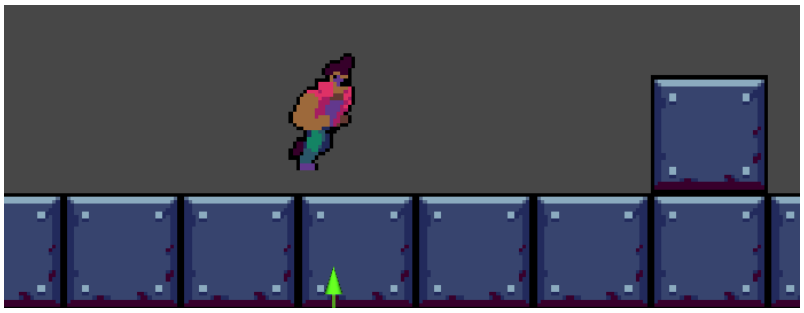


Рис. 3.7. Візуалізація переміщення

Наступна анімація яку потрібно додати, та є фінальною в механіці руху це анімація стрибку.

Рис.3.8. Анімація стрибку

Для анімації стрибку в самому юніті ми створимо нову анімацію, та назвемо її «jump», сама анімація буде прив’язана до об’єкту «Player», та матиме зацикленість ця анімація буде доданою до Аніматор «Player» і використовуватись з будь якого стану в незалежності біжить, стоїть, чи атакує персонаж. Для цього в ієрархії аніматору потрібно створити атрибут будь якого стану, а до самої анімації додати умови виклику «jumpAnimTrigger», логіку цієї умови ми пропишемо потім але для цього в самому «Player» створимо нову сущність «JumpTrigger», та створимо новий файл «JumpTrigger.cs»

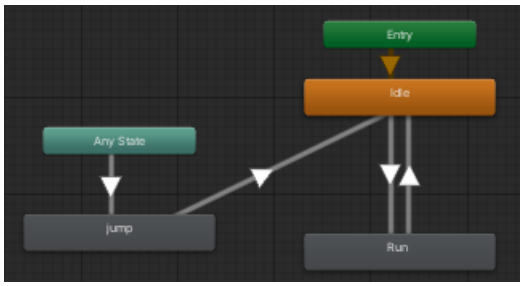


Рис. 3.9. Фінальна ієрархія аніматору для руху

Можна вже переходити до етапу написання коду, в самому коді мають бути реалізовані механіки руху, активації умов для зміни анімації, зміна вектору руху, та стрибок, заздалегідь відключення безкінечного стрибка. Для відключення безкінечного стрибка ми і створювали «JumpTrigger» , та «JumpTrigger.cs»

Приступимо до реалізації коду в файлі «Player.cs» та «JumpTrigger.cs»

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

public class Player : MonoBehaviour
{
    public Animator AnimatorControler;
    public float Speed = 5f;
    public float RealSpeed;
    public float JumpSpeed = 5f;
    private Rigidbody2D rb;
    public JumpTrigger jt;
    Unity Message
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    Unity Message
    void FixedUpdate(){

        float RunFloat = Input.GetAxis("Horizontal");
        RealSpeed = rb.velocity.y;
        if(RunFloat != 0){
            transform.position += new Vector3(Speed*RunFloat*0.01f, 0, 0);

        }
        if(RunFloat != 0&&jt.isGraunded){
            AnimatorControler.SetBool("Run", true);
        }
        if(RunFloat>0){
            Vector3 theScale = transform.localScale;
            theScale.x = 1;
            transform.localScale = theScale;
        }
        if(RunFloat<0){
            Vector3 theScale = transform.localScale;
```

Рис. 3.10. Код

```
            theScale.x = -1;
            transform.localScale = theScale;
        }
        if(RunFloat == 0){
            AnimatorControler.SetBool("Run", false);
        }



        if(!jt.isGraunded){
            if(!AnimatorControler.GetBool("Jump")){
                AnimatorControler.SetTrigger("JumpAnimTrigger");
            }

            AnimatorControler.SetBool("Jump", true);
            AnimatorControler.SetBool("Run", false);


        }
        if(jt.isGraunded){
            AnimatorControler.SetBool("Jump", false);
        }
    }
    Unity Message
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.Space)&&jt.isGraunded){
            rb.AddForce(transform.up*JumpSpeed, ForceMode2D.Impulse);


        }

    }
}

```

Рис. 3.11. Код

Спочатку потрібно розібрати файлі «Player.cs» почнемо з буплічни залежностей: public Animator AnimatorControler – звернення до аніматору public float Speed = 5f – швидкість персонажу, public float RealSpeed – реальна швидкість персонажу, public float JumpSpeed = 5f – швидкість стрибку, private Rigidbody2D rb – звернення та доступ до колайдеру «Player», public JumpTrigger jt звернення до «JumpTrigger» , та «JumpTrigger.cs».

Було вирішено використовувати метод FixedUpdate() для більш плавного руху персонажу, частота оновлень цього методу значно більша ніж частота методу по за умовчуванням.

Мета цього коду полягає в тому щоб при використанні методу transform.position та Input.GetAxis("Horizontal") задавати умову руху та, активації анімацій. Умови які відображенні коду в залежності від вводу гравця змінюють напрямок руху, швидкість та анімацію. Єдина проблема яка була при написанні коду це безкінечний стрибок, для вирішення цієї проблеми було створено колайдер «JumpTrigger», який фіксував момент коли персонаж немає контакту з землею і блокував можливість стрибку ось код з файлу «JumpTrigger.cs» який опрацьовував цей процес.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class JumpTrigger : MonoBehaviour
6   {
7       public bool  isGraunded;
8       public string name;
        Unity Message
9       private void OnTriggerEnter2D(Collider2D collision)
10      {
11          isGraunded = true;
12          name = collision.name+" в зоне";
13
14      }
        Unity Message
15      private void OnTriggerExit2D(Collider2D collision)
16      {
17          isGraunded = false;
18          name = collision.name+" Не зоне";
19
20      }
21  }
22
```

Рис. 3.12. Код

Цей код відповідав за перевірку позиції персонажу в просторі да не дає змогу використання безкінечного стрибку public bool isGraunded – є інформатором контакту персонажу з поверхнею, public string name – фіксує інформацію до чого саме торкається персонаж.

На разі ми вже маємо готову механіку руху персонажу і можемо нею скористуватись на тестовому полі, тестування механік після їх створення є одною з ключових частин розробки, тому тестуванню буде виділено сновну частину розробки.

Рис. 3.13. Тестування руху та стрибку

Тестування механіки руху надало задовільний результат, для подальшої реалізації цієї механіки в проекті, тому було вирішено зупинитися на цьому та перейти до наступних механік.

Наступні два елементи які будуть опрацьовані це динаміка оточення, та взаємодія оточення на гравця. Елементи які я б хотів додати та опрацювати це конвеєр, який би переміщував головного персонажа в просторі в незалежності дій персонажу, та рухаючі платформи які б змінювали свою позицію, та були б чудовим елементом в побудові рівнів. Ці два елементи нададуть змогу в більш цікавому створенні рівнів.

Перший елемент до опрацювання це конвеєрна смуга.

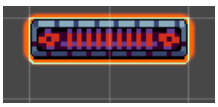


Рис. 3.14. Конвеєрна смуга

Конвеєрна смуга буде опрацьована як звичайний об’єкт сітки, та для її використання буде створено додаткову сутність «Triger» вякому будуть створюватись «PushTriger» головне завдання яких полягатиме в тому щоб штовхати гравця в тому чи іншому напрямку. Тобто спрайт конвеєру буде використовуватись як звичайна частина ландшафту, поверх якого буде додано тригер який і буде виконувати функціонал конвеєру. До тригеру потрібно

створити новий скипт, для цього створимо новий файл, який буде називатись «PushTrigger.cs» в ньому ми маємо прописати такий код.

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



public class PushTrigger : MonoBehaviour
{
    public float way = -1f;  // Сила пушу

    // Викликається, коли інший об'єкт входить в тригер
    Unity Message
    private void OnTriggerStay2D(Collider2D other)
    {

        if (other.name=="Player")  // Перевіряємо, чи це гравець
        {
            Rigidbody2D playerRb = other.GetComponent<Rigidbody2D>();

            if (playerRb != null)
            {
                // Застосовуємо силу пушу до гравця
                playerRb.transform.position += new Vector3(5f*0.5f*0.01f*way, 0, 0);

            }
        }
    }
}
```

Рис.3.15. Код

Цей код відповідає за переміщення персонажу в просторі без врахування, вводу з боку гравця, public float way = -1f – є параметром який відповідає за напрямок та швидкість руху, якщо змінювати значення з від’ємного на позитивне, або з позитивного на від’ємне, буде змінюватись напрямок руху персонажу з ліва на право, з право на ліво

Наступним елементом для реалізації має стати платформа яка б рухалась в просторі не зважаючи на команди гравця, ця платформа матиме рухатись як вертикально так і горизонтально.

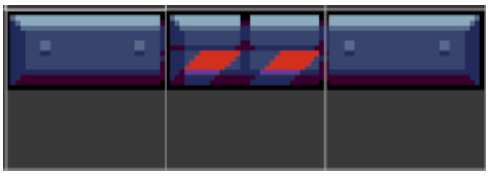


Рис.3.16. Платформа

Для створення цієї платформи створимо сутність яка має назву «Platforms» до цієї сутності ми будемо створювати та додавати платформи які будуть називатись «Plat», до цієї сутності потрібно додати колайдер та фізику тіла, додатково для візуалізації додамо спрайти початку, середини, та кінця платформи. Далі створимо файл «PlatformMovement.cs» в якому опишемо код переміщення платформи в ігровому просторі. Потрібно зазначити що код має реалізувати можливість руху платформи як вертикально так і горизонтально, та надати можливість зовнішнього впливу для зміни напрямку руху платформи в навколишньому просторі гри. Ось код який має відповідати даним запитам, технічного завдання, яке було наведене вище в тексті.

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

using UnityEngine;

public class PlatformMovement : MonoBehaviour
{
    public float speed = 2f; // Швидкість руху платформи
    public float leftBound = -5f; // Ліва межа руху
    public float rightBound = 5f; // Права межа руху
    public bool HV = true;

    // Оновлення в кожному кадрі
    Unity Message
    private void Update()
    {
        if(HV){
            MovePlatformH(); // Викликаємо метод руху платформи
        }
        if(!HV)  {
            MovePlatformV(); // Викликаємо метод руху платформи
        }
    }

    // Метод для руху платформи
    private void MovePlatformH()
    {
        // Визначаємо напрямок руху
        Vector3 movement = Vector3.right * speed * Time.deltaTime;
        transform.Translate(movement);

        // Застосовуємо зміщення до платформи


        // Перевіряємо, чи платформа вийшла за ліву чи праву межу
        if (transform.position.x < leftBound||transform.position.x == leftBound)
        {
            //Debug.Log(transform.position.x);
```

Рис. 3.17. Код

```
38
39              // Якщо так, змінюємо напрямок руху, множачи на -1
40              speed *= -1;
41          }
42          if(  transform.position.x > rightBound||transform.position.x == rightBound){
43              speed *= -1;
44             // Debug.Log(transform.position.x);
45          }
46      }
47      private void MovePlatformV()
48      {
49          // Визначаємо напрямок руху
50          Vector3 movement = Vector3.up * speed * Time.deltaTime;
51          transform.Translate(movement);
52
53          // Застосовуємо зміщення до платформи
54
55
56          // Перевіряємо, чи платформа вийшла за ліву чи праву межу
57          if (transform.position.y < leftBound||transform.position.y == leftBound)
58          {
59              //Debug.Log(transform.position.x);
60
61              // Якщо так, змінюємо напрямок руху, множачи на -1
62              speed *= -1;
63          }
64          if(  transform.position.y > rightBound||transform.position.y == rightBound){
65
66              speed *= -1;
67             // Debug.Log(transform.position.x);
68          }
69      }
70  }
```

Рис. 3.18. Код

Цей код відповідає за рух платформи в просторі, розглянемо публічні атрибути, та внутрішні функції. Атрибут public float speed відповідає за швидкість та напрямок руху, атрибути public float leftBound, public float rightBound відповідають за точки маршруту в просторі, звдки і куди має курсувати платформа в ігровому просторі, атрибут public bool HV відповідає за напрямок руху платформи в просторі, горизонталь, вертикаль.

Наступна механіка яку ми маємо розглянути це механіка бою головного персонажу, та механіка здоров'я, для них потрібно додати та створити анімації удару, та отримання шкоди, додатково потрібно створити окремі скрипти для атаки та слідкуванням здоров'я, надалі ми зможемо їх використовувати, для прописування логіки ворогів, або створювати нові скрипти, при цьому використовуючи ці як основу.

Перша анімація яка підійде до опрацювання це анімація удару для неї створимо, в юніті, анімацію «Hit», та додамо її до аніматору.

Рис. 3.19. Анімація атаки.

Також потрібно змінити ієрархію аніматору гравця та додати тригер атаки, тепер вона виглядає так

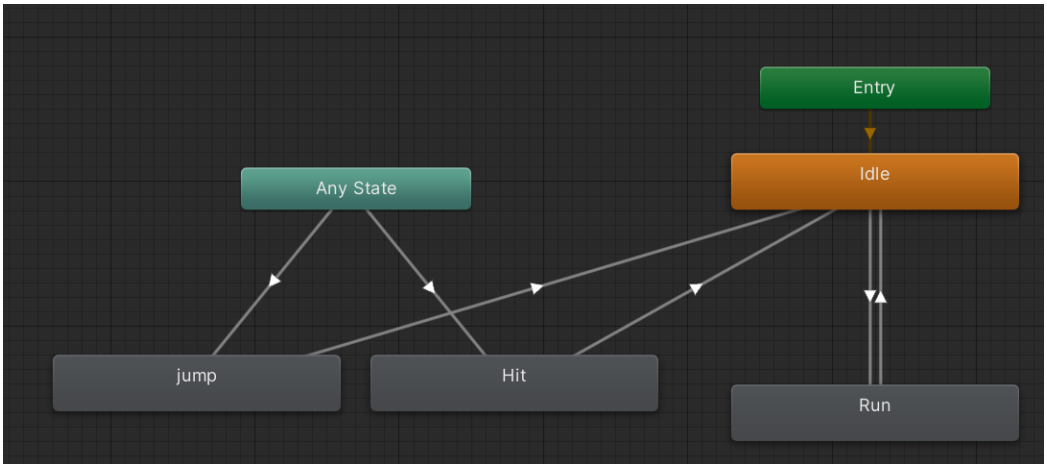


Рис.3.20. Ієрархія аніматору з атакою

Наступний крок це написання коду удару, для цього потрібно додати, новий файл «PlayerAttack.cs» та вньому описати логіку удару, код буде не повний, бо в майбутньому потрібно описати функцію отримання шкоди, для цього буде створено окремий файл. Нижче описано код.

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


public class PlayerAttack : MonoBehaviour
{
    public float attackRange = 1.5f;  // Дальність атаки
    public LayerMask enemyLayer;      // Шар, на якому перебувають вороги
    public Animator animator;
    public PlayerHealth PlayerHealth;

    // Оновлюється кожен кадр
    Unity Message
    void Update()
    {
        // Перевірка натискання клавіші атаки (припустимо, що це просто лівий клік миші)
        if (Input.GetKeyDown(KeyCode.LeftAlt))
        {
            // Виклик функції атаки
            Attack();
        }
    }

    // Функція атаки
    void Attack()
    {
        // Отримати всі колайдери в заданій дальності та шарі
        Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(transform.position, attackRange, enemyLayer);
        animator.SetTrigger("Attack");
        // Обробка кожного ворога, з яким персонаж стикається
        foreach (Collider2D enemy in hitEnemies)
        {
            // Тут можна додати логіку зменшення здоров'я ворога, анімацію атаки тощо.
            //Debug.Log(enemy.name);
            enemy.GetComponent<EnemyHealth>().TakeDamage(25);
        }
    }
```

Рис. 3.21. Код

```

    // Допоміжна функція для візуалізації діапазону атаки (відобразиться в сцені в редакторі Unity)
    Unity Message
    private void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, attackRange);
    }
}
```

Рис. 3.22. Код

Цей код передбачає, що персонаж атакує, коли гравець натискає лівий альт на своїй клавіатурі. Функція атаки перевіряє всі колайдери в заданій дальності та шарі та обробляє їх. Додатково потрібно буде дописати функцію для врахування шкоди, завданої ворожому персонажі, для цього буде створено новий файл, в якому буде реалізована ця логіка.

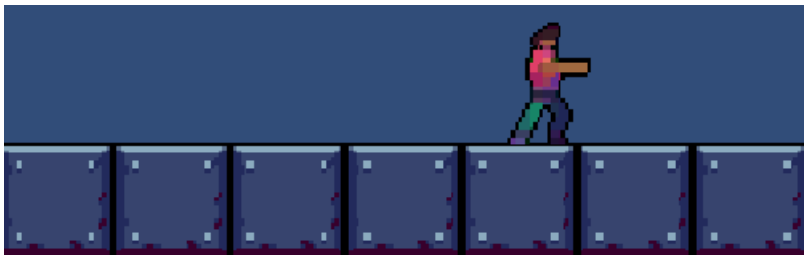


Рис. 3.23. Візуалізація бою персонажу.

Наступним що потрібно реалізувати це отримання шкоди персонажем для цього з початку потрібно додати анімацію отримання удару, для цього в юніті створемо нову анімацію, та додамо до аніматору головного персонажу, додатково пропишемо умову виклику анімації.

Рис. 3.24. Анімація шкоди.

Додатково внесемо, зміни до ієрархії зараз вона виглядає так.

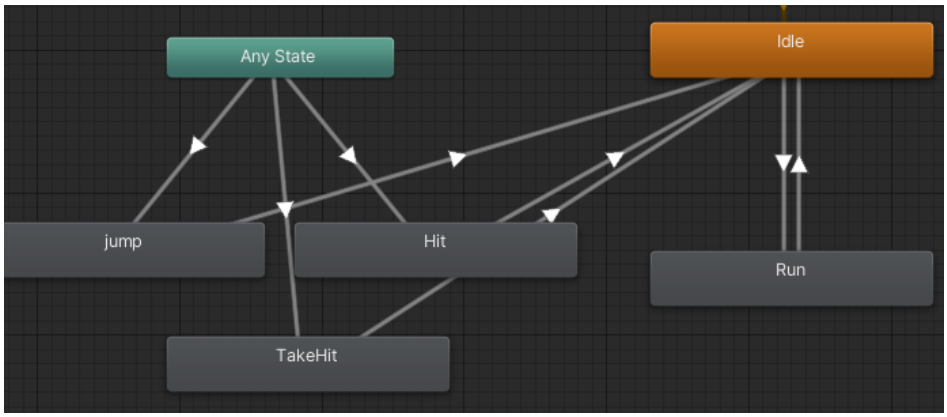


Рис. 3.25. Кінцева ієрархія аніматору гравця.

Та додатково було створено файл з кодом для опису отримання урону, для тесту була надана, можливість шкодити собі на правий альт. Ось код для реалізації шкоди гравцю.

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

using UnityEngine;

public class PlayerHealth : MonoBehaviour
{
    public float maxHealth = 100f;  // Максимальне здоров'я гравця
    private float currentHealth;     // Поточне здоров'я гравця
    public float x = 0f;
    public float y = 0f;
    public Animator animator;
    public Text displayText;
    Unity Message
    private void Start()
    {
        currentHealth = maxHealth;   // Встановлюємо поточне здоров'я на максимальне при старті гри
    }

    // Метод для отримання урону
    Unity Message
    void Update()
    {
        // Перевірка натискання клавіші атаки (припустимо, що це просто лівий клік миші)
        if (Input.GetKeyDown(KeyCode.RightAlt))
        {
            // Виклик функції атаки
            TakeDamage(10f);
        }
    }
    public void TakeDamage(float damage)
    {
        // Зменшуємо поточне здоров'я на величину отриманого урону
        currentHealth -= damage;
        animator.SetTrigger("TakeHit");
        // Перевіряємо, чи гравець помер
```

Рис.3.26. Код

```cs
            displayText.text = ""+currentHealth;
            if (currentHealth <= 0)
            {
                Die();
            }
        }

        // Логіка при смерті гравця
        private void Die()
        {
            // Додайте сюди логіку, яка відбувається при смерті гравця
            Debug.Log("Гравець помер!");
            transform.position = new Vector3(x, y, 0);

            currentHealth = maxHealth;
            displayText.text = ""+currentHealth;


        }
    }
```

Рис. 3.27. Код

В цьому коді описано отримання шкоди та, в разі смерті повернення на початкову позицію.

Наступні механіки це механіки ворогів, першим кроком буде – створення, нового ігрового персонажа, який має проявляти агресію до нашого основного ігрового персонажу. Для цього персонажу ми вже обрали підходящий набір ресурсів при зборі данних.

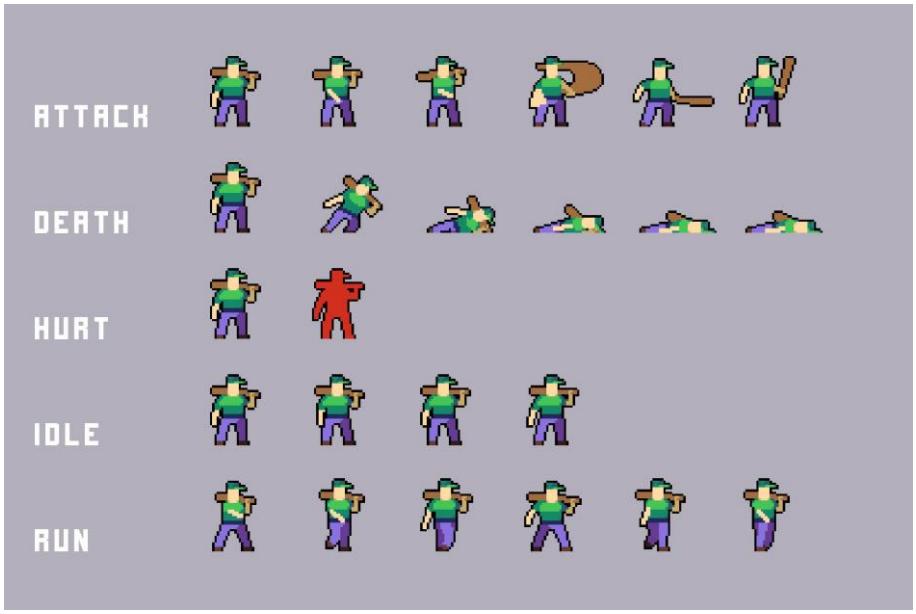


Рис. 3.28. Набір ресурсів для ворога

Першим кроком буде створення нової сутності яку ми назвемо «bots», в середині якої створимо ще одну сутність яка і буде відігравати роль ворога в нашій грі, та назвемо її «bot», далі по закінченню роботи над цією сутністю, вона буде клонуватись для збільшення кількості ворогів на ігровому полі. Сутність «bot», матиме майже такий набір параметрів як і в головного героя, окрім деяких відмінностей, наприклад вона не матиме можливості стрибку, бо їй це не потрібно.

Наступним кроком буде це інтеграція анімацій в юніті, для початку як з і головним персонажем створимо початкову анімацію «idle».

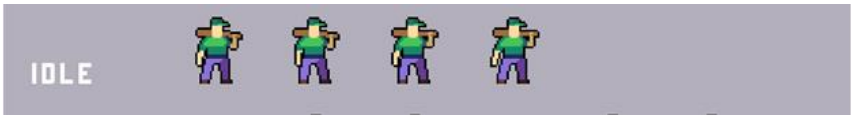


Рис. 3.29. Анімація – idle

Для цього в юніті створимо нову анімацію з аналогічною назвою, та додамо її до аніматору ворога.

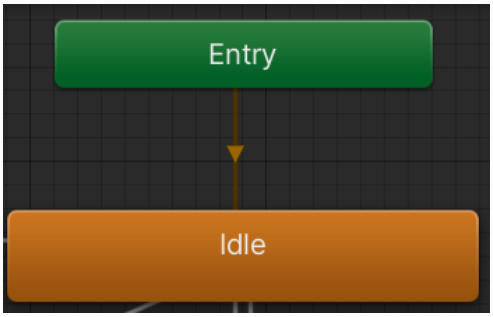


Рис. 3.30. Початкова ієрархія аніматору ворога

Наступна анімація яка буде додана це анімація бігу – «run», для неї створимо одно іменну анімацію в юніті, та додамо її до аніматору.

Рис. 3.31. Анімація бігу противника

Далі в самому аніматорі створимо логічну умову «run», для подальшого використання в коді.

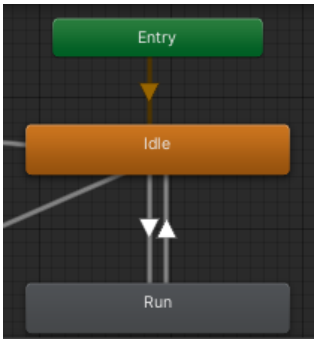


Рис. 3.32. Ієрархія аніматору ворога з бігом

Наступна анімація це анімація удару, для неї також створимо анімацію в юніті та додамо, її до аніматору.

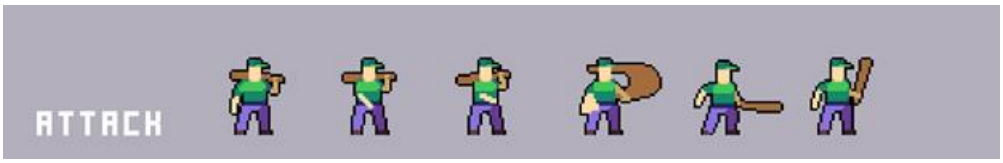


Рис. 3.33. Анімація атаки

Також створимо в аніматорі спеціальну умову виклику цієї анімації, та назвемо її «Hit». На разі аніматор відображає, в одно час, три стани противника та виглядає так.

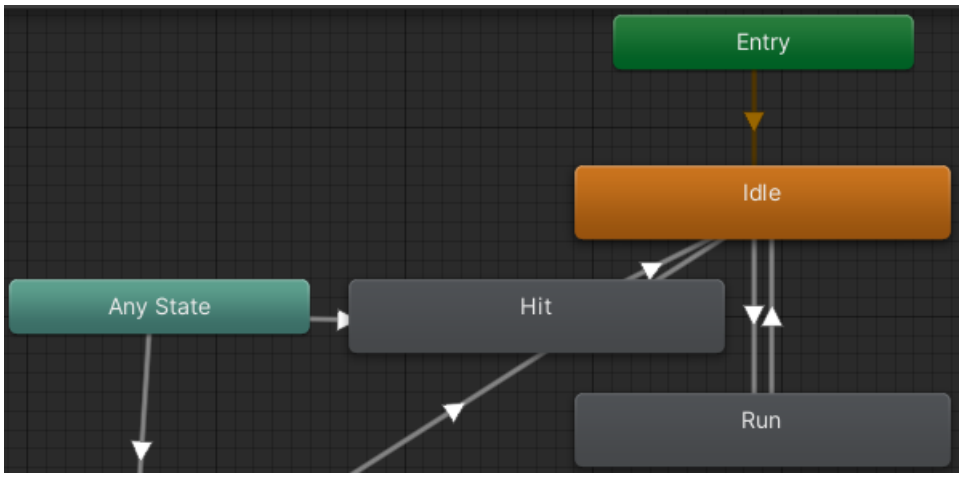


Рис. 3.34 Аніматор противника з доданою атакою.

Остання анімація це анімація отримання шкоди, під час нанесення шкоди від гравця, для створення цієї анімації, як і для всіх для цього стовчемо спеціальну анімацію в самому юніті, та додамо її до аніматору, який належить виключно до ворожого персонажу, ця анімація має виглядати ось так.

Рис. 3.35. Анімація отримання шкоди противником.

Додатково створимо тригер а самому аніматорі, при я кому буде викликатися анімація , отримання шкоди противником, від гравця. Фінальна версія аніматору виглядає ось так.

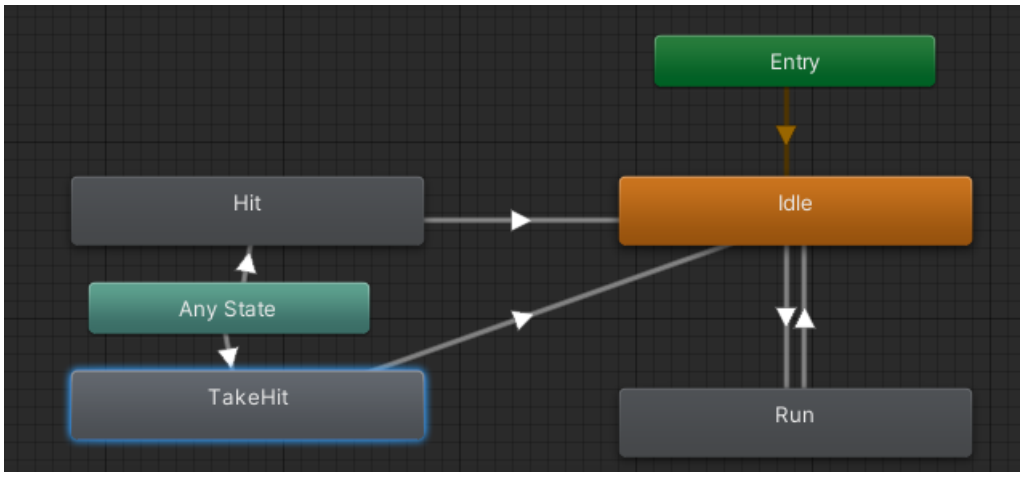


Рис. 3.36. Фінальна версія аніматору для ворожого персонажу.

На разі ми маємо готового ворожого персонажа зі всі візуальними характеристиками на, єдине що залишилось це прописати його поведінку, та логіку, його фундаментальних механік, які б могли оживити його, в ігровому просторі, та зробити не від'ємною складовою нашої гри. Ось так зараз виглядає наш противник, в ігровому просторі.

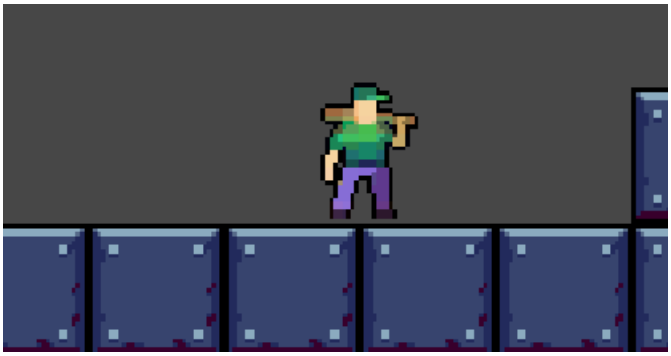


Рис. 3.37 Противник в ігровому просторі

Для початку треба зрозуміти яким основним набором механік має оволодіти цей персонаж. Наш ворог має розуміти, та мати змогу побачити гравця, та прокласти до нього най коротший маршрут. Додатково наш негативний персонаж має наносити, та отримувати шкоду.

Для початку можна написати код який би надав змогу нашому персонажу, знаходити маршрут до нашого гравця, цей код має надати змогу, нашому ворогу

побачити, та рухатись до нашого гравця. Для цього створими новий файл та назвемо його «Bot» і пропишемо ось такий код.

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

public class EnemyMovement : MonoBehaviour
{
    public float speed = 5f;  // Швидкість руху ворожого юніта
    public float detectionRange = 10f;  // Дальність, на яку ворожий юніт може виявити гравця
    public Animator Animator;
    private Transform player;  // Ссилка на трансформ гравця
    private Rigidbody rb;
    Unity Message
    void Start()
    {
        player = GameObject.FindWithTag("Player").transform;  // Знаходимо гравця за тегом "Player"
        rb = GetComponent<Rigidbody>();
    }

    Unity Message
    void Update()
    {
        // Перевірка, чи гравець знаходиться у діапазоні виявлення
        if (Vector2.Distance(transform.position, player.position) < detectionRange)
        {

            Vector2 direction = (player.position - transform.position).normalized;

        if(direction.x>0){
            Vector3 theScale = transform.localScale;
            theScale.x = 1;
            transform.localScale = theScale;
            Animator.SetBool("Run", true);
        }

        if(direction.x<0){
            Vector3 theScale = transform.localScale;
```

Рис. 3.38. Код

```
            theScale.x = -1;
            transform.localScale = theScale;
            Animator.SetBool("Run", true);
        }

            // Рухаємо ворожого юніта в напрямку гравця
            transform.Translate(direction * speed * Time.deltaTime);

        }
        if(Vector2.Distance(transform.position, player.position) > detectionRange){
            Animator.SetBool("Run", false);
        }
    }
}
```

Рис. 3.39. Код

Цей код знаходить гравця в зоні видимості ворога, та починає рух в бік гравця до тих пір поки гравець, знову не зникне з поля зору. Додатково підчас

руху використовується анімація бігу, яка була прописана та додана до середовища вище в коді.

Наступним кроком буде створення можливості отримувати шкоду противником, та додатково мати здоров’я, та можливість смерті. Для цього ми створимо новий фай, та назвемо його «EnemyHealth». Деякі функцію цього коду мають бути публічні, щоб була можливість викликати його, з інших функцій. Пропишемо ось такий код, який відображено нижче, в нашій роботі.

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

public class EnemyHealth : MonoBehaviour
{
    public int maxHealth = 100;  // Максимальне здоров'я ворога
    private int currentHealth;    // Поточне здоров'я ворога
    public Animator animator;
    Unity Message
    void Start()
    {
        currentHealth = maxHealth;  // Встановлення початкового здоров'я
    }

    public void TakeDamage(int damage)
    {
        currentHealth -= damage;  // Зменшення здоров'я на величину отриманого урону
        animator.SetTrigger("TakeHit");
        if (currentHealth <= 0)
        {
            Die();  // Виклик функції смерті, якщо здоров'я ворога вичерпано
        }
    }

    void Die()
    {
        // Логіка для обробки смерті ворога
        Destroy(gameObject);
    }
}
```

Рис. 3.40. Код

Цей код надає можливість, противником отримувати шкоду, програвати анімацію шкоди, померти, та встановлювати йому початкову кількість його здоров’я. Також цей код має можливість бути визваним з коду гравця, при

нанесені, гравцем, шкоди, нашому ворожому персонажі, якого ми зараз розробляємо.

На ступним та фінальним етапом написань, механік ворога має стати, нанесення шкоди гравцю від ворога, цей код має вже використовувати функцію яку ми прописали для нашого гравця раніше. Для цього коду створимо новий файл та назвемо його «EnemyAutoAttack», та пропишемо в ньому ось такий код.

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

public class EnemyAutoAttack : MonoBehaviour
{
    public int damage = 10;          // Кількість урону ворога
    public float attackRange = 2f;   // Дальність атаки ворога
    public LayerMask playerLayer;    // Шар гравця
    public Transform attackPoint;    // Точка атаки ворога
    public float timeBetweenAttacks = 2f;  // Інтервал між атаками
    public Animator animator;
    private float timeSinceLastAttack = 0f;

    Unity Message
    void Update()
    {
        // Запускаємо таймер між атаками
        timeSinceLastAttack += Time.deltaTime;

        // Перевіряємо, чи пройшов достатній час між атаками
        if (timeSinceLastAttack >= timeBetweenAttacks)
        {
            // Перевірка, чи гравець знаходиться у зоні атаки
            Collider2D player = Physics2D.OverlapCircle(attackPoint.position, attackRange, playerLayer);

            if (player != null)
            {
                // Викликаємо функцію атаки, якщо гравець виявлений
                Attack();
                timeSinceLastAttack = 0f;  // Скидаємо таймер атаки
            }
        }
    }

    void Attack()
```

Рис. 3.41. Код

```
    {
        Collider2D[] hitplayer = Physics2D.OverlapCircleAll(transform.position, attackRange, playerLayer);
        animator.SetTrigger("Hit");
        foreach (Collider2D player in hitplayer)
        {
            // Тут можна додати логіку зменшення здоров'я ворога, анімацію атаки тощо.
            //Debug.Log(enemy.name);
            player.GetComponent<PlayerHealth>().TakeDamage(damage);
        }

    }

    // Функція для візуалізації області атаки у редакторі
    Unity Message
    void OnDrawGizmosSelected()
    {
        if (attackPoint == null)
            return;

        // Відображення області атаки як кола у редакторі
        Gizmos.DrawWireSphere(attackPoint.position, attackRange);
    }
}
```

Рис. 3.42. Код

Цей код описує роботу, атаки противника по гравцю, при цьому він звертаються, до коду який був написаний раніше, та зазначений вище в коді, до цього.

Всі механіки ворога готові, та опрацьовані практично ворог тепер може йти на зближення та атакувати, також йому можна надати аналогічну відповідь, на його агресію.

Рис. 3.43. Ворог в ігровому просторі.

Додатково було вирішено додати ще декілька механік, для подальших рівнів, а саме механіка пасток,зони збреження , та зони повної смерті. Ці механіки нададуть можливість покращити дизайн сайту. Додатково з приводу дизайну було вирішено створити нову сітку, яка би використовувалась лише для дикору. Додатково було вирішено додати анімаційні тайли, такі як тайли конвеєру, вогню, та стуї вогню.

Ось так виглядає нова сітка для декору.

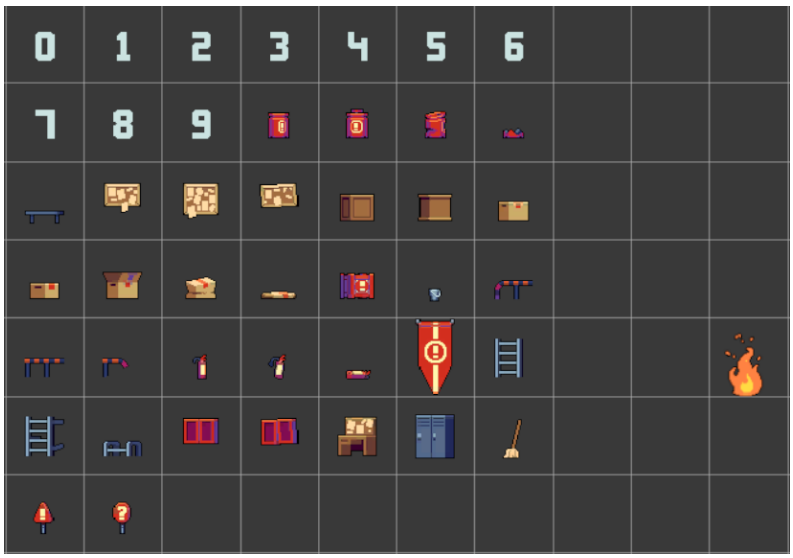


Рис. 3.44. Тали для декору

Наступним кроком, буде створення зони смерти, в цій зоні гравець має моментально помирати, та переміщатися або на початок рівня, або до зони збереження. Ця зона буде використовувати колайдер в якості тригеру і коли гравець зайде до нього одразу отримає сто відсотоків шкоди, від здоров'я. Для цього створимо нову сутність, дамо їй колайдер, та встановимо його як тригер. Цю сутність назвемо «DeatgZone», та помістимо її в «DeatgZones». Далі ми маємо прописати код, для цього потрібно створити новий файл, назвемо його «Triggerdeth». Ось такий код має бути в цьому файлі.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Triggerdeth : MonoBehaviour
6   {
7       // Start is called before the first frame update
        Unity Message
8       private void OnTriggerStay2D(Collider2D other)
9       {
10          if (other.name=="Player")  // Перевіряємо, чи це гравець
11          {
12              Rigidbody2D playerRb = other.GetComponent<Rigidbody2D>();
```

Рис. 3.45. Код

```
13
14              if (playerRb != null)
15              {
16                  // Застосовуємо силу пушу до гравця
17                  playerRb.GetComponent<PlayerHealth>().TakeDamage(100f);
18
19              }
20          }
21      }
22
23
24  }
25
```

Рис. 3.46. Код

Цей код описує моментальну смерть гравця при заходженні його до тригеру. Нижче візуалізована, зона смерті.

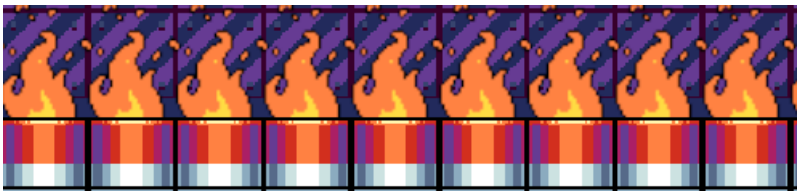


Рис. 3.47. Місце де може бути використана зона смерті.

Наступна механіка яку би хотів додати це – механіка пасток. Ідея полягає в тому, щоб створити пастку яка би вистрілювала, потоком полум’я в гравця, та наносила шкоду. Для цієї пастки створимо нову сутність та назвемо її «UPFS», в ній створимо ««UPF». В ««UPF» ми і будемо прописувати логіку, це буде тригером який буде з інтервалом, який ми задамо в розробці, буде стріляти полум’ям в зону. Для цього потрібно додатково додати анімацію вогню. Для анімації вогню створимо в юніті анімацію, та аніматор для цього колоайдеру, та додатково додамо умову активації цієї анімації. Анімація зображено нижче.

Рис. 3.48. Анімація полум’я

Далі потрібно прописати логіку полум’я, для цього нам вже потрібен код. Для нового коду створимо новий файл та назвемо його «TrigerFire». В даному файлі пропишемо код який би наносив шкоду гравцю, по заданому інтервалу, та напрямку.

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

public class TrigerFire : MonoBehaviour
{
    public float timeBetweenAttacks = 2f;  // Інтервал між атаками
    public Animator animator;
    private float timeSinceLastAttack = 0f;
    public int damage = 20;
    public LayerMask playerLayer;
    public Vector3 boxSize = new Vector2(1.0f, 1.0f);
    public Transform attackPoint;
    Unity Message
    void Update()
    {
        // Запускаємо таймер між атаками
        timeSinceLastAttack += Time.deltaTime;

        // Перевіряємо, чи пройшов достатній час між атаками
        if (timeSinceLastAttack >= timeBetweenAttacks)
        {

            // Перевірка, чи гравець знаходиться у зоні атаки
            Collider2D[] hitplayer = Physics2D.OverlapBoxAll(attackPoint.position,boxSize, playerLayer);

            animator.SetTrigger("Fire");
            foreach (Collider2D player in hitplayer)
            {

                if (player.GetComponent<PlayerHealth>() != null)
            {
                player.GetComponent<PlayerHealth>().TakeDamage(damage);
            }

            timeSinceLastAttack = 0f;  // Скидаємо таймер атаки
```

Рис. 3.49 Код

```
                timeSinceLastAttack = 0f;  // Скидаємо таймер атаки
            }

        }
    }
    Unity Message
    void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;

        // Отримання координат кутів квадрату
        Vector3 topLeft = attackPoint.position + new Vector3(-boxSize.x / 2,  boxSize.y / 2);
        Vector3 topRight = attackPoint.position + new Vector3(boxSize.x / 2,  boxSize.y / 2);
        Vector3 bottomLeft = attackPoint.position + new Vector3(-boxSize.x / 2,  -boxSize.y / 2);
        Vector3 bottomRight = attackPoint.position + new Vector3(boxSize.x / 2,  -boxSize.y / 2);
```

Рис. 3.50 Код

```
        // Намальовано кордони квадрату
        Gizmos.DrawLine(topLeft, topRight);
        Gizmos.DrawLine(topRight, bottomRight);
        Gizmos.DrawLine(bottomRight, bottomLeft);
        Gizmos.DrawLine(bottomLeft, topLeft);
    }
}

```

Рис. 3.51 Код

В цьому коді прописана логіка нанесення шкоди гравцю, коли він стоїть в зоні ураження в момент активації пастки. В коді також прописано активація анімації, та активації самої пастки. В грі це виглядає ось так.

Рис. 3.52 Вогняна пастка.

Тепер потрібно прописати механіки збереження гравця, або інакше зони схрону. Для цих зон створимо нову сутність та назвемо її «SaveZone», в цій сутності створимо сутність «Save», в цій сутності створимо колайдер, в якості тригеру та додамо візуал в вигляді коробки. Виглядає, зона збереження ось так

Рис. 3.53. Зона збереження.

Наступним кроком ми маємо прописати код для цієї механіки. Для коду ми маємо створити новий файл, та назвати його «Save».

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

public class Save : MonoBehaviour
{

    Unity Message
    void OnTriggerEnter2D(Collider2D other) {
        Debug.Log(other.name);
        other.GetComponent<PlayerHealth>().Save(transform.position.x, transform.position.y);
    }
}
```

Рис. 3.1.54 Код

Цей код описує зміну позиції, на яку телепортується гравець, коли помирає.

Наступним кроком, це створення механіки кінця гри, для цього створюємо сутність «EndGame», додаємо колайдер, в якості візуалу нехай будуть двері.

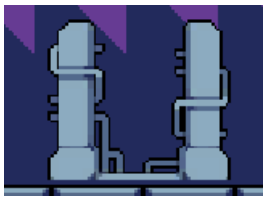


Рис. 3.55. Зображення кінця рівня

Для цього елементу також створимо код, для цього створюємо файл, «ENDGAME» , і пропишемо ось такий код.

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

public class ENDGAME : MonoBehaviour
{
    public int x = 0;
    Unity Message
    private void OnTriggerStay2D(Collider2D other)
    {

        SceneManager.LoadScene(x);
    }
}

```

Рис. 3.56 Код

Цей код відповідає за зміну сцен.

Додатково потрібно відобразити здоров’я головного героя на головному екрані. Для цього створимо сутні інтерфейсу. Та допишемо старий код здоров’я. Ось одразу готовий результат.

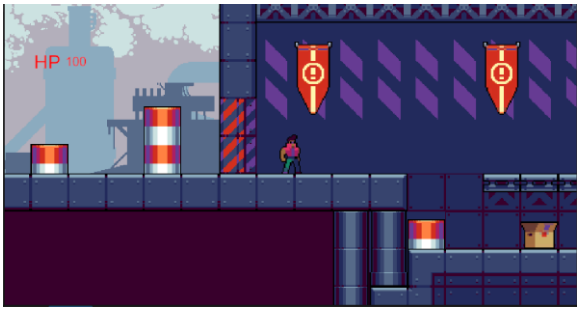


Рис. 3.57 Відображення здоров’я.

На разі ми вже маємо всі механіки для створення ігрового простору. Наступним етапом буде створення ігрових рівнів, використовуючи ті елементи які було про працьовано на етапі розробки механік, та на етапі підготовки.

3.2 Конструювання рівнів, та їх дизайн

Розробка рівнів для 2D платформера - це процес, що поєднує творчість та інженерію для створення привабливого та викликового ігрового досвіду. Це включає в себе створення інтересних та збалансованих геометричних конфігурацій, розташування перешкод та навколишнього середовища, які підтримують історію гри та прокладають шлях для гравця.При проектуванні рівнів для 2D платформера важливо:Створювати Різноманіття: Додавайте різноманіття в елементи рівнів, такі як перешкоди, платформи та вороги, щоб гравець не нудьгував та відчував виклик.Розташування Об'єктів: Розміщуйте об'єкти так, щоб створювали логічний та приємний для взаємодії шлях для гравця.Баланс Викликів та Винагород: Створюйте сценарії, де виклики та винагороди взаємодіють, створюючи динамічний геймплей.Забезпечувати Переходи: Додавайте елементи, що дозволяють гравцеві легко переходити між рівнями, створюючи враження єдності гри.Використовувати Графічні Ефекти: Застосовуйте графічні ефекти, які додають атмосферу грі та надають візуального задоволення.Навчання Гравця: Використовуйте перші рівні для навчання гравця основам гри та поступово ускладнюйте завдання.Сприяйте Експериментуванню: Залишайте простір для творчості гравця та його можливості експериментувати з власними стратегіями.Створюйте Відчуття Прогресу: Забезпечте відчуття досягнення та прогресу, надаючи гравцеві важливі цілі.

Окрім цього, важливо враховувати, що рівні повинні вписуватися в загальний контекст історії гри та підтримувати гравця у вивченні унікальних аспектів геймплею.

Для описання моїх рівнів я буду вже використовувати готовий результат, який створював самостійно, показувати загальну картину та описувати що на ній зображено.

Для початку хочу зазначити що було розроблено два рівні, та головне меню. Почнемо одразу з нього.

Рис. 3.58. Головне меню гри

В цьому меню є всього дві кнопки старту, та виходу з гри, до кожної кнопки був додатково прописаний код.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
6   public class SceneTransition : MonoBehaviour
7
8   {
9
10      public void Transition(){
11          SceneManager.LoadScene(1);
12      }
13  }
14
```

Рис. 3.59. Код для старту

Цей код починає першу сцену.

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

public class Exit : MonoBehaviour
{
    // Start is called before the first frame update
    public void Transition(){
        // Викликати функцію закриття гри
        #if UNITY_EDITOR
            UnityEditor.EditorApplication.isPlaying = false; /
        #else
            Application.Quit(); // Для виходу з гри на платфор
        #endif
    }
}
```

Рис. 3.60. Код для закінчення гри.

Цей код закінчує всі процеси в грі, та вимикає її.

В принципі на цьому функціонал першої сцени закінчився, її місія є чисто символічною, та слугує користувацьким інтерфейсом для початку, та закінчення гри. Додатково під час гри можна нажати Esc і відкриється це меню.

На цій сцені не потрібно зациклюватись, тому одразу краще перейдемо до соновних двух сцен, а саме до першої.

Перший рівень має бути, більш навчальним, оскільки це ознайомлення гравця з основними механіками які будуть зустрічатись в подальшому.

Я вирішив одразу розробити рівень, і далі по ходу роботи описати які механіки я для цього використовував. Нижче його візуалізація в двох скріншотах.

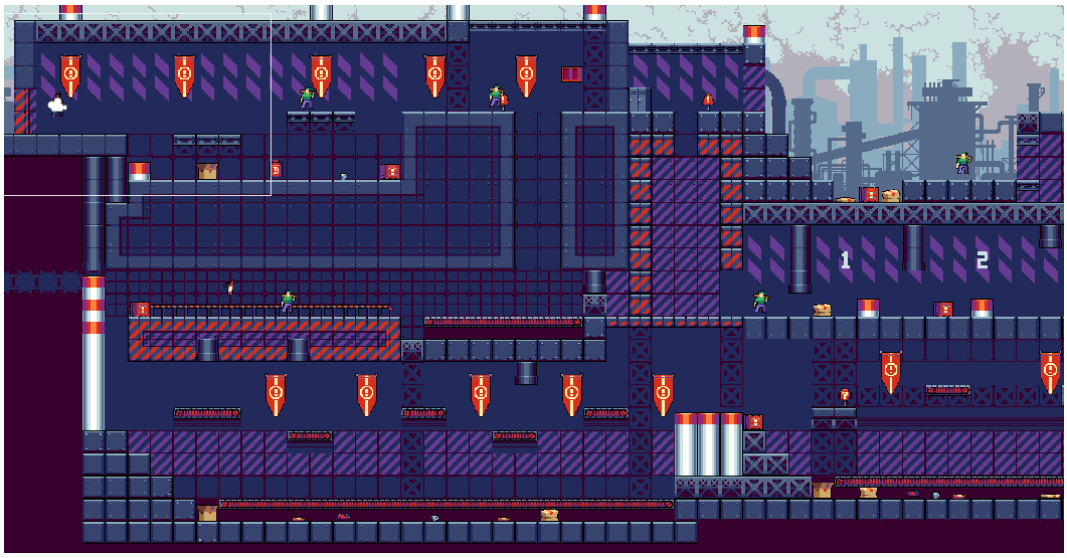


Рис. 3.61 Перший рівень частина перша

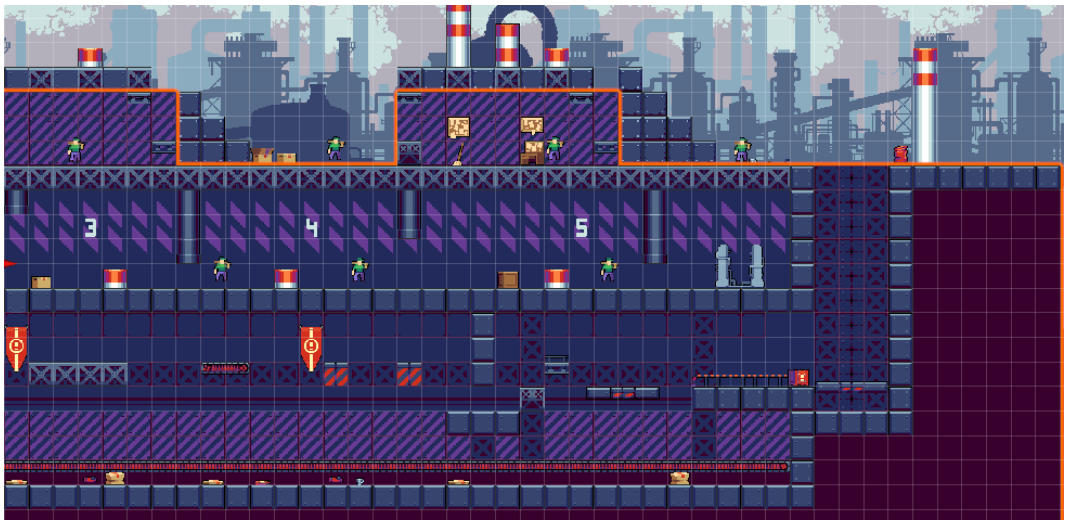


Рис. 3.62 Перший рівень частина друга

Як видно з зображення в самому рівні використовують майже всі механіки які було спроектовано підчас розробки основних механік. Не всі механіки були використані в цьому рівні, наприклад такі як, зона смерті, та пасток, але ключові вже використовуються.

Сам рівень було вирішено розбити на фрагменти для подальшого дослідження. Спочатку я опиши перший фрагмент, це фрагмент входу на рівень. Візуалізація фрагменту зображена нижче.

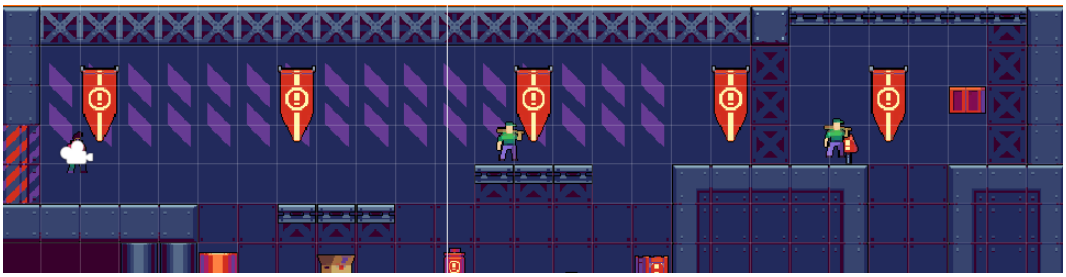


Рис. 3.63. Перший фрагмент, першого рівня

В цьому фрагменті використовуються, перешкоди в вигляді прірв між платформ, та для складнощів було додано декілька ворогів, якщо гравець не зможе здолати перешкоди, то він може піднятися і спробувати знову. В самому фрагменті додатков використовуються елементи декору. Взагалі першим етапом побудови рівня було нанесення ландшафту, потім заднього фону, далі декор, після механічні частини рівня і в кінці додавались вороги. Перейдемо до наступного фрагменту.

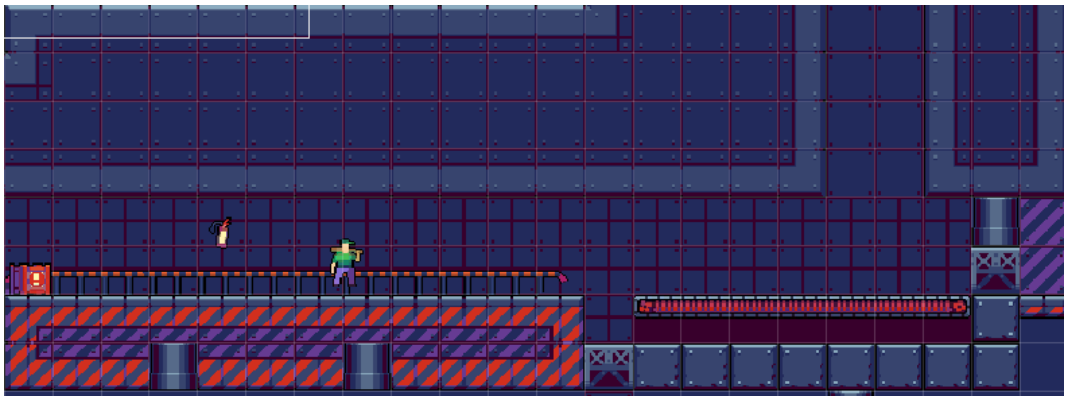


Рис. 3.64 Другий фрагмент, першого рівня

Після того як гравець пройде перший фрагмент, гравець має стрибнути в прірву, та потрапити на другий фрагмент рівня. В цьому фрагменті гравець вперше познайомиться з конвеєром, та з зоною збереження, цей фрагмент є проміж уточним для переходу на інший фрагмент. В цьому фрагменті

використовується невелика кількість декору, але був доданий ворог, для того щоб гравець не втрачав пильність. Переходимо до третього фрагменту.

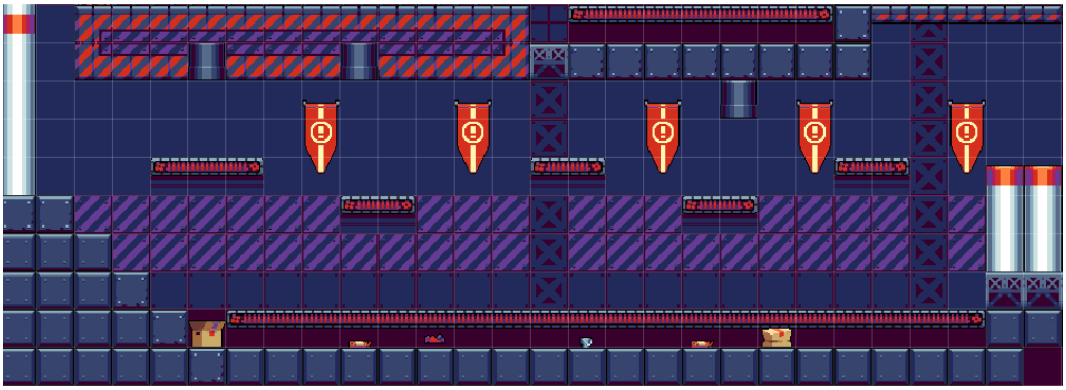


Рис. 3.65 Третій фрагмент, першого рівня

В цьому фрагменті гравець зустрічається з першим випробуванням на спритність. На цьому фрагменті використовується велика кількість конвеєрів з яких потрібно перестрибувати, з одного на інший. Самі конвеєри мають різний напрямок руху, та швидкість, що ускладнює гравцю проходження цього фрагменту, але оскільки це перший рівень в разі невдачі гравець може спробувати знову. Перейдемо до наступного фрагменту.

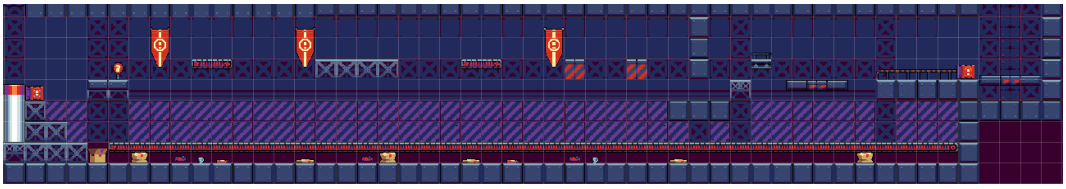


Рис. 3.66 Четвертий фрагмент, першого рівня

Минулий фрагмент можна вважати розминкою перед наступним. В цьому фрагменті гравець вперше зустрічається з платформою, гравцю критично важливо дочекатися платформи, та разом з нею, по верхній частині фрагменту пробігти до його кінця. Найгіршим є те що при помилці гравця йому потрібно чекати, поки платформа доїде до кінця, потім поїде назад для наступної спроби. Вкінці на гравця чекає ліфт на гору. Настала черга п’ятого фрагменту.

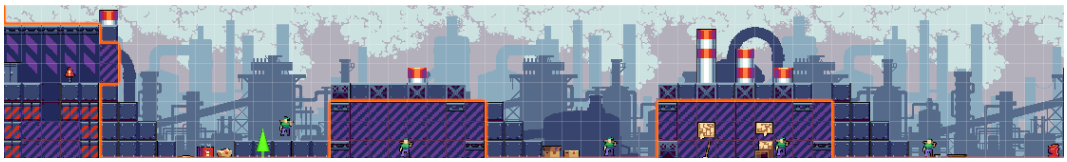


Рис. 3.67. П’ятий фрагмент, першого рівня

Цей фрагмент немає великої кількості паркурних елементів, особливість цієї частини це вихід на поверхню, та велика кількість противників. Фрагмент можна пробігти як і по верхній частині, щоб не пересікатися з ворогами, так і по нижній, вбиваючи всіх ворогів. В кінці є перехід на останній фрагмент в вигляді ями. Перейдемо до фінального фрагменту цього рівняю

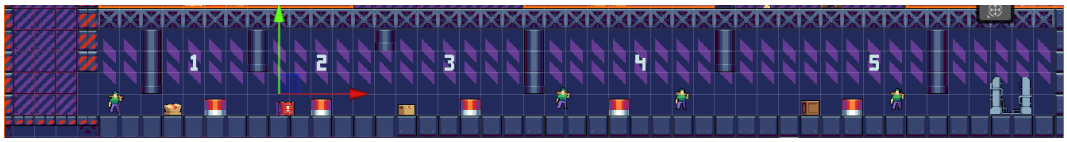


Рис. 3.68 Фінальний фрагмент, першого рівня

Це є фінальним фрагментом цього рівня, він поєднує в собі невелику кількість паркуру та, бойових механік, ворогів можна або вбити, або пробігти. На кінці фрагменту гравця очікує портал для переходу на наступний рівень.

На цьому перший рівень закінчується, сам рівень використовує елементи паркуру та бойовки. Механіки які не були відображені в цьому рівні було вирішено не додавати, тому що це перший рівень, та можуть викликати навантаження на гравця. В цілому в даному рівні було використано: десять конвеєрів, дві платформи, п’ять зон збереження, дванадцять ворогів. Сам рівень не має викликати складнощів у будь якого гравця, навіть у тих хто раніше з іграми не знайомився.

Після огляду першого рівня можна перейти до огляду наступного він також має вже готовий результат, єдине що потрібно це розібрати його, сам рівень я також вирішив поділити на фрагменти, для більш легкого дослідження. Нижче буде описано кожний фрагмент, та його візуалізація але для початку візуалізація всього рівня нижче.

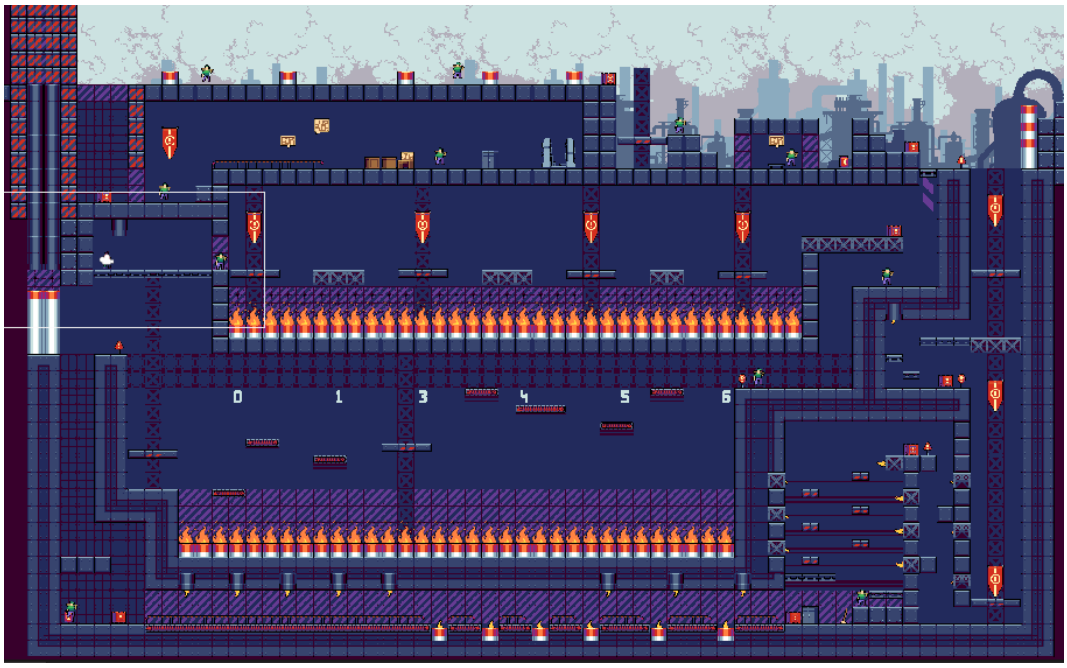


Рис. 3.69 Другий рівень

Це є другим рівень і останнім рівнем нашої гри, як можна побачити він використовує фрагменти які були описані в тексті, але в минулому рівні не використовувались. Після першого рівня гравець, мав більш менш зрозуміти логіку гри тому можна перейти до системи покарання, в разі невдачі. Можна перейти до першого фрагменту.

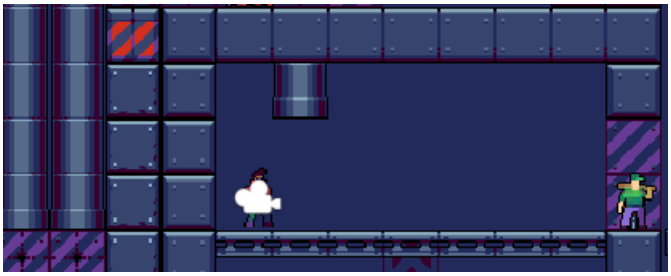


Рис. 3.70 Перший фрагмент другого рівня

Перший фрагмент це вітальна кімната, площадка старту в якій знаходиться один ворог, та мінімум декору. Можна одразу перейти до наступного фрагменту, який більш складним випробуванням, для не підготовленого гравця.

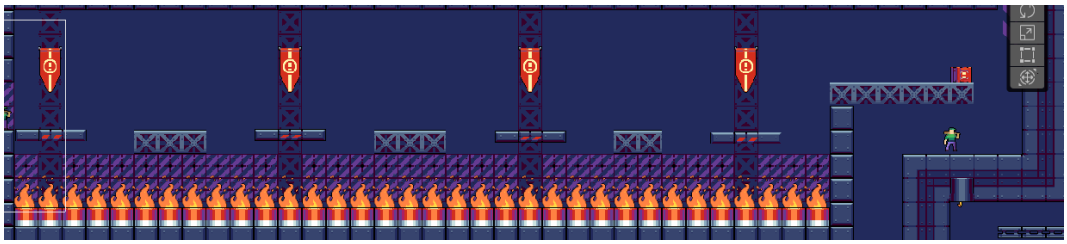


Рис. 3.71. Другий фрагмент другого рівня

Другий фрагмент має паркурну складову, в якій гравець не має право на помилку, сам рівень використовує платформи, та зону смерті, яка візуалізована вогняним стовпом, в кінці рівня на гравця чекає збереження та перехід на інший фрагмент. Огляд наступного фрагмента відображенно нижче в тексті.

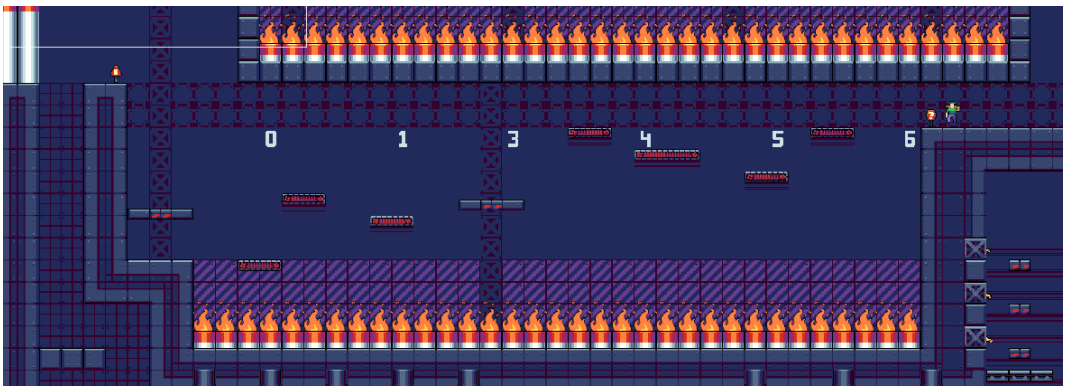


Рис. 3.72. Третій фрагмент другого рівня

Третій фрагмент є більш складним ніж минулий він поєднує в собі використання платформ та конвеєрів. В кінці гравець має втриматись на конвеєрі та встигнути за стрибнути на ліфт.

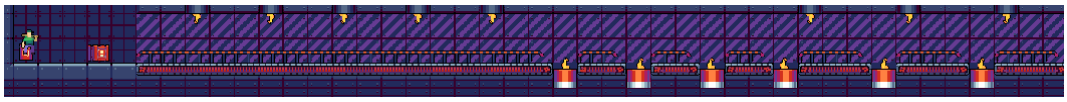


Рис. 3.73. Четвертий фрагмент другого рівня

Цей фрагмент містить у собі перше використання вогневих пасток, та поєднує це використанням конвеєрів, перший конвеєр є дуже швидким, наступні більш повільні, але мають різні вектори руху. В кінці зона збереження та ворог. Наступний фрагмент.

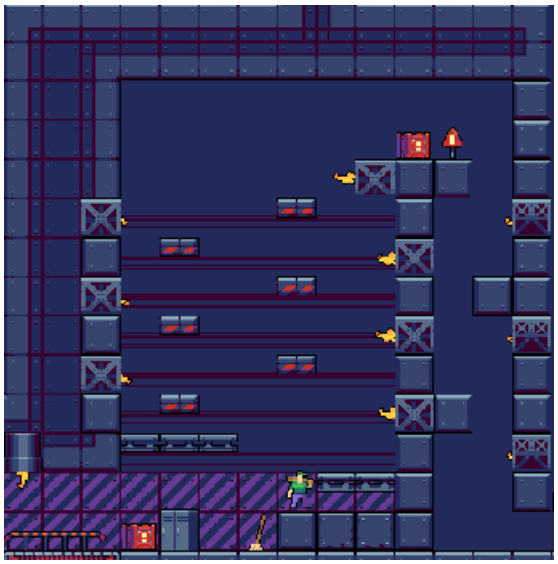


Рис. 3.74 П’ятий фрагмент другого рівня

В цьому фрагменту використовується велика кількість паркуру, та пасток, гравець може мати невеликі складнощ і при проходженні цього фрагменту. Перейдемо до наступного фрагменту.

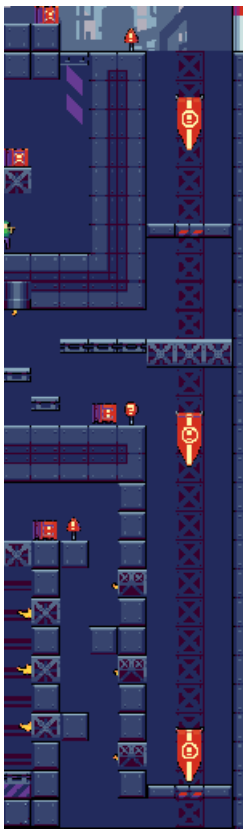


Рис. 3.75. П’ятий фрагмент другого рівня

Цей фрагмент є більш переходом між фрагментами, але має все ж таки деякі технічні складнощі в проходженні. Перейдемо до наступного фрагменту.

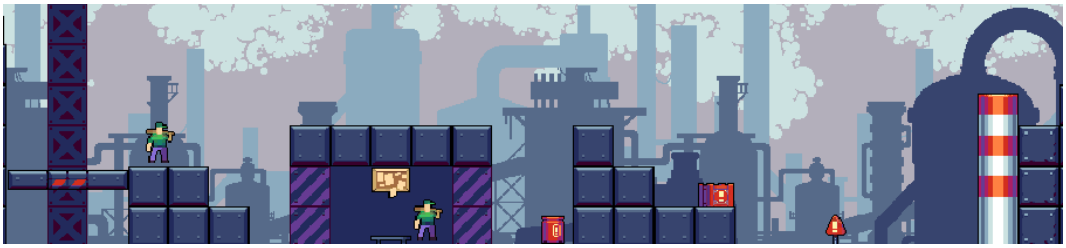


Рис. 3.76. Шостий фрагмент другого рівня

Цей фрагмент є вулицею він має невелику кількість ворогів, та ліфт на кінці рівня, цей фрагмент, не має викликати складнощів в гравця, який його буде проходити. Можна перейти до останього фрагменту нашого рівня, та підбити підсумки.

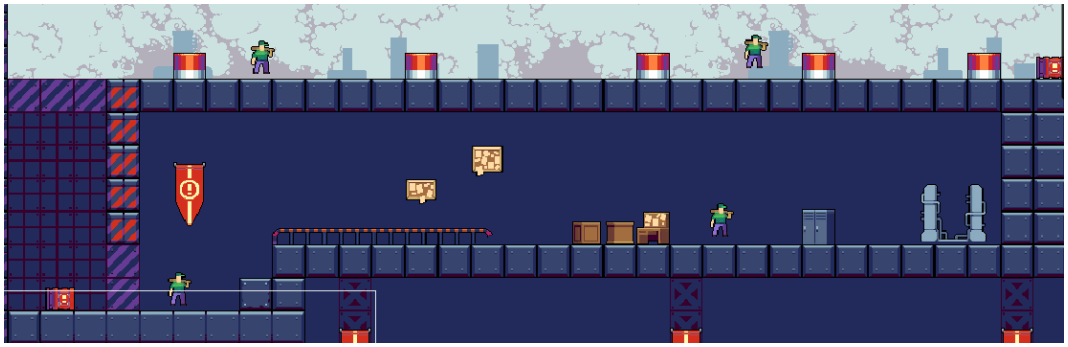


Рис. 3.77. Шостий фрагмент другого рівня

Це останній фрагмент цього рівня він складається з даху та, низу на кінці портал до меню.

В цілому другий рівень закінчено, в ньому використовувались всі механіки які були розроблені під час гри.

На цьому розробку рівнів можна закінчувати та, перейти до фінальної компіляції проекту та, завантаження на https://github.com/.

Настав час компіляції самого додатку, для цього в самому юніті відкриємо в панелі меню вкладку File>Build Settings, далі оперемо платформи віндовс, лінкос,

мак і натиснемо Build. Нам запропонують створити папку ствримо папку «ДипломБілд» в неї скомпілюємо проект.

Все на разі маємо готовий додаток який вже запускається, та може бути перенесеним на інший пристрій

Додатково створимо github репозиторій, посилання на мою роботу «https://github.com/dimagumballl/DiplomKovalenkoDmytro», завантажуйте та грайте наздоров’я, в цілому я дуже задоволений результатом. Нажаль не всі початкові задумки вдалося реалізувати, в самій грі немає звуку, рівнів мало бути хоча би п’ять і фінальний з босом, але і так добре. Дякую вам за те що прочитали мою роботу.

ВИСНОВКИ

В дані магістрській роботі було розроблено гру на основі Unity в жанрі платформер

В даній магістрській роботі розроблено та виконано наступні завдання

- Проведено початкове планування та, загальний аналіз початкових завдань.
- Був проведений пошук та аналіз основної тематики гри яку ми будемо розробляти.
- Був проведений загальний аналіз технічних можливостей Unity.
- Був проведений повний збір ресурсі які, були використанні в розробці, та проведено підготовку з приводу налаштування середовища Unity.
- Було створено основний набір механік до гри.
- Було розроблено набір рівнів який, поєднував собі набір всіх механік, та додатково було проведено компіляцію додатку, та завантаження його на github.com.

В початковому плануванні була розроблено загальний план робіт який потім ми використовували, для розбиття всієї роботи на фрагменти.

При пошуку тематики, було знайдено основний стиль та жанр гри, яки потім було реалізовано в наступних пунктах

При аналізі технічних можливостей Unity був розглянутий основний перелік інструментів Unity, який був використаний вроботі.

При зборі ресурсів, були зібрані основні джерела для, спрайтів, анімацій і додатково була підготовлене середовище розробки Unity.

При розробці механік, було написана основна логіка ігрових процесів, та взаємодія між ними, та ініціалізація гри.

При розробці рівнів було реалізовано поєднання всіх механік які були розроблені до цього.

В кінцевому підсумку було розроблено гру від самого планування, до її ініціалізації. Було отримано досвід в базовій розробці ігор, жанру платформ ер. В подальшому можна використовувати цей досвід для, реалізації більш складних проектів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Документація Unity – https://docs.unity3d.com/ru/530/Manual/PresetLibraries.html

2.Візуальні ресурси для гри – https://craftpix.net/

3.Допоміжні ресурси для розробки – https://habr.com/ru/articles/236755/

4.Unity developer tools – https://unity.com/developer-tools

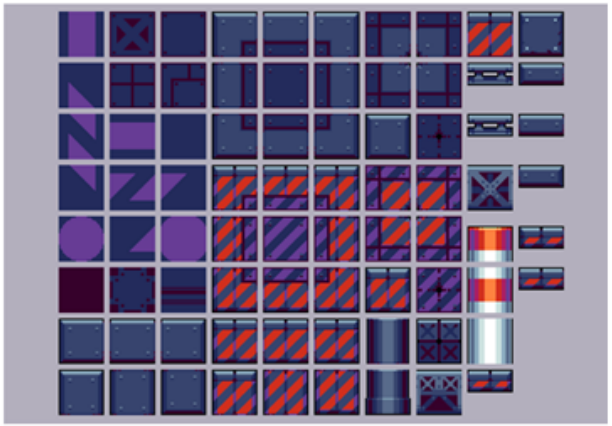
5.UNITY ENGINE – https://unity.com/products/unity-engine

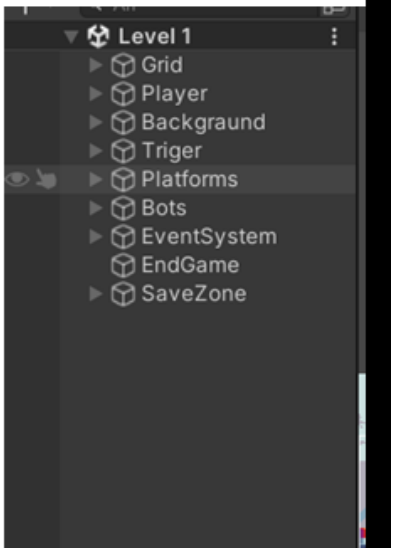
6. Tilemap component reference – https://docs.unity3d.com/Manual/class-Tilemap.html

7. Animation – https://docs.unity3d.com/Manual/AnimationSection.html

ПЛАНУВАННЯ РІВНІВ

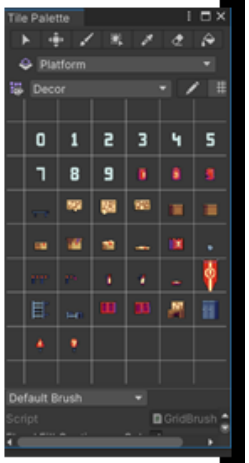
Під час планування відбувається аналіз технічних можливостей, підбір основної тематики, та планування самого проекту.

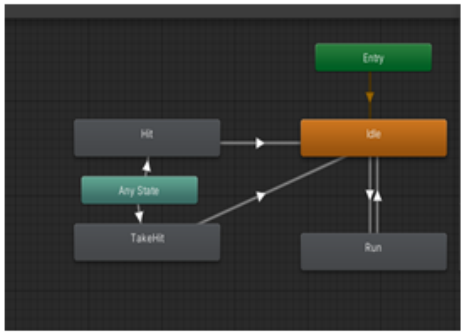


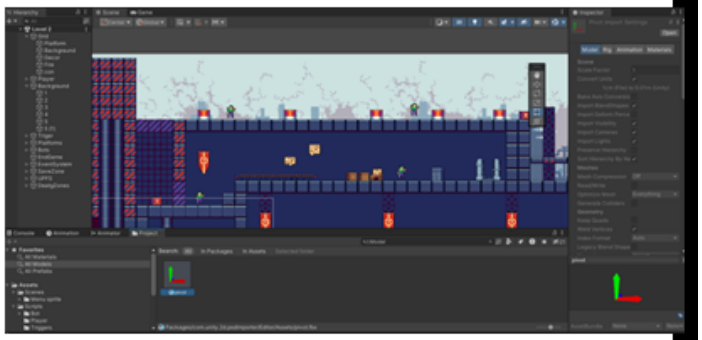


РОЗГЛЯД ІНСТРУМЕНТАРІЮ

Unity має великий набір інструментів, які можна використовувати, для реалізації завдань, які виникають під час створення технічного завдання.

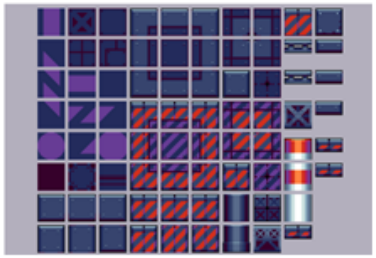






ПІДГОТОВЧА ФАЗА

Під час підготовчої фаза проводиться збір ресурсів,
та інтеграція їх в середовище Unity.



СТВОРЕННЯ МЕХАНІК

Під час реалізації технічного завдання, проводиться основна робота по створенню основних компонентів гри, які надалі будуть використані в конструюванні самої гри.

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



public class PushTrigger : MonoBehaviour
{
    public float way = -1f;  // Сила пуху

    // Викликається, коли інший об'єкт входить в тригер
    Unity Message
    private void OnTriggerStay2D(Collider2D other)
    {

        if (other.name=="Player")  // Перевіряємо, чи це гравець
        {
            Rigidbody2D playerRb = other.GetComponent<Rigidbody2D>();

            if (playerRb != null)
            {
                // Застосовуємо силу пуху до гравця
                playerRb.transform.position += new Vector3(5f*0.5f*0.01f*way, 0, 0);

            }
        }
    }
}
```

КОНСТРУЮВАННЯ РІВНІВ

При конструюванні рівнів, відбувається поєднання всіх компонентів, та ресурсів, які були підготовлені, або створені до цього.