

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ЗАКЛАД
„ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА”

Навчально-науковий інститут математики та інформаційних технологій


Свистунов Михайло Геннадійович

**РОЗРОБКА ДОДАТКУ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ
GODOT**

кваліфікаційна робота

**здобувача вищої освіти першого (бакалаврського) рівня
освітньої програми «Інженерія програмного забезпечення»
за спеціальністю 121 Інженерія програмного забезпечення**

Особистий підпис _____ Михайло СВИСТУНОВ

Науковий керівник _____  Ольга СМАГІНА,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Полтава – 2024

Міністерство освіти і науки України
Державний заклад „Луганський національний університет
імені Тараса Шевченка”

Факультет (інститут)

Навчально-науковий інститут математики
та інформаційних технологій

Кафедра

Інформаційних технологій та систем

Рівень освіти

перший (бакалаврський)

Спеціальність

121 «Інженерія програмного
забезпечення»

(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТС

Микола СЕМЕНОВ

(підпис)

(ім'я, прізвище)

“ ”

2024 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Свистунова Михайла Геннадійовича

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка додатку з використанням ігрового рушія Godot

Керівник кваліфікаційної роботи

Смагіна О.О.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджена наказом по університету

Від“ ” 2024 року №

2. Строк подання студентом проекту (роботи)

3. Вихідні дані до роботи (проекту) у результаті виконання роботи

повинно бути розроблено двомірну гру для ПК на рушії Godot Engine

(визначаються кількісні або (та) якісні показники, яким повинен відповідати об'єкт розробки)

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.

ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

ПРОЕКТУВАННЯ ТА КОНСТРУЮВАННЯ ГРИ.

(визначаються назви розділів або (та) перелік питань, які повинні увійти до тексту ПЗ)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проекту (роботи)

| Розділ | Прізвище, ініціали та посада Консультанта | Підпис, дата | |
|--------|--|----------------|------------------|
| | | завдання видав | завдання прийняв |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання „_____” _____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН


| № з / п | Назва етапів дипломного проекту (роботи) | Строк виконання етапів проекту (роботи) | Примітка |
|------------------|--|--|----------|
| | Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника. | До 1 березня | |
| | Аналіз літературних джерел за темою роботи. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи та плану експериментальних досліджень. | До 20 березня | |
| | Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання науковим керівником. | До 1 квітня | |
| | Усунення зауважень, урахування рекомендацій наукового керівника. Подання теоретичної частини роботи на друге читання. | До 15 квітня | |
| | Проведення експериментальної роботи. Поетапний аналіз та обговорення її результатів. Перевірка стану виконання роботи. | До 30 квітня | |
| | Урахування рекомендацій наукового керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Розробка презентації. | До 30 травня | |
| | Попередній захист роботи на кафедрі | травень | |
| | Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії | За 10 днів до державної атестації | |
| | Подання на кафедру остаточного варіанта роботи, переплетеного та підписаного автором, науковим керівником і рецензентом. | За 5 днів до державної атестації | |

Студент

М.Г. Свистунов

Керівник проекту (роботи)

підпис



підпис

О.О. Смагіна

АНОТАЦІЯ

Свистунов.М.Г

Тема: Розробка додатку з використанням ігрового рушія Godot

Спеціальність: 121 "Інженерія програмного забезпечення"

Установа: ЛНУ імені Тараса Шевченка, 2024 р.

Об'єкт дослідження: ігри на ПК, що являються основоположниками ігрової індустрії

Предмет дослідження: створення з врахуванням постійних вимог споживачів до нововведень, реалістичної гри, розробленої для операційної системи Windows

Мета роботи - розробка додатку з використанням ігрового рушія Godot

Результати роботи. В роботі проведено аналіз статистичних даних, пов'язаних з розробкою комп'ютерних ігор, а також виконано проектування і розробка двомірної гри на платформу Windows.

Ключові слова. Godot, ігровий рушій, гра, розробка ігор, ігровий додаток, Godot Engine, 2D/3D графіка, GDScript, інтерфейс, фізика в іграх, анімація, кросплатформовість, дизайн рівнів.

ABSTRACT

Svystunov. M.G

Topic: Application development using the Godot game engine

Specialty: 121 "Software engineering"

Institution: DZ Luhansk Taras Shevchenko National University, 2024

The object of research: PC games, which are the founders of the gaming industry

The subject of the study: creation, taking into account the constant demands of consumers for innovations, a realistic game developed for the Windows operating system

The purpose of the work is to develop an application using the Godot game engine

Work results. The work includes an analysis of statistical data related to the development of computer games, as well as the design and development of a two-dimensional game for the Windows platform.

Keywords. Godot, game engine, game, game development, game application, Godot Engine, 2D/3D graphics, GDScript, interface, game physics, animation, cross-platform, level design.

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»
Факультет (інститут) Навчально-науковий інститут математики та
інформаційних технологій

(повна назва)
Кафедра Інформаційних технологій та систем

(повна назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТС

М.А. Семенов

(підпис)

(ініціали, прізвище)

“ ” 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання програмної розробки (ПР):

РОЗРОБКА ДОДАТКУ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ
GODOT

ПОГОДЖЕНО
Керівник кваліфікаційної роботи

О.О. Смагіна
“ ” 2024р

ВИКОНАВЕЦЬ
Студент групи 4ПЗ

Свистунов М,Г
“ ” 2024р

ЗМІСТ

| | |
|---|---|
| ВСТУП | 3 |
| 1. Підстави для розробки програми..... | 3 |
| 2. Призначення розробки..... | 3 |
| 3. Вимоги до програми | 3 |
| 3.8. Спеціальні вимоги: | 5 |
| 4. Вимоги до програмної документації..... | 5 |
| 5. Етапи виконання розробки..... | 5 |
| 6. Порядок контролю і прийому | 5 |
| 7. Порядок внесення змін до технічного завдання, що затверджено. | 5 |

ВСТУП

Найменування програми: Додаток "платформер" з використанням ігрового рушія Godot

Область застосування: Програма призначена для використання на персональних комп'ютерах.

1. Підстави для розробки програми

1.1. Перелік документів, на підставі яких ведеться розробка:

- Технічне завдання
- Пояснювальна записка
- Програма

1.2. Організація: ЛНУ імені Тараса Шевченка.

1.3. Терміни розробки:

Початок 30 жовтня 2023 р.

Закінчення 01 травня 2024р

1.4. Умовне позначення: к-с додаток " платформер "

2. Призначення розробки

2.1. Функціональне призначення: Здатність гравців керувати персонажем на двомірному ігровому полі, взаємодіяти з ігровими об'єктами та іншими персонажами.

2.2. Експлуатаційне призначення: Програма повинна експлуатуватися на персональних комп'ютерах користувачів.

3. Вимоги до програми

3.1. Вимоги до функціональних характеристик:

3.1.1. Користувач програми повинен мати змогу виконувати наступні функції:

- Можливість керувати персонажем на двомірному ігровому полі.
- Можливість взаємодіяти з різними ігровими об'єктами та елементами.
- Можливість проходити рівні.

- Можливість взаємодії з ігровим середовищем.

3.2 Вимоги до надійності:

3.2.1. Програма повинна працювати без аварійних завершень або зависань протягом усього часу ігрового процесу.

3.2.2. Всі дані, що вводяться гравцем, повинні коректно оброблятися ігровою системою без втрати інформації.

3.2.3. Гра повинна пройти ретельне тестування перед випуском, щоб виявити та виправити всі можливі помилки

3.3. Умови експлуатації:

3.3.1. Кінцевий користувач використовує персональний комп'ютер, на якому додаток повинен надійно працювати.

3.4. Вимоги до складу та параметрів технічних засобів:

3.4.1. Процесор з тактовою частотою 1.2 ГГц і більше.

3.4.2. Відеокарта із 128 Мб відеопам'яті і вище.

3.4.3. 256 Мб оперативної пам'яті і вище.

3.4.4. 100 Мб вільного місця на накопичувачі.

3.4.5. Дісплей із розширенням 1366x768 і вище.

3.4.6. Клавіатура, комп'ютерна миша/тачпад.

3.5. Вимоги до інформаційної і програмної сумісності:

4.5.1. Операційна система Windows XP/Vista/7/8/8.1/10.

3.6. Вимоги до маркування та упаковки:

3.6.1. Програма поширюється може поширюватися за допомогою флеш-картки.

3.6.2. Вимоги до маркування не пред'являються.

3.7. Вимоги до транспортування і зберігання:

4.7.1. Поширення програми можливе через мережу інтернет, на зовнішніх накопичувачах тощо – на розсуд замовника.

3.7.2. Особливості зберігання відповідні до особливостей зберігання накопичувачів, на яких розміщена програма.

3.8. Спеціальні вимоги:

3.8.1. Мова програмування – GD Script

4.Вимоги до програмної документації

Перелік документів, що йдуть у комплекті з програмою:

4.1. Технічне завдання

4.2. Пояснювальна записка

5. Етапи виконання розробки

Етапи виконання можуть уточнюватись згідно календарного плану робіт по узгодженню між замовником та виконавцем.

6. Порядок контролю і прийому

6.1. Представлення дипломної роботи до попереднього захисту

6.2. Представлення дипломної роботи до захисту

7. Порядок внесення змін до технічного завдання, що затверджено.

Дане технічне завдання може уточнюватися в процесі розробки ПР при узгодженні сторін з оформленням доповнень до ТЗ.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ЗАКЛАД «ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА»

Факультет (інститут)

Навчально-науковий інститут математики та
інформаційних технологій

(повна назва)

Кафедра

Кафедра інформаційних технологій та систем

(повна назва)

Пояснювальна записка
до кваліфікаційної роботи
за першим (бакалаврським) рівнем освіти

Розробка додатку з використанням ігрового рушія godot

Виконав: студент 4 курсу
напряму підготовки (спеціальності)
121 «Інженерія програмного
забезпечення»

(шифр і назва напряму підготовки, спеціальності)

Свистунов М.Г.

(прізвище та ініціали)

Керівник

Смагіна О.О.

(прізвище та ініціали)

Рецензент

Козуб Ю.Г.

(прізвище та ініціали)

Полтава – 2024 року

ЗМІСТ

| | |
|--|----|
| ВСТУП | 3 |
| РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД СТВОРЕННЯ ВІДЕОІГОР | 4 |
| Актуальність відеоігор | 4 |
| Процес створення відеоігор..... | 5 |
| Аналіз актуальних ігрових рушіїв..... | 8 |
| Висновки до розділу 1 | 15 |
| РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА ДИЗАЙН..... | 16 |
| Графіка | 16 |
| Аналіз існуючих розробок..... | 20 |
| Створення концепції гри | 24 |
| Висновки до розділу 2 | 28 |
| РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ | 29 |
| Опис інтерфейсу ігрового рушія | 29 |
| Створення проекту | 37 |
| Додавання заднього фону..... | 40 |
| Створення рівнів..... | 44 |
| Створення персонажу | 47 |
| Додавання анімацій..... | 51 |
| Технічні характеристики | 56 |
| ВИСНОВКИ..... | 57 |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... | 58 |
| ДОДАТОК А..... | 59 |
| ДОДАТОК Б | 66 |

ВСТУП

Актуальність дослідження: Важливість дослідження обраної теми полягає як у її слабкій вивченості, і у колосальних темпах розвитку об'єкта вивчення. Комп'ютерні ігри стали невід'ємною частиною сучасної культури, та життя багатьох людей. Ігри мають великий культурний, соціальний та економічний вплив на суспільство та технології. У ряді країн ігри офіційно визнаються частиною мистецтва і навіть виступають як кіберспортивні дисципліни. Відеоігри це невід'ємна частина життя сучасної людини та найуспішніша медіа, що перевершила навіть кінематограф.

Мета за завдання роботи: метою дипломної роботи є дослідження процесу розробки відеоігор, та історію їх виникнення. Проаналізувати жанри відеоігор та середовищ які використовуються для розробки відеоігор а також розробити програмний продукт у вигляді відеогри з двовимірною графікою у жанрі платформер з можливістю її подальшої реалізації на сервісах продажу відеоігор

Об'єкт дослідження: Об'єктом дослідження є відеогра як програмний продукт а також аналіз методів і технологій, використовуваних у процесі створення відеоігор, включаючи програмування, дизайн, та тестування.

Методи дослідження: аналіз наукової літератури, узагальнення та систематизація теоретичних положень про ігровий рушій Godot та мову програмування GD Script.

РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД СТВОРЕННЯ ВІДЕОІГОР

Комп'ютерна гра - це вид розважального або навчального програмного забезпечення, яке використовується для відтворення віртуальних інтерактивних середовищ або сценаріїв.

Гравець може контролювати персонажів або об'єкти в грі за допомогою контролерів, таких як клавіатура, миша, геймпад або сенсорний екран. Відеоігри можуть бути розроблені для різних платформ, таких як персональні комп'ютери, ігрові консолі, мобільні телефони, планшети тощо. Вони можуть мати різні жанри, від стратегічних та рольових ігор до шутерів, головоломок, ігор з відкритим світом та інших.

Актуальність відеоігор

Відеоігри стали важливим аспектом розважальної індустрії, яка генерує значні прибутки та має великий вплив на культуру. Вони пропонують спосіб відпочинку для людей різного віку та інтересів. Деякі відеоігри пропонують мультиплеєрний режим, який сприяє соціальній взаємодії та співробітництву між гравцями з усього світу. Також відеоігри можуть бути використані для навчання та розвитку різних навичок, таких як стратегічне мислення, співпраця, та проблемне вирішення.

Галузь відеоігор стала значним гравцем на світовому ринку, з численними компаніями, які вкладаються в розробку і рекламу гри, а також в організацію глобальних турнірів та ігрових подій. Усі ці фактори свідчать про те, що відеоігри залишаються актуальними і продовжують розвиватися як важлива складова сучасної культури та розваг.

Процесс створення відеоігор

Загальний алгоритм розробки комп'ютерної гри мало чим відрізняється від алгоритму розробки будь-якого іншого програмного продукту та

включає в себе 3 великі етапи:

- 1) проектування;
- 2) розробка;
- 3) видання та підтримка.

На етапі проектування визначаються мета гри та засоби її розробки. При визначенні мети виділяються ідея, жанр та сеттинг гри.

Ідея це те, що буде спонукати гравця грати в гру, що створюється, і вона дуже тісно пов'язана із жанром. Так, наприклад, основна ідея RPG – дозволити гравцеві прожити свою роль так, як захочеться, а основна ідея шутера – дозволити гравцеві взяти участь у реальних чи вигаданих бойових діях. Таким чином, визначивши основні ідеї гри, жанр буде підібраний. майже відразу.

Визначившись із жанром та ідеєю гри, наступним кроком буде вибір сеттингу. Сеттинг – це середовище, в якому відбуватиметься основна дія гри. Він визначає місце, час та умови дії. Вибір сеттингу може сильно полегшити розробку сценарію для гри, тому його краще вибирати заздалегідь, ґрунтуючись на смаках цільової аудиторії.

До засобів розробки в першу чергу відносять програмний код та ігровий рушій. Від їхнього грамотного вибору залежить як швидкість самої розробки, так і працездатність продукту. Програмний код в першу чергу залежить від платформи, для якої створюватиметься комп'ютерна гра. Наприклад, якщо гра створюється для браузерів, то логічно буде використання мови Java або Flash, але якщо гра створюється для персонального комп'ютера, оптимальним вибором буде, наприклад, мова програмування C#.

Ігровий рушій відповідає за низькорівневий опис фізики об'єктів, правил рендерингу графіки та ін. При виборі ігрового рушія першим чином дивляться на його доступність і вже зроблений вибір мови програмування. Наприклад, ігровий рушій Unity дозволяє розробляти ігри мовою C# і він є безкоштовним, якщо середній дохід компанії не перевищує \$100000 на рік.

Після вибору мети гри та засобів розробки, починається другий етап реалізації проекту – розробка. Розробка найбільший і найдовший етап реалізації проекту, він включає велику кількість кроків, без яких неможливо створити працездатний продукт. Насамперед потрібно визначитися із сюжетом та ігровою механікою. Ігрова механіка ґрунтується на цілі гри, вона визначає всі об'єкти та правила, за якими гравець взаємодіятиме з ними. Зазвичай паралельно із розробкою ігрової механіки йде написання сюжету гри. Сюжет відіграє не маловажну роль, він визначає те, наскільки гравцеві буде цікаво грати у вашу гру. Сюжет представляють у двох варіантах: літературний та режисерський сценарії. Літературний сценарій описує основні події та персонажів гри, які беруть участь у грі. Режисерський же є докладним описом рівнів гри, подій, які на цих рівнях відбуваються. Так само на даному етапі починається раннє опрацювання графічної складової та дизайну гри. На основі сюжету та заздалегідь обговореного дизайну, створюються ранні концепт-арти, на основі яких згодом буде опрацьовано основний вид гри та персонажів.

Після розробки сюжету та ігрової механіки починається найважливіша частина – розробка самої гри.

На основі сюжету та концепт-артів починається створення персонажів та об'єктів гри, паралельно із цим йде розробка ігрових рівнів. При розробці ігрових рівнів спочатку створюється його спрощений план, на якому схематично зображено сам рівень, а також предмети, з якими згодом взаємодіятиме гравець. Після цього створюється перша версія рівня. Зазвичай, вона є просто голою локацією, з мінімумом необхідних для проходження предметів. Ця версія рівня служить для того, щоб протестувати рівень на

прохідність. Після тесту рівень починають поступово заповнювати рештою об'єктів. Незабаром після створення перших рівнів відбувається складання першого прототипу гри, яку називають альфа-версією гри. Вона необхідна для того, щоб розробник міг провести тестування (альфа-тестування) основних механіки гри, та перевірити наскільки вона відповідає заявленим вимогам. Часто в альфа-версії гри, об'єкти навіть не мають текстур або вони взагалі представлені у вигляді абстрактних об'єктів.

Якщо альфа-версія гри успішно проходить тестування, настає наступний етап розробки – опрацювання механіки та об'єктів гри. На даному етапі йде доопрацювання рівнів та механіки гри, і починають додавати перші сюжетні події в гру, такі як відеоролики, сюжетні діалоги та кат-сцени. Також виправляються перші помилки та несправності в коді гри, які були виявлені при тестуванні альфаверсії гри.

Після цього настає етап створення другого прототипу гри, або як заведено говорити, бета-версії. Бета-версія служить для того, що протестувати гру на несправності, фактично бета-версія є практично готовою грою. В ній можуть бути відсутні незначні елементи, які не впливають на геймплей гри. При тестуванні бета-версії гри перевіряється абсолютно все. Часто, особливо якщо гра є мультиплеєрною, на тестування запрошуються звичайні гравці це дозволяє сильно прискорити час проведення тестування, а також зняти частину навантаження з команди розробки.

Якщо гра проходить бета-тестування, вона вирушає на остаточне доопрацювання та виправлення критичних помилок, після чого йде складання фінальної версії гри та її реліз.

Після релізу гри наступний крок це її підтримка. Підтримка полягає у випуску патчів (файлів виправлень помилок у готовому продукті). А для того, щоб продовжити життєвий цикл гри, для неї випускають додатковий контент у

вигляді DLC (Downloadable content), який додає різні предмети та/або можливості для гравця.

Таким чином, стало зрозуміло, що етапи розробки комп'ютерних ігор мало відрізняються, від етапів розробки будь-якого іншого програмного продукту.

Аналіз актуальних ігрових рушіїв

Проаналізувавши декілька інтернет ресурсів можна виділити 4 найпопулярніших ігрових рушіїв на даний момент а саме Unity, Unreal Engine, GameMaker: Studio 2 та Godot давайте розберемо їх більш детально, та виявимо їх переваги та недоліки

Unity є чи не найпопулярнішим ігровим рушієм завдяки своїй гнучкості та розширюваності. Серед плюсів Unity можна відзначити:

Безкоштовне використання. Unity має гнучкий тарифний план (Personal), де платити за використання рушія потрібно тільки якщо розробник отримав від гри дохід у розмірі не менше ніж \$100 тис. протягом 12 місяців Кросплатформовість. Unity дозволяє збирати проекти під PC, Mac, Linux, Android, IOS, tvOS, Xbox, PS4, WebGL та VR-платформи. Усе це є основними платформами для реалізації створюваних ігор на сьогоднішній день.

Магазин ассетів та ком'юніті. Unity має свій власний магазин ассетів - ігрових ресурсів - неподільних частин ігрового контенту. Ассетами можуть бути спрайти, 3d-моделі, звуки, анімації, скрипти, сцени unity і т.д. У магазині є безліч ассетів, розділених за різними категоріями. Причому є як платні, так і безкоштовні ресурси.

Уроки, курси, документація. В інтернеті можна знайти безліч різних курсів або уроків з розробки ігор на Unity. Курси можуть бути як безкоштовними, так і платними. Їх можна знайти як на майданчиках на зразок Youtube, так і на спеціалізованих платформах на зразок Udemy, Coursera. Крім того, самі Unity

Technologies представляють курси по роботі зі своїм двигуном на англійській мові.

Для зручного навчання, крім курсів необхідна ще й зручна документація, де будуть описані методи, класи, влаштування компонентів програми. Unity має зручну докладну документацію, виконану у вигляді Wiki-платформи.

Мінуси в Unity також є:

Середній поріг входу. Unity - рушій, який розвивається з 2005. За цей час у ньому залишилося багато легасі-функцій, зросло безліч варіантів використання. Unity також використовує для програмування мову C#, яка розроблялася та розробляється окремо від даного ігрового рушія, що також вносить складнощі у вивчення та розуміння Unity.

Неповноцінний 2D. Хоча Unity і дуже часто використовується для розробки 2D-ігор, 2D у ньому не повноцінне. Якщо розбиратися більш детально то 2D в Unity це 3D але без координати z. Строго говорячи, це не є виключно мінусом. Такий хід дозволяє зручно змішувати 2D-об'єкти з 3D-об'єктами всередині редактора, але це уповільнює та ускладнює роботи в деяких випадках, при розробці 2D-гри

Unreal Engine - ігровий рушій, що розробляється і підтримується фірмою Epic Games. Unreal Engine, наряду з Unity, є одним з найпопулярніших рушіїв для ігрової розробки. Проте на відміну від Unity, він має більш високий поріг входу та спеціалізований більш великих проектах.

Плюсами Unreal Engine можна назвати:

Практично безкоштовне використання. Epic Games дозволяє використовувати Unreal Engine безкоштовно без будь-яких обмежень поки дохід за час існування проекту не досягне \$1 000 000. Як тільки дохід перевищить цю позначку, вони просять 5% від доходу.

Кросплатформовість. Unreal engine підтримує майже всі сучасні платформи. Найважливіші з них: Windows, Linux, Mac, Xbox, PS, Android, IOS, Nintendo Switch.

Blueprints — візуальна мова програмування, яка допомагає конструювати логіку гри без використання c++ коду. Blueprints не рекомендується використовувати поза написаним на c++ кодом, але для багатьох новачків у програмуванні ця візуальна мова стає зручним трампліном до вивчення Unreal engine. Blueprints зручно замінює код у рамках програмування скриптів для взаємодії з візуальним контентом, внутрішня логіка гри хоч і доступна blueprints, але набагато краще пишеться вже головною мовою програмування. Вже після blueprints інтерпретується як c++, а далі компілюється.

Магазин ассетів та роздачі Epic games. Epic games володіє магазином ассетів який ні чим не поступається магазину Unity. Що стосується високоякісних ресурсів для AAA-ігор, то тут їх навіть більше. Крім того, щомісяця Epic games роздають різні асети зі свого магазину безкоштовно будь-якому розробнику Unreal engine.

Уроки, курси, документація. Unreal Engine є другим за популярністю серед рушіїв, та уроків на різних майданчиках у нього не менше. Щоправда, варто врахувати, що це стосується англomовної частини інтернету. В Україні Unity переважає і знайти інформації щодо нього Українською куди простіше. Epic games також має свої курси та уроки з різних аспектів які стосуються Unreal Engine. Документація теж нічим не поступається Unity.

Мінуси:

Високий поріг входу. Unreal Engine розвивається з 1998 року, коли на ньому була випущена перша гра - Unreal. Тоді це був рушій, заточений під шутери від першої особи, але після багаторічних переробок він став універсальним помічником для розробників ігор будь-яких жанрів. Однак, як і у випадку з unity, залишилося багато легаси-коду. Усе це впливає зручність роботи в

окремих випадках, і розробники це помічають. Звичайно, до цього можна звикнути і перестати бачити в цьому проблему, але для новачків такі проблеми можуть стати перепоною до глибокого вивчення цього ігрового рушія.

Не стандарт у мобільній розробці. Незважаючи на те, що Unreal Engine можна використовувати для створення ігор під мобільні пристрої, він застосовується там рідко. Зазвичай це якісь великі розраховані на багато користувачів проекти. На це є причини, довгий час Unreal Engine не був найкращим варіантом для ігор під смартфони. На сьогоднішній день більшість відомих утиліт з мобільної аналітики, збору статистики написано в основному під Unity а не під Unreal Engine.

GameMaker: Studio Потужний інструмент для професійної розробки ігор. Є одним з найпопулярніших ігрових двигунів. Розробляється компанією YoYo Games . GameMaker Studio відмінно підходить для виробництва невеликих 2D чи простих 3D ігор. проте GameMaker слабко використовується серед ігрових студій, а значить знання даного ігрового рушія не сильного допоможуть при подальшому пошуку роботи

Плюсами можна назвати:

Кросплатформовість. GameMaker підтримує майже усі популярні Платформи: такі як Windows, Macm Ubuntu, Android, IOS, Xbox та PS.

Магазин Ассетів. Game Maker має магазин ассетів не сильно поступається своїм конкурентам, а враховуючи спеціалізованість GameMaker на 2D-іграх, тут можна знайти багато вузькоспеціалізованих ігрових ресурсів

Документація. Детальна документація, яка пояснює багато аспектів не просто текстом, а ще й зображеннями, що допомагає набагато доступніше пояснити багато моментів.

Ком'юніті. GameMaker розвивається з 1999 року і за цей час знайшов велику кількість шанувальників, а значить і матеріалів: курсів, відповідей на форумах

і т.д. Навряд чи сьогодні знайдеться проблема в рушій, яку хтось в інтернеті не знайшов і не вирішив до цього дня.

Повноцінне 2D. На відміну від Unity та Unreal Engine, 2D тут не просто площина у 3D-просторі. Тут підключається саме 2D простір, що набагато зручніше у розробці. Крім того, GameMaker має більше інструментів для роботи з 2D, наприклад, графічний редактор для піксель-арту.

Мінуси: Мала функціональність для 3D. GameMaker не може зрівнятися за функціоналом роботи з 3D з конкурентами, описаними вище. Строго говорячи, він з іншої ліги. GameMaker спеціалізується на невеликих іграх, в основному 2D. Для них він зроблений дуже добре, він має зручний інтерфейс та наповнений необхідним інструментарієм. Однак, як тільки розробник захоче зробити щось більше, йому доведеться переучуватися під використання іншого ігрового рушія.

Обмеження під час використання безкоштовної версії. Безкоштовна версія дається лише на 30 днів використання, причому проекти, створювані у межах цієї версії, неможливо знайти комерційними. Якщо розробник захоче експортувати свою гру під Windows чи Mac, йому буде необхідно купити річну ліцензію за \$39 для кожної операційної системи або безстрокову за 99 \$ для всіх Desktop-платформ. Для мобільних пристроїв за \$199 назавжди. Для консолей \$799 на рік для кожної окремої серії консолей (Xbox, PS, Nintendo Switch)

Godot є безкоштовним рушієм з простим інтерфейсом та підтримкою розробки для різних платформ. Він як і Unreal Engine надає доступ до свого вихідного коду що дозволяє не лише користуватися готовими рішеннями для створення ігор, але і вносити власні зміни, адаптувати рушій для своїх потреб, а також спільно працювати над вдосконаленням програмного забезпечення серед плюсів Godot можна виділити

Швидкодія. Повністю встановлений Godot важить лише 50 Мб. Цей показник значно відрізняє його від конкурентів. Крім того, така маленька вага говорить ще й про його швидкість роботи. Godot містить у собі тільки необхідні елементи і працює куди швидше за своїх конкурентів.

Повна безкоштовність. Godot є продуктом з відкритим вихідним кодом та поширюється за ліцензією MIT. Це говорить про те що Godot engine повинен вільно поширюватися і вимагати плату за скачування цього програмного забезпечення незаконно.

Набирає популярності. Godot - новачок серед ігрових рушіїв. Про нього почали говорити лише останніми роками. Однак він подобається людям. На підтвердження цього можна навести гранти від великих компаній, які дістаються цьому ігровому рушію. Наприклад, грант від ігрової студії Kefir на \$120 тис. чи грант від Epic Games на \$250 тис. Крім того, патрони жертвують йому близько \$15 тис. щомісяця через майданчик Patreon

Мінуси:

Нестача ком'юніті, уроків та ассетів. Godot ще не встиг зібрати велику базу користувачів, тому інтернет не наповнений незліченною кількістю уроків та курсів з цього ігрового движка. У нього є свій магазин ассетів, однак у ньому поки що значно менше контенту, ніж у конкурентів. Однак час має виправити ці недоліки.

Слабке 3D. Проблема ідентична GameMaker Studio, проте Godot у прискореному темпі вводить можливості роботи з 3D і, можливо, в майбутньому стане сильним конкурентом для інших 3D ігрових рушіїв.

Платформи та ринки сбуту На сьогоднішній день існує надзвичайно велика кількість платформ, які надають доступ до різноманітного контенту, та ігрові платформи не є винятком. По всьому світу ми спостерігаємо зростання кількості ігрових сервісів і платформ, які пропонують унікальний вміст та послуги для різних категорій гравців. Від традиційних консолей та

персональних комп'ютерів до мобільних пристроїв і хмарних геймінгових сервісів, вибір платформ для отримання ігрового контенту надзвичайно великий і різноманітний.

Раніше ігри продавалися в спеціалізованих магазинах, та на фізичних носіях, Сьогодні їх можна придбати практично скрізь. Проте в основному це відбувається в мережі на спеціальних сервісах

Зараз основними платформами розповсюдження ігор є ПК, ігри на Windows, Linux і MacOS а також ігрові приставки Xbox і PlayStation, портативна ігрова приставка Nintendo Switch, та мобільні пристрої. До того-ж існують різні версії гри для пристроїв Android та IOS, а також для консолей залежно від моделі консолі, наприклад PS3, PS4 і PS5 або Xbox 360, XboxOne та Xbox Series X

В залежності від платформи розповсюдження ігор відбувається в цифрових магазинах і сервісах. Ігрові консолі Xbox, PlayStation та портативна консоль Nintendo Switch мають власні сервіси де поширюють ігри. Мобільні пристрої мають сервіси залежно від ПЗ, для Android це Play Market, для IOS це App Store.

Персональні комп'ютери, незалежно від ПЗ, можуть використовувати однакові цифрові сервіси. Ці сервіси, крім веб-сайтів, мають додатки, через які здійснюється завантаження ігор та програм. У деяких випадках файл для завантаження можна отримати на емейл. Найпопулярніші сервіси для покупки та завантаження ігор на ПК наступні: Steam, GOG Galaxy, Humble Bundle, Epic Games, Xbox (store). Також деякі видавці випускають ігри у власних сервісах, таких як Ubisoft Connect, EA Play, Battle.net, тощо

Steam є найбільш популярною платформою для продажу ігор для комп'ютерів. Окрім електронного магазину, вона має в собі інтегровану систему спільноти, форуми, майстерню розробників, модифікацій та контент для ігор. Steam також регулярно організовує спеціальні ігрові нагороди та сезонні розпродажі зі знижками.

Найбільшими споживачами ігрових продуктів серед країн є Китай, США та Японія, що за різними підрахунками мають 700, 200 і 75 мільйонів гравців відповідно.

В залежності від регіону змінюються і вподобання гравців. Наприклад, у США популярні ігрові консолі, тоді як у Європі переважає сегмент ПК, а в Китаї найпопулярнішими є мобільні пристрої. Це добре пояснює, чому найбільшу частку ринку наразі займають саме мобільні ігри.

Зрозуміло, що кожен ринок має свої унікальні характеристики та можливості для розробників ігор, і вибір платформи для розміщення ігор залежить від стратегії розробника та його цілей.

Висновки до розділу 1

На основі проведеного аналізу можна зробити висновок, що індустрія комп'ютерних ігор продовжує розвиватися та має величезний потенціал для подальшого зростання.

Розробка ігор стає все більш доступною завдяки різноманітним ігровим рушіям і платформам для поширення контенту. Крім того, ігри стають важливим інструментом для навчання та розвитку, що робить їх не лише засобом розваги, але й важливим культурним явищем.

Завдяки великій кількості доступних платформ та рушіїв, розробники мають широкі можливості для реалізації своїх ідей. Кожен рушій та платформа мають свої переваги та недоліки, що дозволяє розробникам вибирати найбільш підходящі інструменти та стратегії для досягнення успіху у цьому динамічному середовищі.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА ДИЗАЙН

Графіка

У відеоіграх існують різноманітні типи графіки, які допомагають створювати візуальні ефекти та атмосферу гри. Проаналізувавши декілька сайтів в інтернеті я зміг виділити такі типи графіки:

Растрова графіка:— це цифрові зображення, які складаються з крихітних прямокутних пікселів одного розміру, певним чином розташованих у прямокутній сітці. У растровій графіці графічне зображення нагадує мозаїку, що складається з маленьких камінців. Пікселі є найменшими об'єктами растрового зображення. Чим більша кількість пікселів і чим менші їх розміри, тим якісніше виглядає зображення.

Растрову графіку найчастіше використовують у тих випадках, коли потрібно якісно передати повну гаму відтінків кольорів зображення для його реалістичності.



Рис 2.1 Приклад растрового зображення

Векторна графіка: Цей тип графіки базується на математичних розрахунках, які визначають форму об'єктів, що відображаються. Інакше кажучи векторна графіка це зображення в комп'ютерній графіці з сукупності геометричних примітивів – точок, ліній, кривих, полігонів, тобто об'єктів, які можна описати математичними виразами. Векторна графіка для опису зображення використовує вектори, на відміну від растрової графіки, яка описує зображення як масив пікселів

Векторні зображення використовують для створення графічних об'єктів, для яких має значення збереження чітких та ясних контурів навіть при зміні розмірів це креслення, схеми, логотипи, мапи, діаграми тощо.



Рис 2.2 Приклад векторного зображення

3D-графіка: Це тривимірна графіка, що використовується для створення тривимірних об'єктів, та світів. Дана графіка працює в кінематографії, архітектурній справі, комп'ютерних іграх, а також у поліграфії, науці та промисловості.

3D графіка являється більш реалістичним відображенням інформації, оскільки в повсякденному житті людина бачить світ саме в трьох вимірах. З допомогою

цього виду графіки дуже легко маніпулювати такими ефектами як освітлення і тінь.

2D-графіка це стилістика, в якій візуальні елементи відображаються в двовимірному просторі. Вона широко використовувалася в ранніх відеоіграх і залишається популярною завдяки своїй унікальній естетиці, простоті та можливості створення атмосферних світів. 2D графіка може бути представлена в різних стилях, від піксель-арту до високодеталізованих ілюстрацій. Вона має свої унікальні переваги та характеристики. Навіть у часи розквіту тривимірних ігор, 2D-графіка залишається популярною, особливо в жанрах ретро-ігор, інді-ігор та мобільних ігор. І хоча 2D-графіка може здаватися менш реалістичною порівняно з 3D, вона все ще має великий потенціал для створення унікальних та цікавих вражень для гравців.

2.5D графіка - це гібридна форма, яка поєднує елементи як 2D, так і 3D графіки для створення вражаючих візуальних ефектів. У цьому підході деякі об'єкти чи шари гри можуть бути відображені у 3D, тоді як інші залишаються в 2D.

Піксель-арт: Це мистецтво, де графіка створюється шляхом розміщення пікселів вручну. Вона нагадує растрову графіку, але використовується спеціально для створення стилізованих, абстрактних або ретро-стилізованих зображень. На старих комп'ютерах, в іграх для Game Boy, старих ігрових приставок і багатьох іграх для мобільних телефонів в основному використовується піксельна графіка, оскільки це єдиний спосіб зробити чітким невелике зображення при малій роздільній здатності екранів, характерній для цих пристроїв.

Піксель-арт може бути створений вручну, піксель за пікселем, або з використанням спеціалізованих програм, які дозволяють автоматизувати процес створення. І хоча піксель-арт часто асоціюється з 2D-іграми, він також може використовуватися в 3D-графіці, де пікселі представлені у текстурах для стилізації об'єктів.



Рис 2.3 приклад зображення в стилі піксель арт

Комікс-стиль: Комікс стиль в іграх, або cel-shading, являє собою художню техніку, яка імітує вигляд і відчуття малюнків, типових для коміксів або мультфільмів. Цей стиль виділяється своєю унікальністю та візуальною привабливістю, роблячи ігри впізнаваними та виразними. Основними характеристиками комікс стилю є жирні, темні контури навколо об'єктів та плоскі кольори, що робить їх схожими на малюнки в коміксах.

Комікс стиль в іграх є важливим елементом сучасної індустрії, надаючи розробникам можливість створювати візуально привабливі та художньо виразні проєкти.

Ці типи графіки можуть використовуватися окремо або комбінуватися, щоб створити унікальні візуальні стилі та ефекти в іграх.

Аналіз існуючих розробок

Оскільки в якості проекту був обраний однокористувацький 2D – платформер, тому й основних конкурентів варто шукати в тій же жанровій категорії. Але варто пам'ятати, що цей жанр втратив свою популярність, через чого сучасні ігри цього жанру часто не є чистими платформерами. Найяскравішими представниками цього жанру останнім часом були:

- **Mark of the Ninja;**
- **Ori and the Blind Forest;**
- **Dust: An Elysian Tail;**
- **Limbo;**

Mark of the Ninja – двомірний стелс-екшен, розроблений Канадською студією Klei Entertainment для Xbox 360 і ПК, випущена в сервісах Xbox Live Arcade і Steam. Ця гра не є чистим платформером, але має велику кількість атрибутів даного жанру. Сюжет оповідає про безіменного ніндзя, який носить звання чемпіону клану Хісомі, оскільки отримав особливе татуювання, яке посилює його здібності, але натомість поступово зводить його з розуму. Він намагається звільнити членів його клану, які вижили, які потрапили в полон після атаки від ряду Гессен. Геймплей гри заснований на потайному проходженні рівня, уникненні контактів з противниками та знешкодження пасток. Головний герой вміє ховатися у вентиляції, каналізаційних люках та за предметами інтер'єру. Основна зброя — меч-ніндзято, яким можна виконати безшумне вбивство нічого не підозрюючої жертви а також бамбукові дротики, призначені для відволікання та знищення джерел світла. Незважаючи на наявність смертельної зброї, гра однаково орієнтована як на усунення всієї охорони

рівня, так і на безкровне проходження. У процесі проходження купується і відкривається додаткове спорядження атакуючої та відволікаючої спрямованості.

Практично за кожну дію нараховуються поінти, на які можна відкрити нові прийоми, придбати нове чи покращити старе спорядження.

Додатково на кожному рівні видається 3 різні бонусні завдання. Після виконання завдань одного типу стає доступним костюм, спеціалізований для певного стилю проходження.

Також на кожному рівні, у прихованих місцях можна знайти артефакти, які дадуть вам кілька сотень поінтів. Ще на рівні присутні різні головоломки, які називаються Кімнатами випробувань. Після проходження головоломки ви отримуєте 1 з 3 прихованих на рівні свитків Хі-сомі, котрі також дають додаткові поінти. В основному головоломка полягає в тому, щоб пройти кімнату, уникнувши пасток.

Ця гра є чудовим стелс-екшеном серед двовимірних ігор подібного жанру, яких не так багато виробляється останнім часом.

Вона має стильне графічне оформлення, яке задає атмосферу всієї історії. Мінімалістичний та інформативний інтерфейс, але дещо перевантажене керування, може відбити бажання грати далі

Ori and the Blind Forest – гра в жанрі платформер розроблена

студією Moon Studios та видана Microsoft Studios для платформ Windows та Xbox One. Гра повністю виконана у двовимірному стилі та написана на ігровому рушію Unity

Сюжет розповідає про лісовий дух Орі, який є невеликим листочком величезного духу-дерева, що захищає чарівний ліс.

Якось штормовий вітер відриває Орі і забирає його далеко в ліс. Впавши на землю він перетворюється на невелику білу істоту і знаходить Нару – духа, що живе у лісі і схожого на великого чорного ведмеда, - яке стає для Орі матір'ю.

Згодом, темний дух Куро, краде серце Древа, і ліс починає стрімко в'янути. Незабаром після цього Нару так само гине, і Орі залишається один. Він пускається в мандрівку по лісу, що гине, але незабаром теж вмирає. Але Древо, поряд з яким Орі закінчує своє життя, використовує сили, що залишилися, воскрешає його і просить очистити чарівний ліс від темряви.

Гравець управляє невеликою білою істотою Орі та невеликим лісовим духом Сейном, який його захищає. Ігрова механіка представляє собою типовий платформер, гравець переміщається великою картою долаючи різні перешкоди і бореться з різними ворогами. У грі також присутня проста система прокачування у вигляді невеликого дерева вмінь.

Для придбання нових умінь потрібно збирати невеликі згустки енергії, розкидані світом гри. Карта світу завантажується відразу і повністю, та гравець сам вибирає, куди піти. Єдиним обмеженням у цьому випадку є необхідність певних здібностей для проходження певних ділянок.

Гра відрізняється неймовірно красивим оформленням, що, безперечно, виділяє її серед інших ігор цього жанру. За чудовим сюжетом, що розповідається в грі, цікаво спостерігати, а звуковий супровід відмінно задає необхідну атмосферу. Мінусом можна вважати лише трішки ускладнений геймплей через неінтуїтивне управління, але в усьому іншому, цю гру можна вважати еталоном жанру.

Dust: An Elysian Tail - екшен-РПГ з елементами платформера, розроблена незалежними розробником Діном Додріллом і опублікована Microsoft Studios. Ця гра так само не є чистим платформером.

Dust відбувається у вигаданому світі Фалани, населеному антропоморфними тваринами. Гравець керує головним героєм, Дастом, який намагається згадати

своє минуле. Надалі йому варто дізнатися таємницю своєї появи, а також врятувати свій світ від тиранії Генерала Гая.

Даст має легендарний меч, Лезо Аарах. Фіджет, Охоронець меча, виступає як компаньйон Даста і може використовувати магічні атаки. Поки гравець та його компаньйон подорожують світом, вони можуть придбати бонуси, які постійно змінюватимуть геймплей, такі як можливість подвійного стрибка або вхід на раніше недоступні території. Що включає в себе елементи рольових ігор, Даст може отримувати поінти досвіду, перемагаючи ворогів, і в свою чергу підвищувати рівень. Вони можуть бути використані, щоб підняти різні атрибути, такі як здоров'я, сила, захист чи магія.

Геймплей стандартний для платформерів - гравець пересувається по рівням, долаючи перешкоди, але варто розуміти, що на відміну від попередніх 2х ігор, тут найбільш важливим аспектом є розвиток персонажу та вбивство ворогів. Через це зрозуміло, що данна гра скоріше під-вид платформера, який зветься метроїдванія

Limbo— гра жанру пазл-платформер, створена компанією Playdead.

Як такого сюжету тут немає, все, що відомо гравцеві, це мета гри – дійти до загадкової дівчинки. Геймплей є типовим платформером з елементами пазла - гравцеві потрібно дістатися від початку рівня до його кінцю уникаючи пасток і ворогів, попутно вирішуючи прості просторові головоломки. Гра виконана в стилістиці мінімалізм, і в ній переважають темні тона, через це пастки не завжди помітно і гравець може легко загинути.

Основною проблемою гри є її власна стилістика, яка хоч і створює потрібну атмосферу, в деяких випадках може дуже сильно перешкодити під час гри. Так само геймплей для багатьох людей може здатися нудним і нецікавим, що так само позначається на відношенні до самої гри. Таким чином, зрозуміло, що гра спочатку не розрахована на широку аудиторію

Створення концепції гри

Першим кроком у створенні гри є розробка концепту. Відповіді на питання "що?", "як?", "для чого?", "чому?", "навіщо?" "який все матиме вигляд і як співпрацюватиме?" є ключовими для визначення основних параметрів проекту.

Коли команда розробників працює над концепцією, вони обговорюють ідеї, визначають основні механіки гри, розробляють сюжет та персонажів, вирішують, який буде вигляд світу гри і як всі елементи будуть взаємодіяти.

Це найдовший і найбільш важливий етап розробки, бо від його успішного завершення залежить подальший успіх проекту. Саме тут формується фундаментальна основа, яка буде визначати всі подальші кроки розробки.

Концепція гри наступна. Події відбуваються на декількох локаціях а саме в лісі та згодом у печері



Рис 2.4 що демонструє задній фон лісу

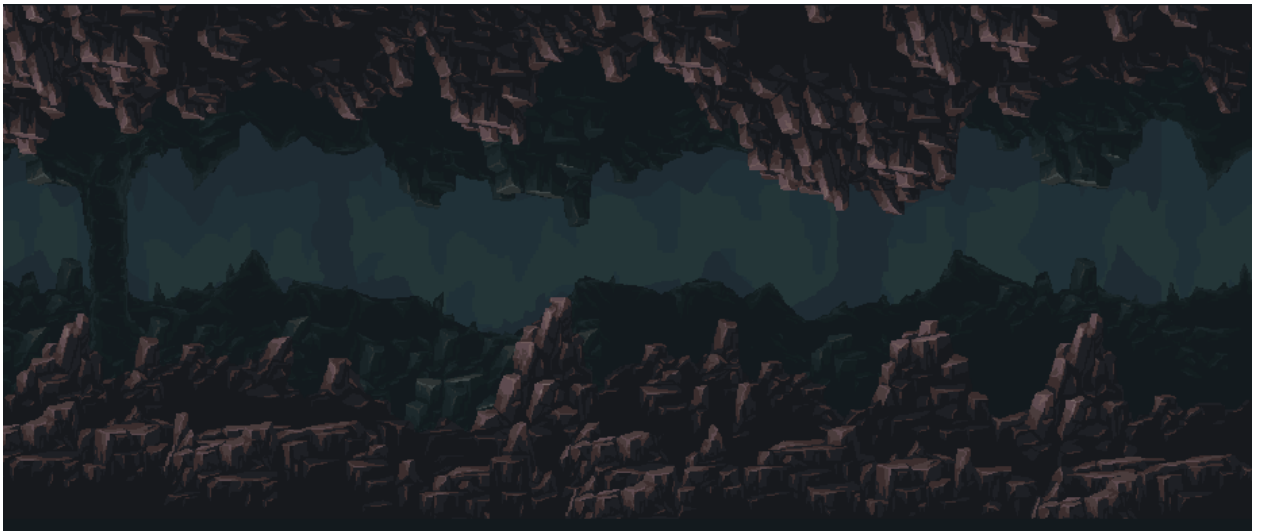


Рис 2.5 що демонструє задній фон печери

Головному герою потрібно подолати фінального боса щоб завершити гру, як такого сюжету в грі немає, як і фінальної мети, вся гра більш зав'язана на геймплеї.

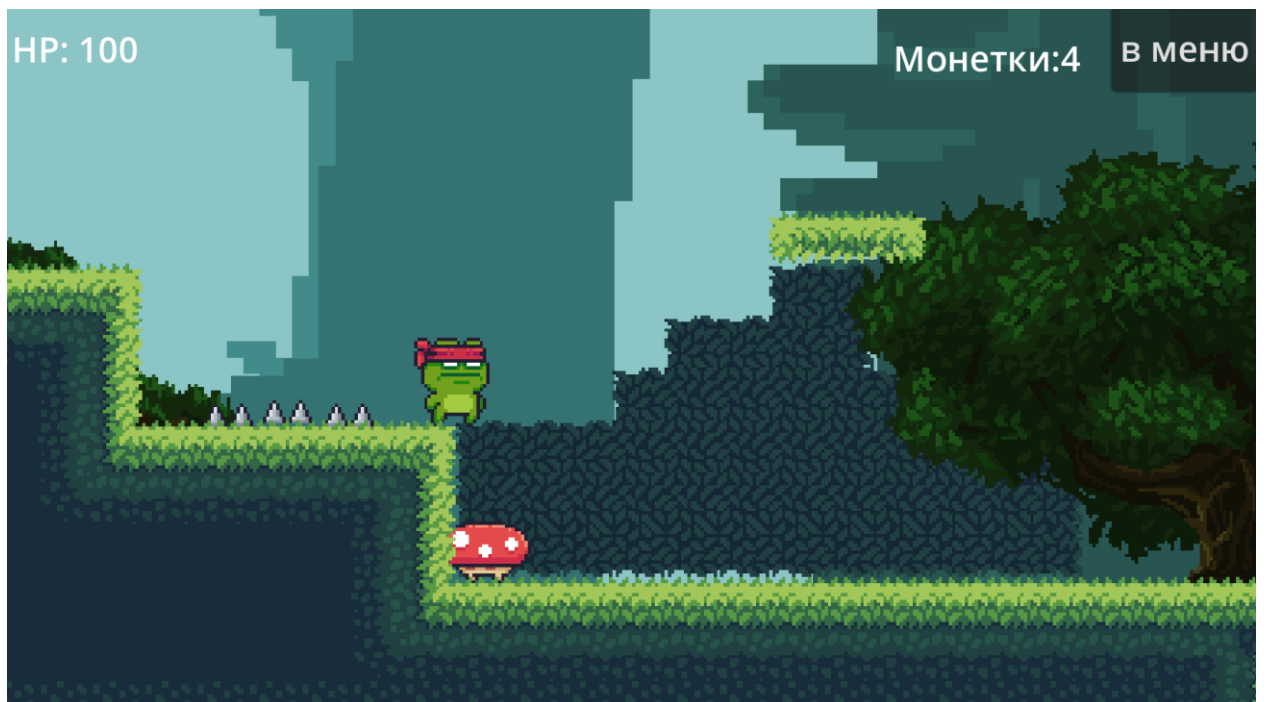


Рис 2.6 Приклад геймплею на локаціях

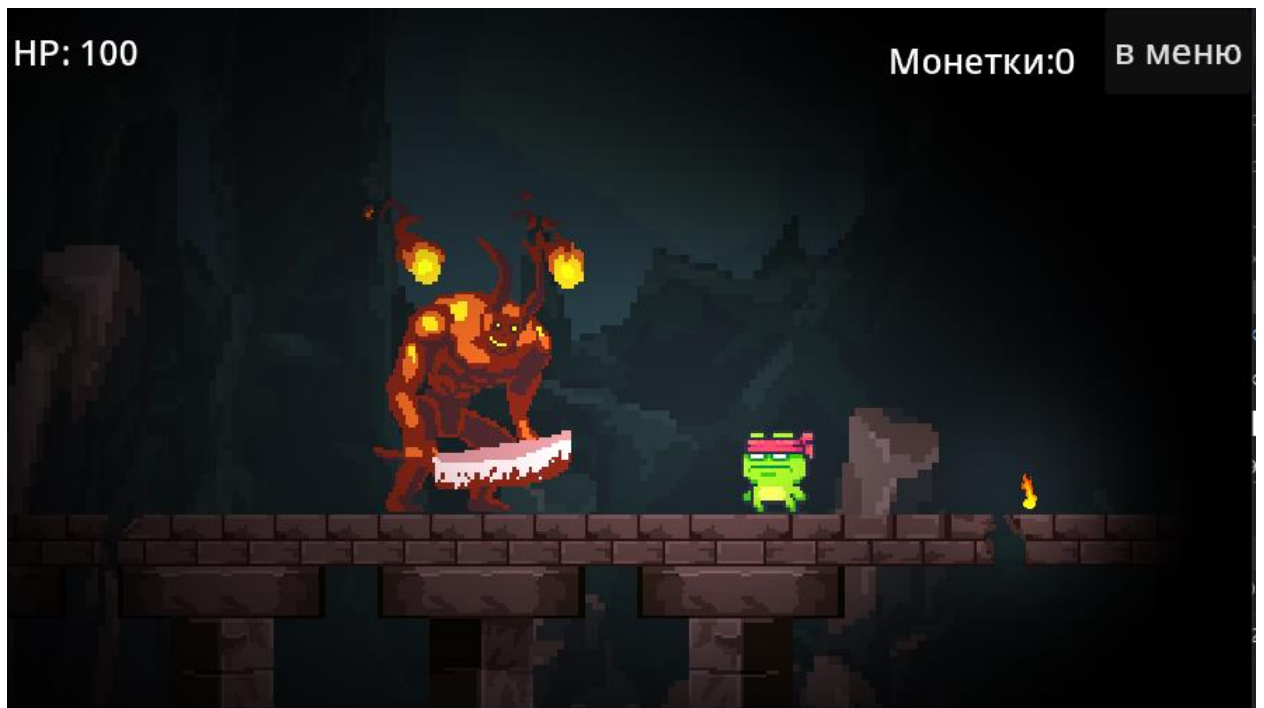


Рис 2.7 Приклад геймплею на локаціях

В ході дослідження більшість асетів було взято з безкоштовних ресурсів таких як itch.io, головний герой став так би мовити основою, фундаментом подальшої розробки



Рис 2.8 головний герой гри

При створенні гри я надихався серією ігор Hollow Knight і хотів відтворити щось подібне проте через нестачу досвіду та навичок від цієї ідеї довелося відмовитися. Згодом я пересмислив геймплей та дизайн і вирішив відтворити щось більш схоже на Super Mario Bros і хоча гра вийшла дещо сируватою я вважаю що зі своїм завданням я впорався.

Godot Engine

Основою для розробки гри виступає ігровий рушій Godot Engine. Це мультиплатформний програмний засіб розробки відеоігор, який використовується для їх створення та запуску. Написання ігрової логіки забезпечується використанням власної високорівневої динамічно типізованої скриптової мови програмування під назвою GDScript, синтаксис якої віддалено нагадує мову Python. Відмінністю від Python є насамперед обов'язкове визначення області видимості змінної через ключове слово var та оптимізація мови під потреби системи сцен та вузлів рушія.

Загальна архітектура двигуна побудована навколо концепції дерева з наслідуваних сцен. Кожен елемент сцени (нода), будь-якої миті сам може стати повноцінною сценою. Тому при розробці можна легко змінювати всю архітектуру проекту, розширювати її елементи в будь-який бік і працювати з комплексними сценами на рівні простих абстракцій.

Всі ігрові ресурси, від скриптів до графічних ассетів та ігрових сцен, зберігаються в папці проекту як звичайні файли, і не є складною базою даних проекту. Ресурси, які не є комплексними даними, зберігаються у простих текстових форматах (наприклад, скрипти та сцени, на відміну від моделей та текстур). Ці рішення дозволяють значно спростити різним командам розробників роботу із системами управління версіями.

Godot був створений з акцентом на простоту використання, ефективність і багатофункціональність, що робить його популярним серед незалежних розробників та малих студій.

Висновки до розділу 2

На основі проведеного аналізу, можна зробити висновок, що у відеоіграх існує велика різноманітність типів графіки, кожен з яких має свої унікальні характеристики та застосування.

Аналіз ігор у жанрі 2D-платформерів, таких як "Mark of the Ninja", "Ori and the Blind Forest", "Dust: An Elysian Tail" та "Limbo", показує, що успішні ігри цього жанру часто поєднують захоплюючий сюжет, стильну графіку та інноваційний геймплей.

Таким чином, створення успішної відеоігри вимагає ретельного планування та використання різних типів графіки для досягнення бажаного візуального стилю та атмосфери, а також врахування досвіду та особливостей інших успішних проектів у жанрі.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

Опис інтерфейсу ігрового рушія

Перш ніж починати створення проекту нам необхідно завантажити ігровий рушій, це можна зробити з офіційного сайту Godot. Варто зауважити що є дві версії Godot для операційної системи Windows а саме стандартна версія з язиком програмування GD Script а також версія з язиком програмування C#



Рис 3.1 Офіційний сайт Godot

Для цієї бакалаврської роботи було обрано перший варіант а саме GD Script тому що він простіше в користуванні а також займає менше місця в редакторі. Нижче наведено приклад

```

public override void _Process(double delta)
{
    var velocity = Vector2.Zero; // The player's movement vector.

    if (Input.IsActionPressed("move_right"))
    {
        velocity.X += 1;
    }

    if (Input.IsActionPressed("move_left"))
    {
        velocity.X -= 1;
    }

    if (Input.IsActionPressed("move_down"))
    {
        velocity.Y += 1;
    }

    if (Input.IsActionPressed("move_up"))
    {
        velocity.Y -= 1;
    }

    var animatedSprite2D = GetNode<AnimatedSprite2D>("AnimatedSprite2D");

    if (velocity.Length() > 0)
    {
        velocity = velocity.Normalized() * Speed;
        animatedSprite2D.Play();
    }
    else
    {
        animatedSprite2D.Stop();
    }
}

```

Рис 3.2 Приклад коду на мові програмування C#

```

func _process(delta):
    var velocity = Vector2.ZERO # The player's movement vector.
    if Input.is_action_pressed("move_right"):
        velocity.x += 1
    if Input.is_action_pressed("move_left"):
        velocity.x -= 1
    if Input.is_action_pressed("move_down"):
        velocity.y += 1
    if Input.is_action_pressed("move_up"):
        velocity.y -= 1

    if velocity.length() > 0:
        velocity = velocity.normalized() * speed
        $AnimatedSprite2D.play()
    else:
        $AnimatedSprite2D.stop()

```

Рис 3.3 Приклад коду на мові програмування GD Script

Після скачування та створення нового проекту нас вітає інтерфейс ігрового рушія.

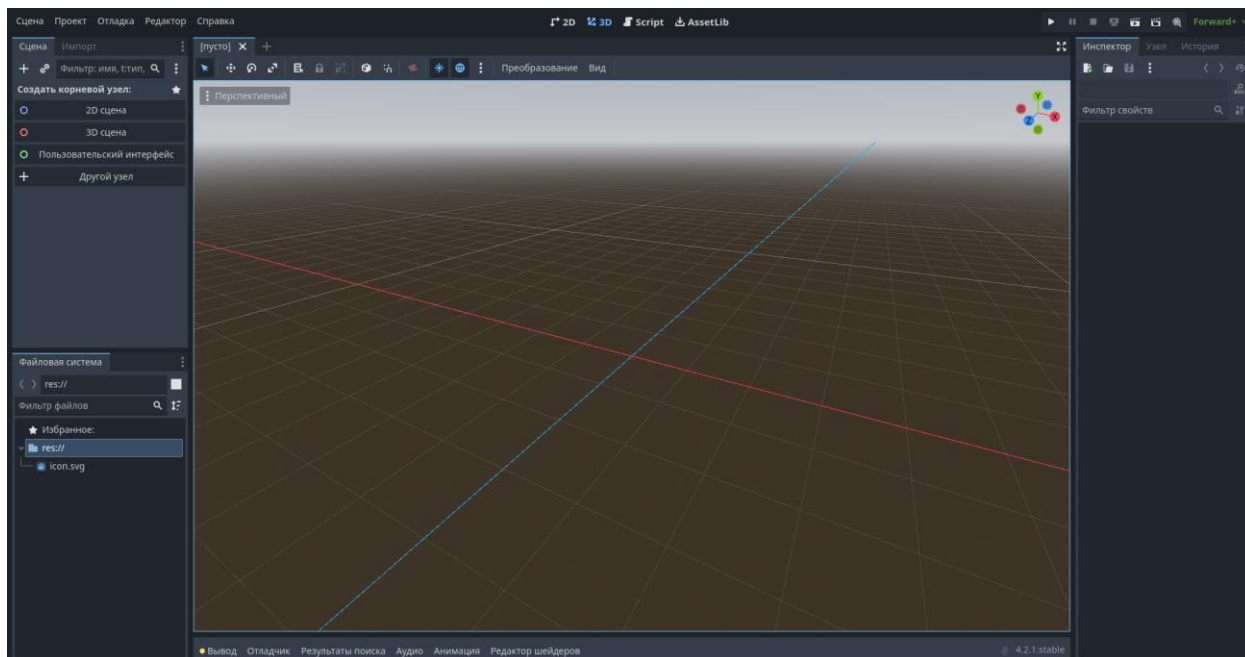


Рис 3.4 Інтерфейс ігрового рушія

Інтерфейс містить меню, головні екрани і кнопки тесту відтворення вздовж верхнього краю вікна.

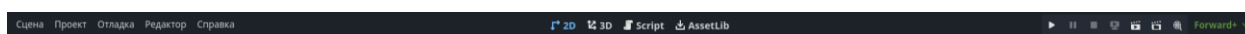


Рис 3.5 Головні екрани і кнопки тесту

В центрі знаходиться вікно перегляду з панеллю інструментів у верхній частині, де знаходяться інструменти для переміщення, масштабування або блокування вузлів сцени.

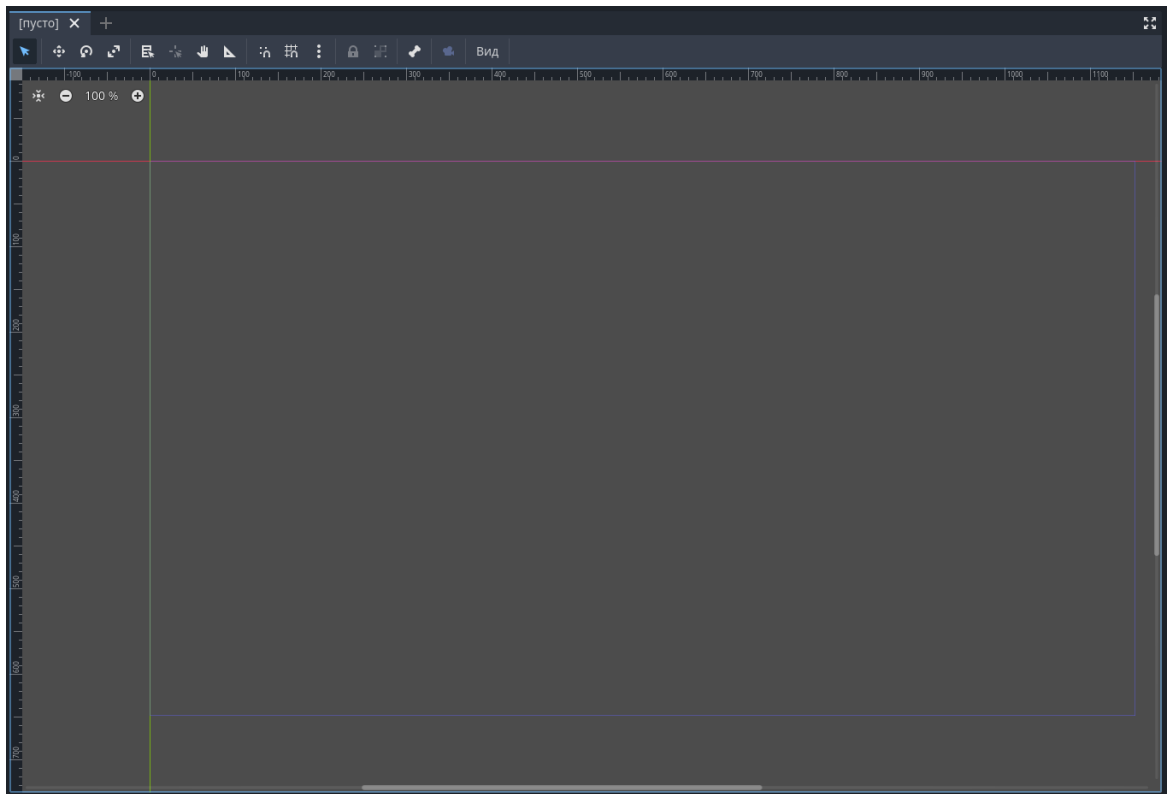


Рис 3.6 Вікно перегляду

По обидва боки від вікна перегляду розташовані переміщувані панелі, розберемо їх більш детально

Панель Scene (Сцена) містить список вузлів активної сцени.

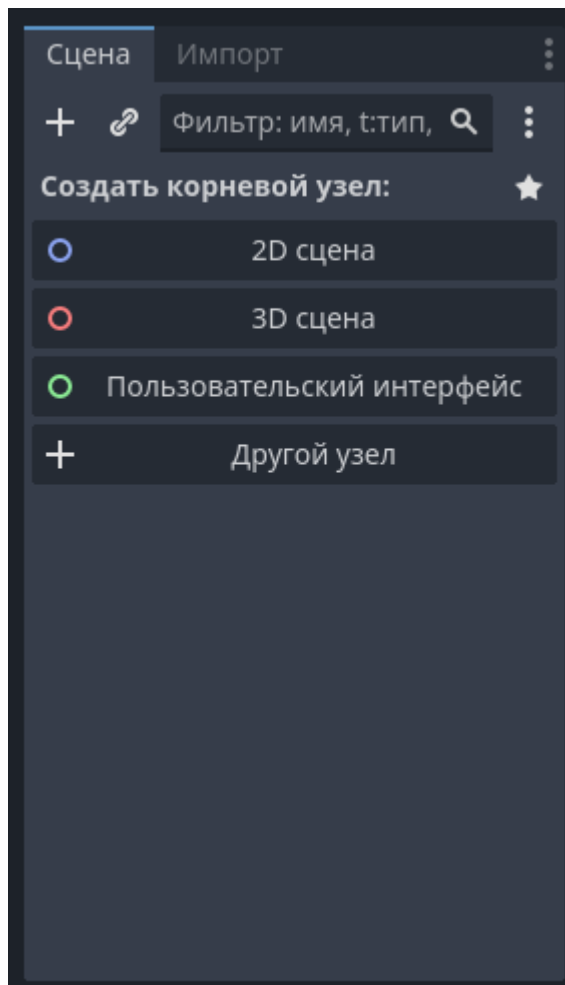


Рис 3.7 Панель сцена

Панель з файловою документацією (FileSystem) містить файли вашого проекту, включно зі скриптами, зображеннями, звуковими семплами тощо

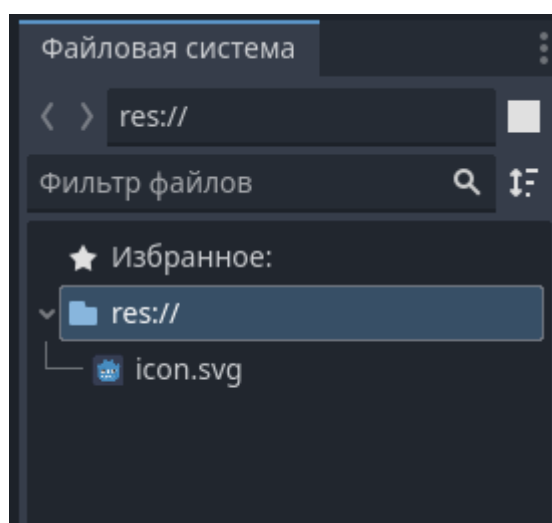


Рис 3.8 Панель з файловою документацією

Панель Інспектор (Inspector) дозволяє редагувати властивості вибраного вузла.

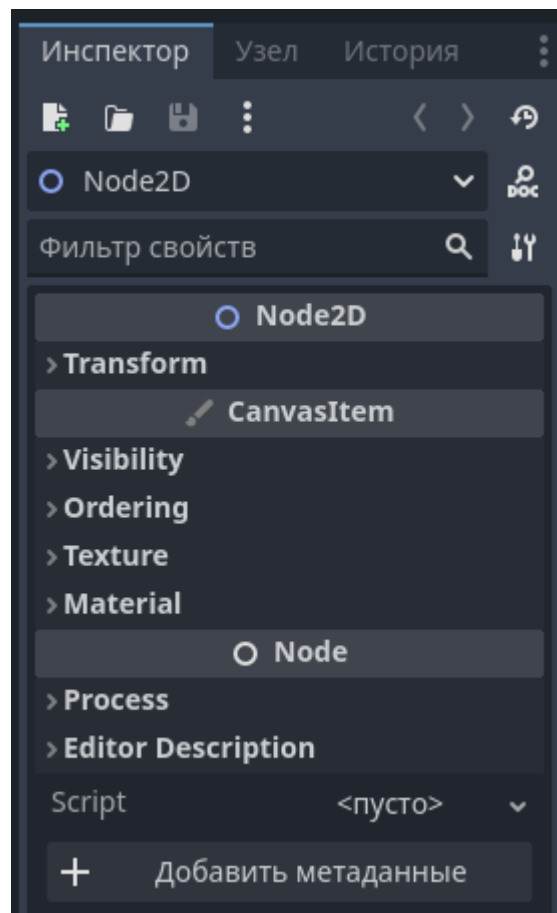


Рис 3.9 Панель Інспектор

У верхній частині редактора розташовані чотири кнопки головного екрана: 2D, 3D, Script і AssetLib.

2D-екран призначений для всіх типів ігор він дозволяє працювати не лише з 2D-іграми, на ньому також можна створювати свої інтерфейси.

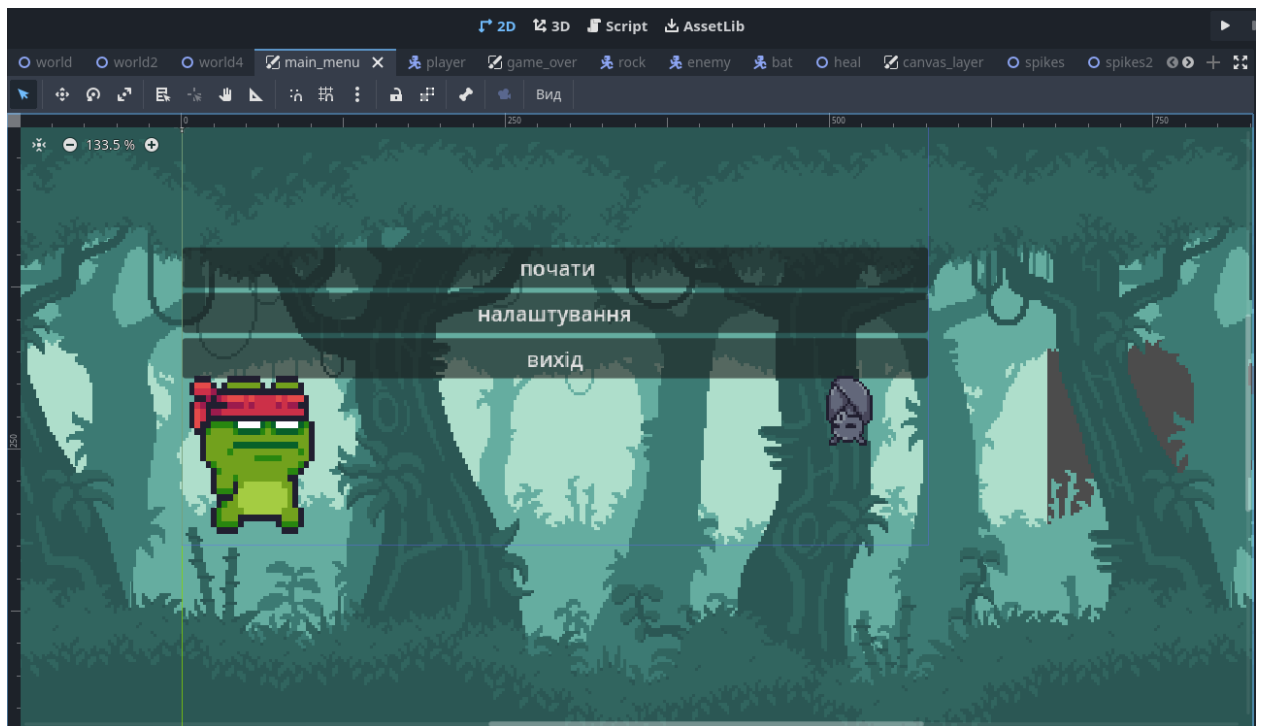


Рис 3.10 2D-екран

На 3D-екрані можна працювати з сітками, світлом і рівнями дизайну для 3D-ігор.

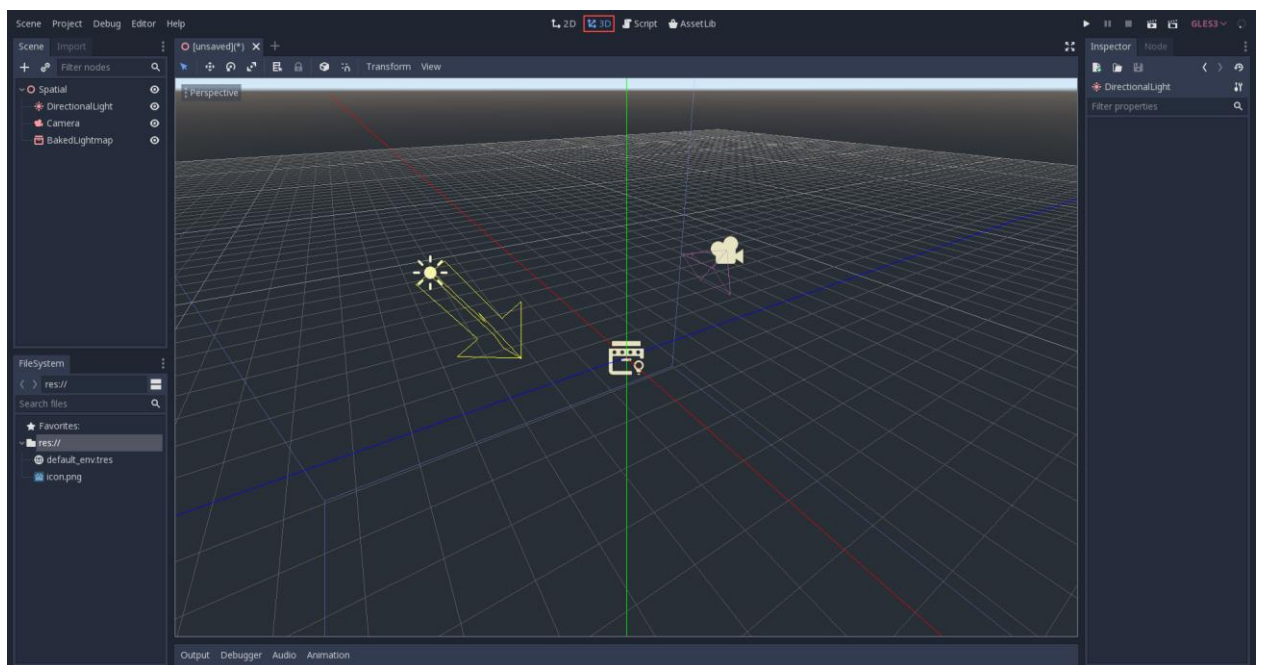


Рис 3.11 3D-екран

Екран сценарію (Script screen) він використовується для написання коду, який керує логікою гри, поведінкою об'єктів та іншими аспектами проекту.

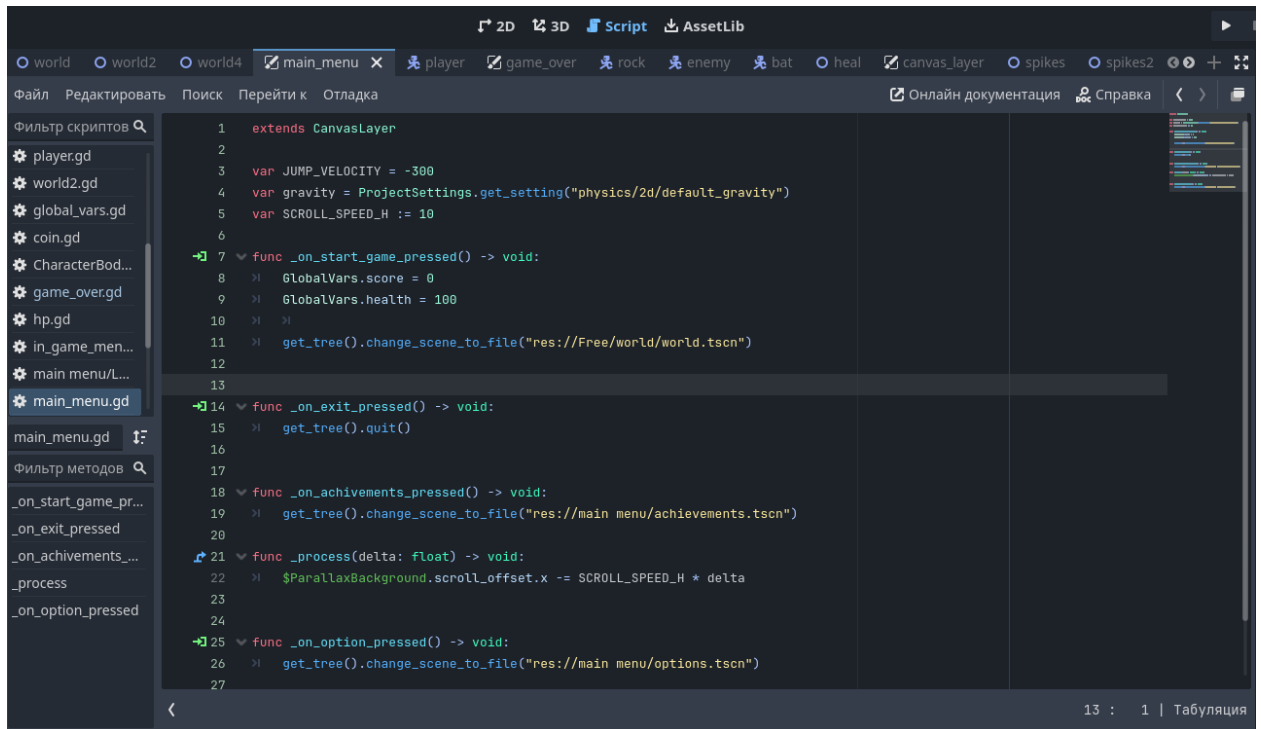


Рис 3.12 Екран сценарію

І нарешті, AssetLib - це бібліотека безкоштовних доповнень, вона дозволяє розробникам легко знаходити, завантажувати та інтегрувати різні ресурси у свої проекти..

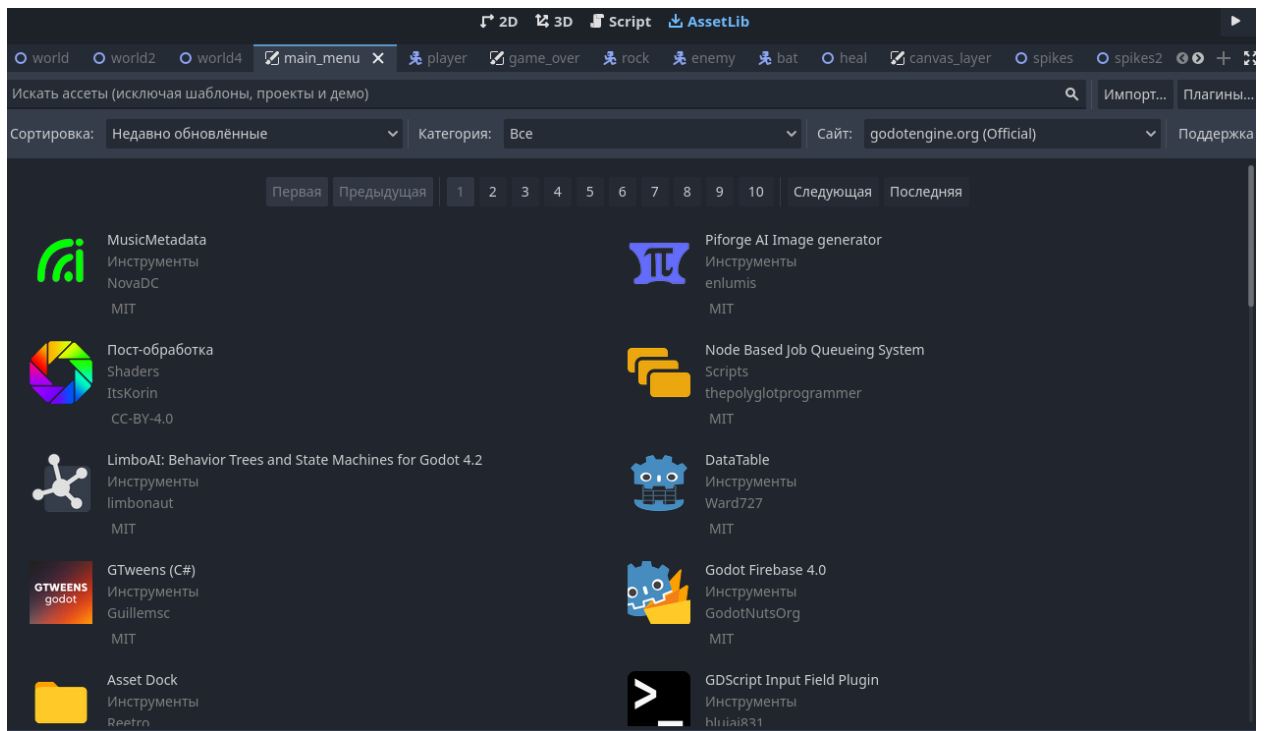


Рис 3.13 AssetLib

З інтерфейсом розібралися тепер можна переходити до створення проекту

Створення проекту

Після того як ми розібралися з інтерфейсом рушія створимо проект, для цього після запуску Godot натискаємо клавішу новий та створюємо папку в якій ми в майбутньому будемо зберігати всі наші данні, включаючи ассети, спрайти персонажів та скрипти.

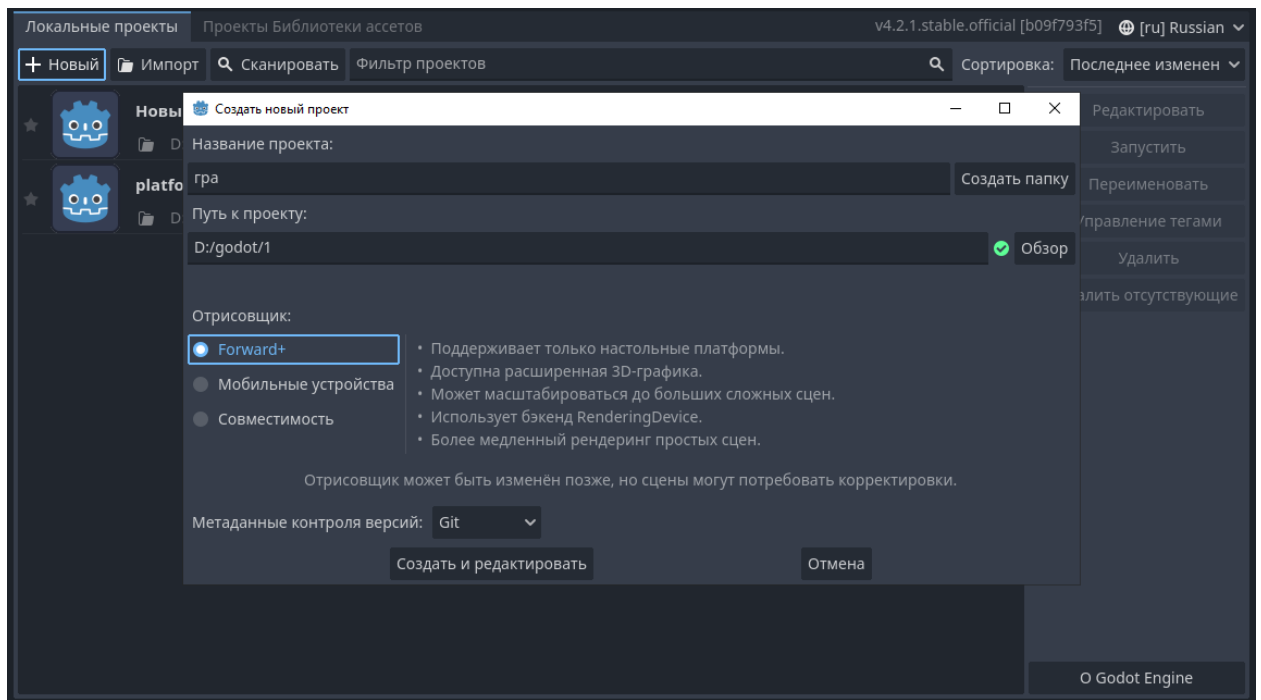


Рис 3.14 створення проекту

Після створення нового проекту візьмемо декілька готових ресурсів та збережемо їх в нашій папці.

Для створення проекту було взято готові ассети з сайту ich.io, цей ассет пак являється повністю безкоштовним і має доволі багато готових ресурсів.

Download Now

Pixel Adventure

This project contains all the graphic images that are presented in the Demo version above (characters, objects, tilesets, items, etc)

Play the Demo to see all the animation and features in action.


Use the arrows on your keyboard to move left, right and jump.

On the top-right corner are buttons to move you through the different scenes.

Combine and modify all these elements, and enjoy creating your own adventure.

Note:


- This project contains graphics images only, in PNG format.
- All animations have a speed of 20 FPS or 50 MS.

License: 

These assets are released under a Creative Commons Zero (CC0) license. You can distribute, remix, adapt, and build upon the material in any medium or format, even for commercial purposes.

Attribution is not required.

If you want to have a closer look at the license, check [here](#). Have fun! :)



Download all the 20 enemy characters in Pixel Adventure 2




Рис 3.15 Ассет пак Pixel Adventure

Після скачування копіюємо файли в нашу папку з грою.









| | | | | |
|---|-----------------|------------------|---------------------|------|
|  | .godot | 28.05.2024 15:08 | Папка с файлами | |
|  | Free | 28.05.2024 15:09 | Папка с файлами | |
|  | .gitattributes | 25.05.2024 19:30 | Файл "GITATTRIBU... | 1 КБ |
|  | .gitignore | 25.05.2024 19:30 | Файл "GITIGNORE" | 1 КБ |
|  | icon | 25.05.2024 19:30 | Microsoft Edge H... | 1 КБ |
|  | icon.svg.import | 25.05.2024 19:30 | Файл "IMPORT" | 1 КБ |
|  | node_2d | 26.05.2024 18:20 | Файл "TSCN" | 1 КБ |
|  | project.godot | 28.05.2024 15:08 | Файл "GODOT" | 1 КБ |

Рис 3.16 Папка з грою

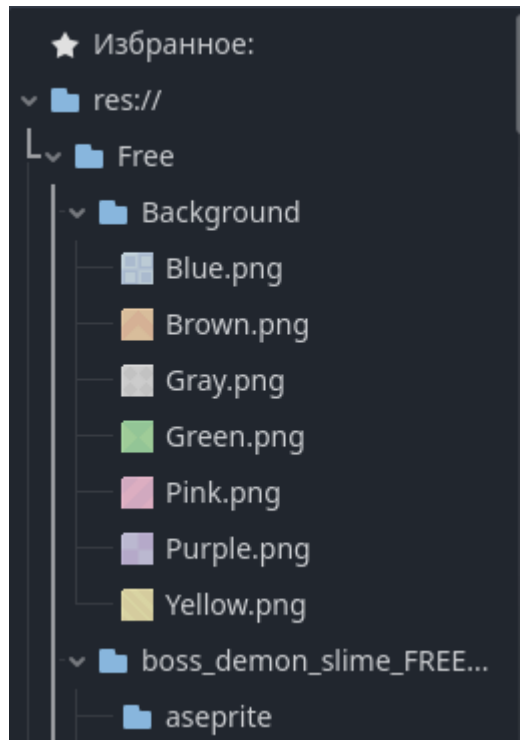


Рис 3.17 Набір ресурсів всередині ігрового рушія

Тепер ми можемо починати створювати гру, для початку додамо задній фон та зробимо головне меню.

Додавання заднього фону

В Godot є декілька вузлів які допоможуть нам легко створити задній фон.

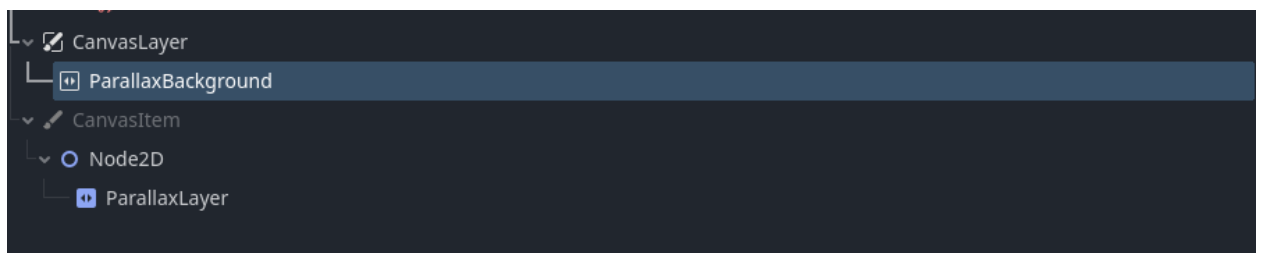


Рис 3.18 Вузол ParallaxBackground

Для початку створимо нову сцену та додамо вузол Parallax Background це основний вузол в який ми будемо додавати слої щоб картинка була з ефектом паралаксу а не статична

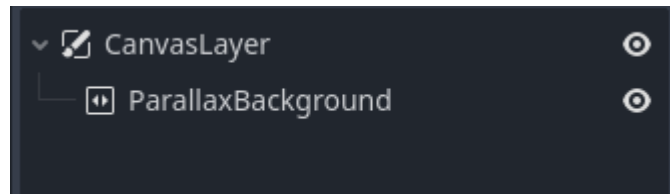


Рис 3.19 Вузол ParallaxBackground

Тепер додаємо декілька вузлів Parallax layer це треба для того щоб налаштувати кожен з шарів окремо додавши в Parallax layer готові спрайти

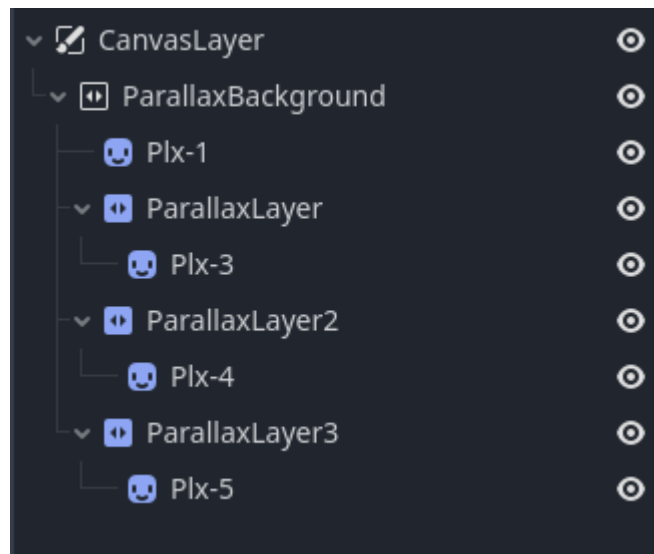


Рис 3.20 Декілька вузлів Parallax layer

Також важливо додати скрипт до основної ноди щоб наші шари з паралаксом рухалися, для цього зробимо змінну `var SCROLL_SPEED_H` та дамо їй якесь значення, наприклад 10

```
1 extends ParallaxBackground
2
3 var SCROLL_SPEED_H := 10
```

Рис 3.21 Змінна var

Тепер робимо функцію та прив'язуємо до неї основну ноду з паралаксом вона поступово переміщає `ParallaxBackground` вліво (оскільки зменшується значення `x`) зі швидкістю, визначеною `SCROLL_SPEED_H`, помноженою на `delta`, щоб рух був постійним незалежно від частоти кадрів.

```

1  extends CanvasLayer
2
3  var SCROLL_SPEED_H := 10
4
5  func _process(delta: float) -> void:
6      $ParallaxBackground.scroll_offset.x -= SCROLL_SPEED_H * delta
7

```

Рис 3.22 Прив'язка основної ноди

В налаштуваннях кожного з шарів робимо різну швидкість та відзеркалюємо шари, це робиться для того щоб утворити ефект безкінечного фону.



Рис 3.23 Ефект безкінечного фону

Тепер зробимо декілька функціональних клавiш для нашого головного меню, для цього додамо вузол MarginContainer та VBoxContainer для того щоб згрупувати наші клавiшi а також додамо самi клавiшi та прив'яжемо до них сигнал pressed()

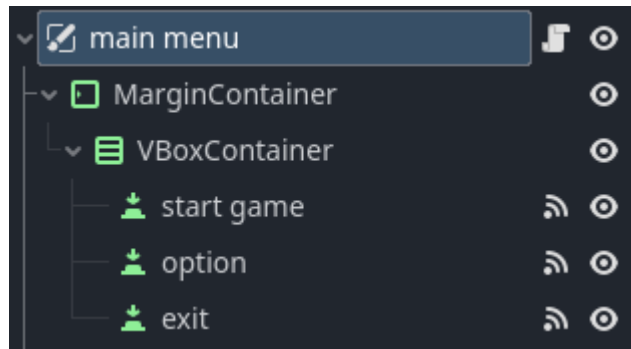


Рис 3.24 Додавання функціональних клавіш

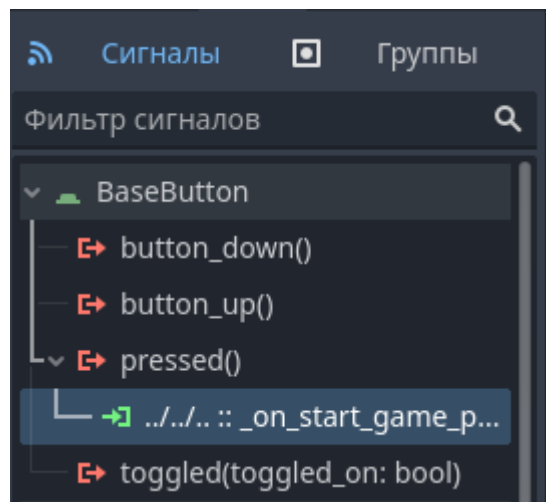


Рис 3.25 Прив'язка сигналу pressed() до клавіші

В скрипті функція перемикає поточну сцену на іншу сцену, розташовану обраним шляхом.

```

1  extends CanvasLayer
2
3  var SCROLL_SPEED_H := 10
4
5  func _on_start_game_pressed() -> void:
6      get_tree().change_scene_to_file("res://Free/world/world.tscn")
7
8  func _on_option_pressed() -> void:
9      get_tree().change_scene_to_file("res://main menu/options.tscn")
10
11 func _on_exit_pressed() -> void:
12     get_tree().quit()
13
14 func _process(delta: float) -> void:
15     $ParallaxBackground.scroll_offset.x -= SCROLL_SPEED_H * delta
16

```

Рис 3.26 Скрипт перемикання сцени

Головне меню готове, тепер можна переходити до створення рівнів, у фінальному варіанті головне меню виглядає так:



Рис 3.27 Фінальний вигляд головного меню

Створення рівнів

В Godot Engine є TileMap це вузол який використовується для створення 2D-рівнів з використанням тайлів (плиток).

Для початку створюємо нову сцену та додаємо до неї TileMap

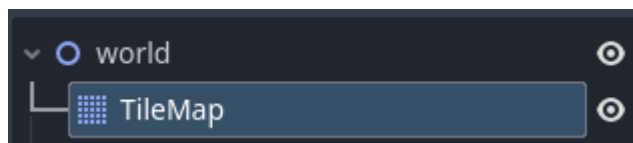


Рис 3.28 Вузол TileMap

Додаємо до TileMap завантажений або створений самостійно tileset

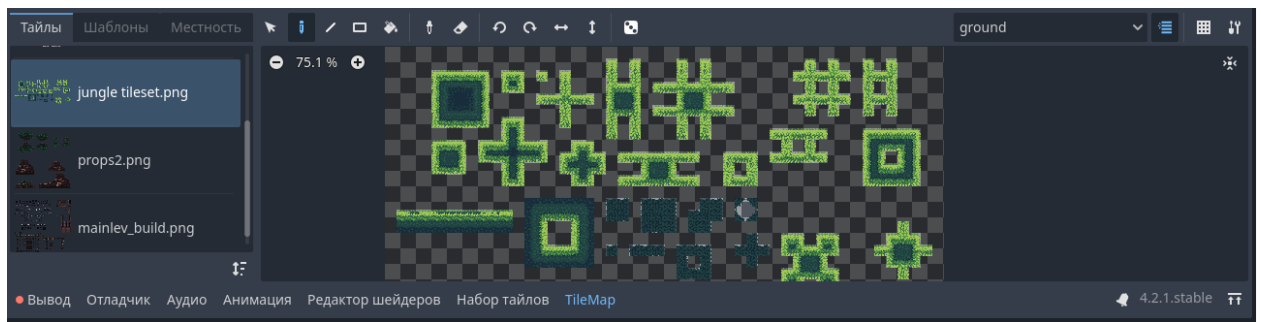


Рис 3.29 Tileset

Godot сам запропонує вам виділити непрозорі текстури тому натискаємо так і йдемо далі

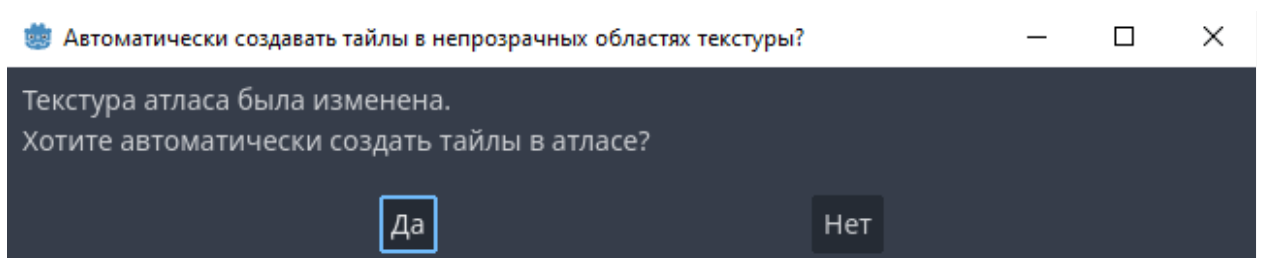


Рис 3.30 Автоматичне створення тайлів

Далі заходимо в налаштування тайлсету та обираємо фізичний шар, це треба для того щоб в майбутньому після створення персонажу наш герой взаємодівав з підлогою а не провалювався крізь неї

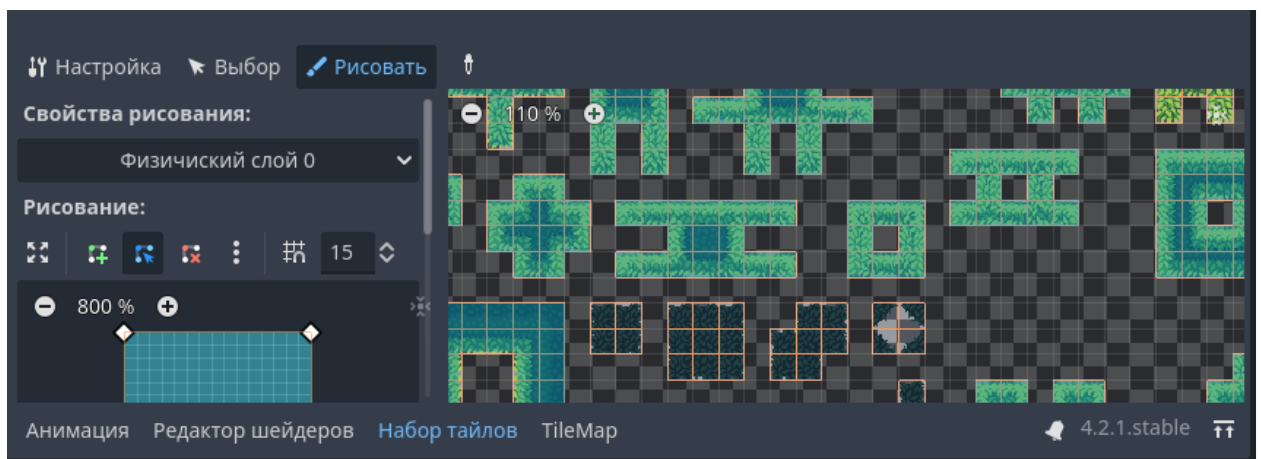


Рис 3.31 Налаштування тайлсету

Тепер можна намалювати декілька рівнів

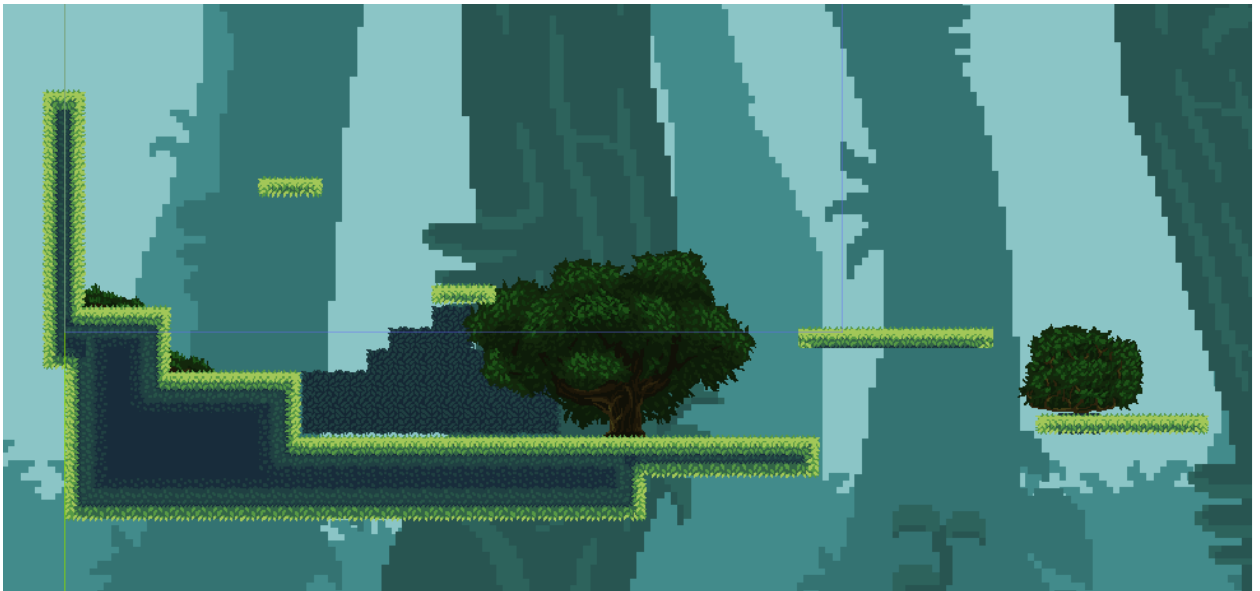


Рис 3.32

Також можна створити декілька різних пасток які в майбутньому будуть перешкоджати пересуванню нашого гравця, для цього створюємо нову сцену з нашою пасткою та додаємо вузол Area2D та дочірній вузол CollisionShape2D це потрібно для того щоб прив'язати входження персонажу в CollisionShape до скрипту.

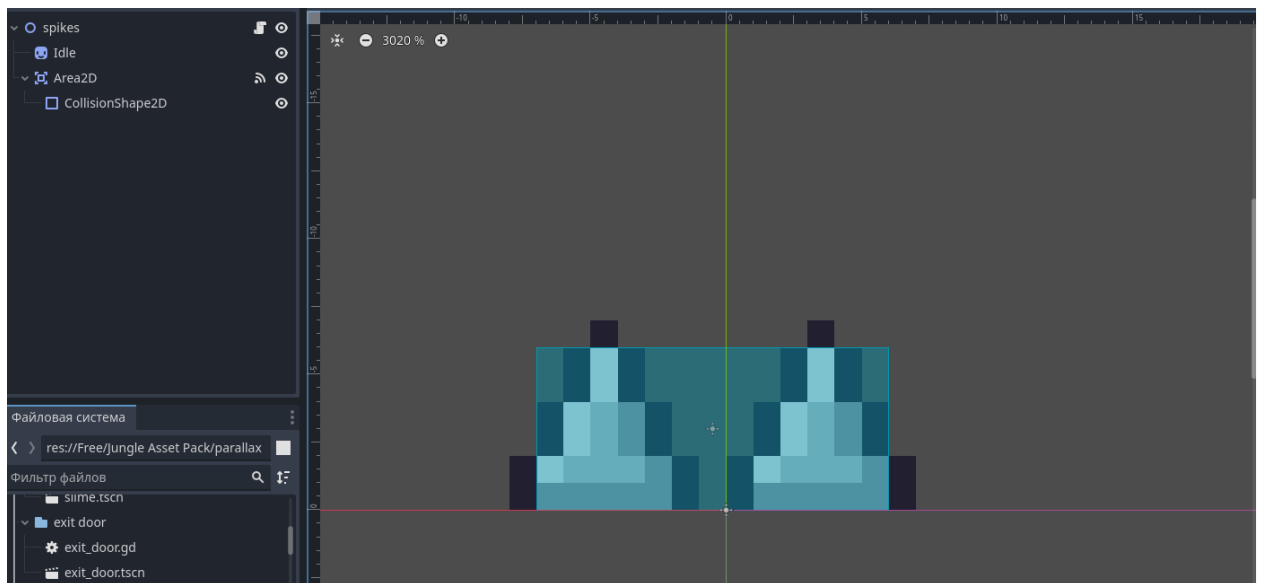


Рис 3.33 Межі пастки

```

1  extends Node2D
2
3
4  func _on_area_2d_body_entered(body: Node2D) -> void:
5      if body.name == "player":
6          GlobalVars.health -= 50
7

```

Рис 3.34 Код пастки

Створення персонажу

Для початку треба створити нову ноду з гравцем та додати CollisionShape2D це треба для того щоб наш персонаж міг взаємодіяти з оточенням.

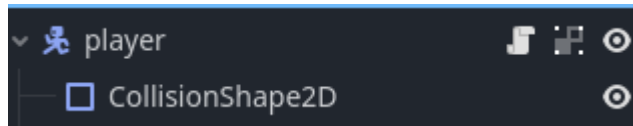


Рис 3.35 Вузол гравця

Надалі слід додати персонажу фізичні межі, оскільки зараз у нього бракує колізії і він може проходити крізь об'єкти. Для цього налаштовуємо CollisionShape під межі нашого персонажу таким чином, якими він має межувати з колізіями інших об'єктів у грі.



Рис 3.36 Межі гравця

Для додавання руху персонажу необхідно насамперед розробити скрипт, який буде додавати необхідні параметри та з їх допомогою контролювати рух персонажу і його взаємодію з іншими об'єктами. Цей код отримує введення від користувача (напрямок вліво або вправо) та встановлює швидкість гравця відповідно до цього напрямку. Якщо немає введення (напрямок дорівнює 0), швидкість плавно зменшується до нуля. Потім оновлюється анімація та виконується рух та зіткнення гравця з об'єктами

```

1 extends CharacterBody2D
2
3 @export var tilemap : TileMap
4
5 var SPEED = 100
6
7 func _process(delta):
8     var direction = Input.get_axis("ui_left", "ui_right")
9     if direction:
10         velocity.x = direction * SPEED
11     else:
12         velocity.x = move_toward(velocity.x, 0, SPEED)
13     update_animation()
14     move_and_slide()

```

Рис 3.37 Фрагмент коду який додає можливість рухатися

Також треба додати гравітацію, на щастя в Godot є гравітація за замовчуванням в налаштуваннях проекту, нам просто потрібно зробити змінну та присвоїти їй значення з налаштувань проекту а далі рушій все зробить за нас

```
var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")
```

Також потрібно додати нашому персонажу можливість стрибку а також подвійного стрибку для цього розробимо такий код. У цьому коді Перевіряється, чи гравець знаходиться на підлозі, і, якщо так, змінна `double_jump` встановлюється в `false`, щоб гравець міг знову виконати подвійний стрибок. Якщо гравець не знаходиться на підлозі, швидкість по вертикалі збільшується на величину гравітації, помножену на час між кадрами (`delta`). Перевіряється, чи була натиснута клавіша стрибка (`ui_assept`). Якщо так, та гравець знаходиться на підлозі, встановлюється можливість виконання подвійного стрибка, і гравець робить один звичайний стрибок. Якщо гравець не знаходиться на підлозі і він має можливість виконати подвійний стрибок, виконується подвійний стрибок. У цьому випадку відтворюється анімація "подвійного стрибка", а також гравець отримує додатковий стрибок зі зниженою силою.

```

var doble_jump = false
func _physics_process(delta):
    if is_on_floor():
        >| >| doble_jump = false
    if not is_on_floor():
        >| >| velocity.y += gravity * delta
    >|
    # Handle jump.
    if Input.is_action_just_pressed("ui_accept"):
        >| >| if is_on_floor():
        >| >| >| can_doble_jump = true
        >| >| >| velocity.y = JUMP_VELOCITY
        >| >| >|
        >| >| if not is_on_floor() and has_doble_jump and can_doble_jump:
        >| >| >| doble_jump = true
        >| >| >| anim.play("doble jump")
        >| >|
        >| >| >| can_doble_jump = false
        >| >| >| velocity.y = JUMP_VELOCITY * 0.95

```

Рис 3.38 Фрагмент коду який додає можливість стрибку

Тепер нам потрібно створити ворогів. Вороги створюються подібним способом, за винятком, що гравець ними управляти не може. Також відмінністю ворогів є те що вони стоять нерухомо допоки гравець не увійде в зазначену зону поблизу ворога, тоді наш противник «оживе» й почне переслідувати гравця

```

1  extends CharacterBody2D
2
3  @onready var anim: AnimatedSprite2D = $AnimatedSprite2D as AnimatedSprite2D
4  @export var points: int = 1
5  var alive = true
6  var SPEED = 50
7  var chase = false
8
9  var gravity: int = ProjectSettings.get_setting("physics/2d/default_gravity")
10
11 func _physics_process(delta: float) -> void:
12     if not is_on_floor():
13         velocity.y += gravity * delta
14     var player = $"../player"
15     var direction = (player.position - self.position).normalized()
16     if chase == true:
17         velocity.x = direction.x * SPEED
18
19     move_and_slide()
20     update_animation()
21
22 func update_animation():
23     if alive == true:
24         if velocity.x < 0:
25             anim.flip_h = false
26         elif velocity.x > 0:
27             anim.flip_h = true
28         if velocity.x:
29             anim.play("run")
30         else:
31             anim.play("idle")
32         if position.y > 480:
33             on_death()
34
35 func on_death():
36     alive = false

```

Рис 3.39 фрагмент коду ворога

Після виконаних дій з налаштування об'єктів і програмування скриптів, головний герой може рухатись, але поки без анімацій.

Додавання анімацій

Для 2D анімації в Godot є AnimatedSprite2D цей вузол послідовно змінює готові зображення з заданою частотою кадрів, додаємо цей вузол до нашого персонажу та вставляємо туди готові набори спрайтів.

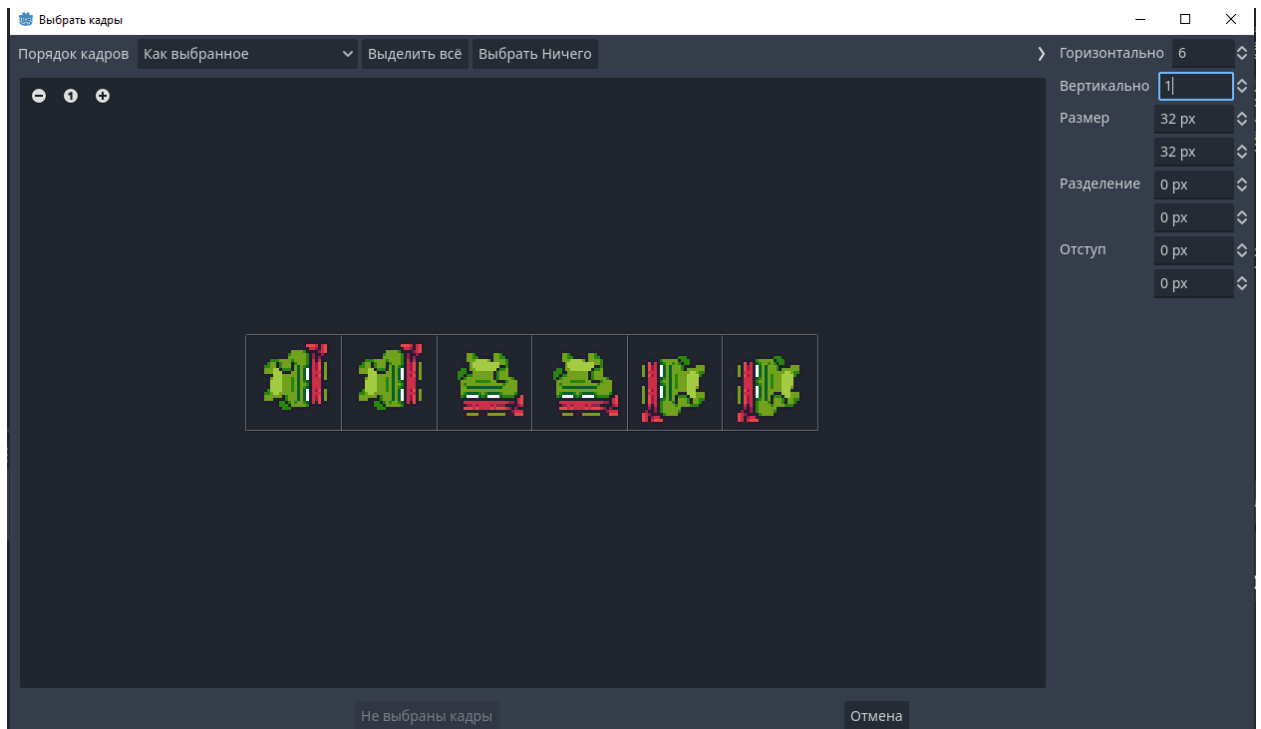


Рис 3.40 Готовий набір спрайтів в вузлі AnimatedSprite2D

Тепер прив'язуємо анімації до дій нашого персонажу а також до ворогів.

```

>| >| if double_jump == false:
>| >| >| if velocity.x:
>| >| >| anim.play("run")
>| >| >| else:
>| >| >| anim.play("idle")
>|
>| >| if velocity.y<0:
>| >| >| anim.play("jump")
>| >| >| elif velocity.y>0:
>| >| >| anim.play("fall")

```

Рис 3.41 Фрагмент коду який програє анімацію

Готово, за аналогією можна зробити ще декілька рівнів і проект можна вважати завершеним.

Перевірити працездатність проекту можна прямо у вікні Godot для цього необхідно натиснути кнопку запуску проекту у верхній частині екрана.

Поруч із нею знаходяться кнопки паузи та закінчення тесту гри. Вони мають такий самий вигляд, як кнопки управління в будь-якому іншому аудіо- або відеоплеєрі.

Вигляд запущеного проекту із вікна Godot представлений на рисунках.



Рис 3.42 Початок першого рівня

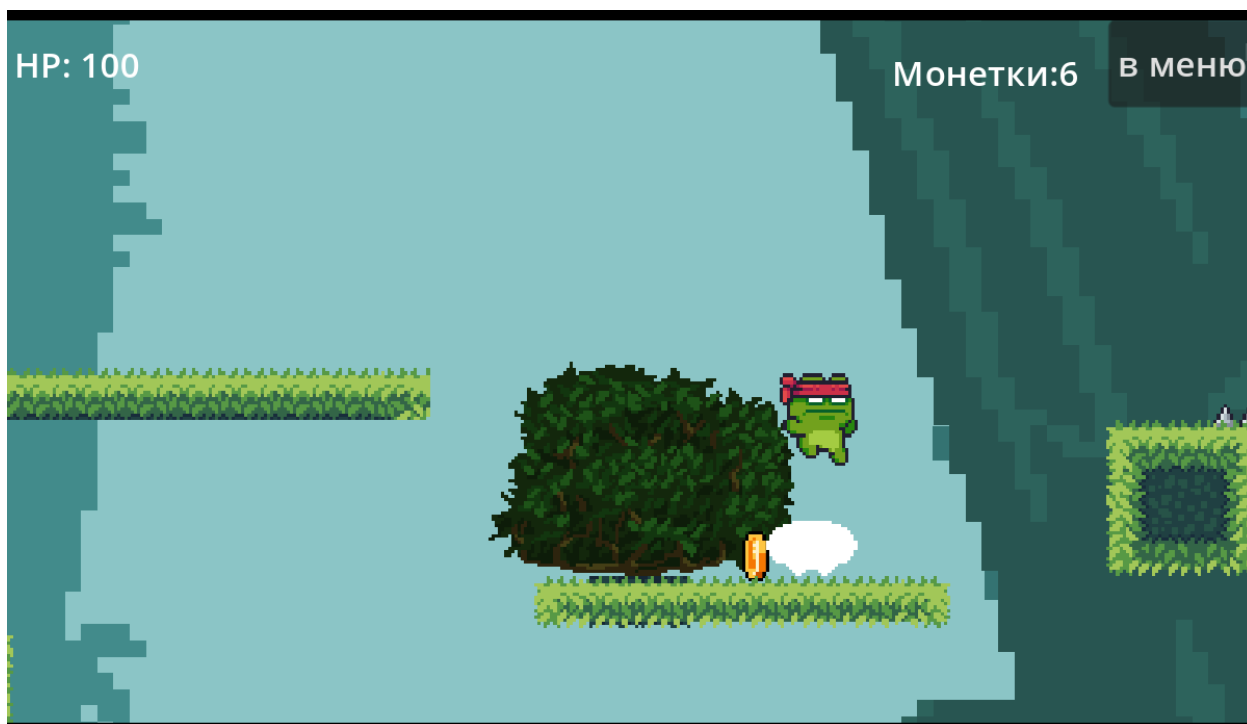


Рис 3.43 Перемога над ворогом



Рис 3.44 Перехід на наступний рівень

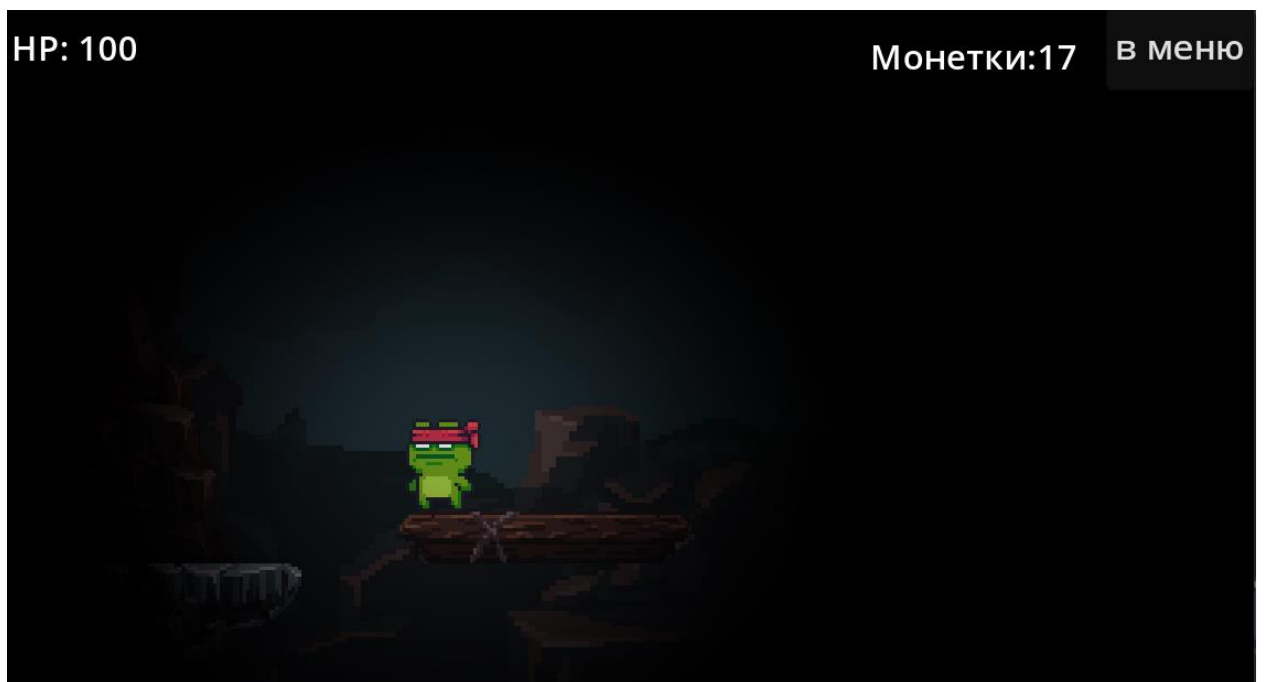


Рис 3.45 Рівень 2

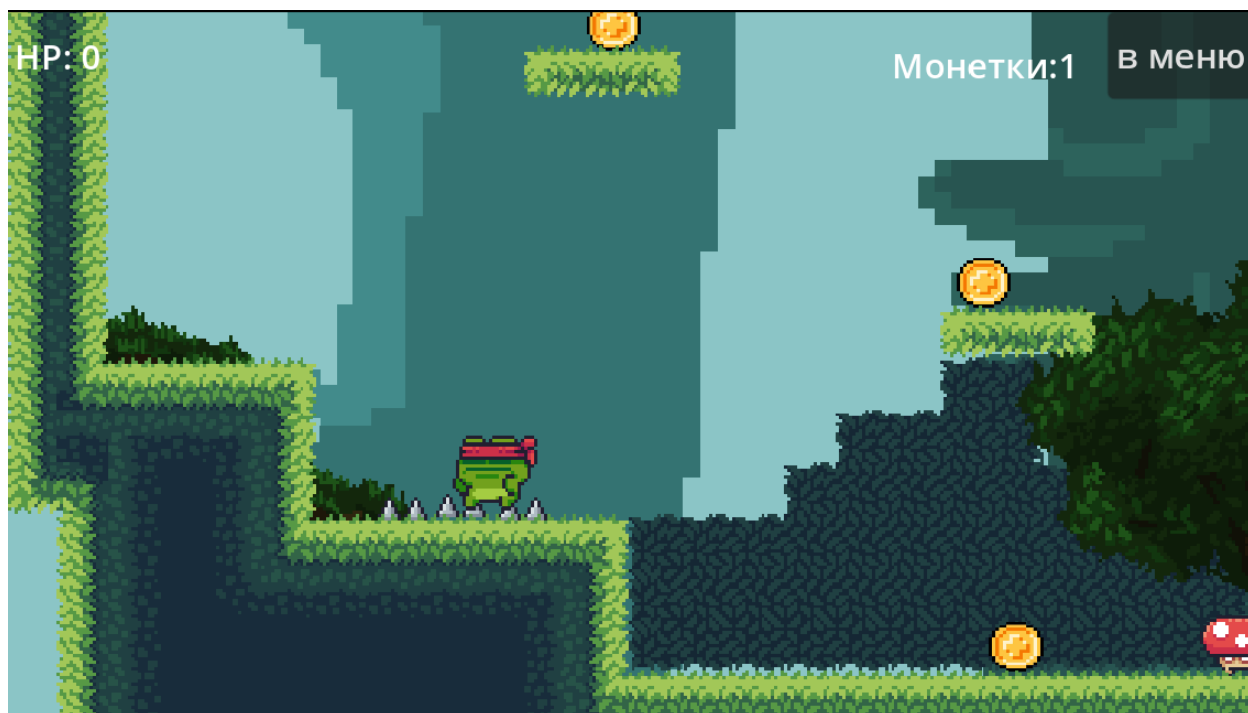


Рис 3.46 «смерть» персонажу

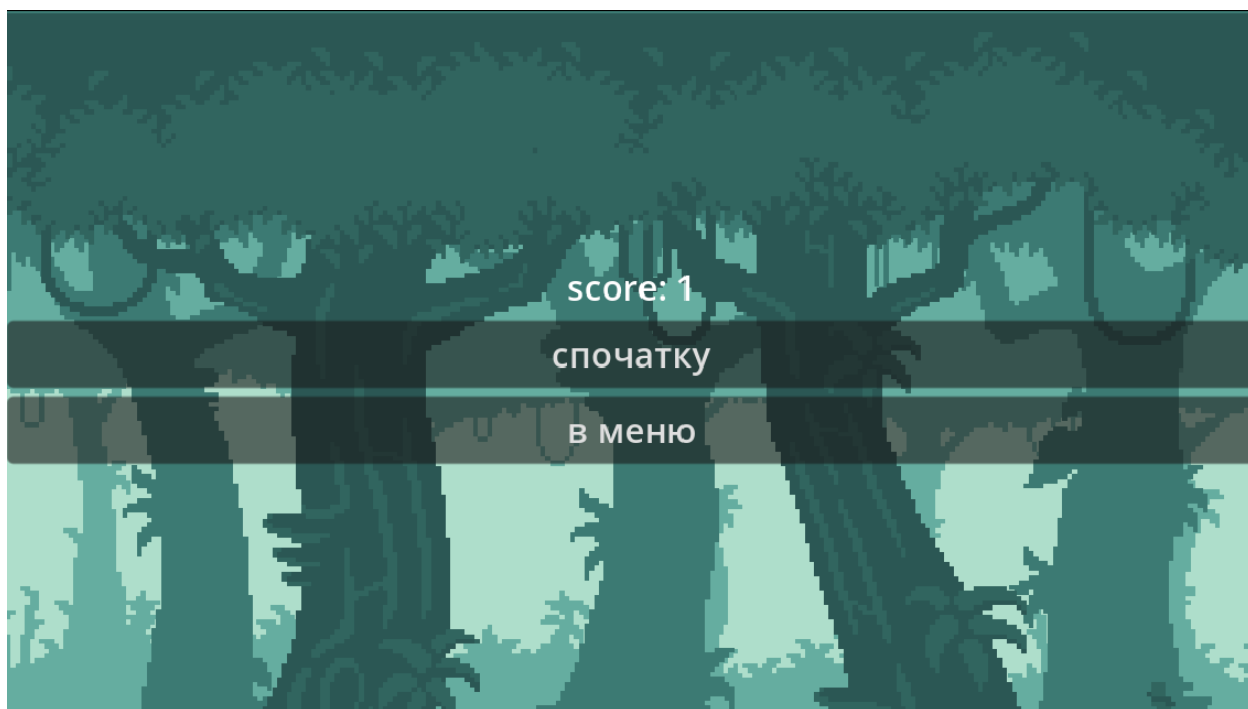


Рис 3.47 Экран «смерті»

Технічні характеристики

Перш за все, варто розуміти, що дані проект не є повністю готовою до релізу комп'ютерною грою, і за своєю суттю є прототипом, що стоїть приблизно на рівні ранньої альфа-версії. Викликано це було тим, що створення якісного продукту займає колосальну кількість часу, Так, наприклад, розробка невеликої гри для мобільних телефонів з рівнем, що процедурно генерується, може займати більше року, за умовою, що над проектом працює не більше трьох осіб. Збільшення кількості людей може прискорити розробку, але не сильно.

На даний момент дуже складно визначити які системні вимоги матиме проект у майбутньому, оскільки велика кількість елементів просто не реалізовано. На даний момент ми маємо лише елементи, які відповідають за основну ігрову механіку, а отже, у майбутньому технічні характеристики для комп'ютерної гри, що розробляється можуть сильно змінитися.

Зараз можна з упевненістю стверджувати лише те, що для запуску даного прототипу потрібно комп'ютер, що володіє системними характеристиками не нижче мінімально допустимих для коректної роботи ігрового рушія Godot, для якого найбільш важливим аспектом є тільки відеокарти. Вона повинна підтримувати DirectX 9 з шейдерами не нижче версії 3.0. Це означає, що в даний момент створений прототип з великою ймовірністю піде на більшості комп'ютерів.

ВИСНОВКИ

У роботі було вивчено особливості розробки відеоігор, їхній вплив на суспільство та зумовлений ними технологічний розвиток периферійних пристроїв і комп'ютерних систем.

Грунтуючись на отриманій в ході дослідження інформації, було вирішено розробити прототип двовимірного платформера для одного гравця

на ігровому рушії Godot. Таке рішення було прийнято за декількома причинами:

- 1) двовимірна графіка, на відміну від тривимірної легше у створенні;
- 2) з ігрової механіки, гра жанру платформер простіше реалізується;
- 3) ігровий рушій Godot поширюється безкоштовно і має широку документацію з розробки відеоігор

В процесі роботи з'ясовано, що індустрія розробки відеоігор має безліч складових, залучає дуже багато людей та професій, це виправдовується великим об'ємом і різним характером роботи. Для розробки гри, в ході дипломної роботи, довелося:

- працювати над концептом майбутнього продукту;
- складати поетапний план проєкта;
- створювати анімації;
- писати програмний код;
- досліджувати нові галузі та методи розробки.

У підсумку була розроблена демонстраційна версія комп'ютерної гри жанру платформер на базі ігрового рушія Unity, із застосуванням стилізованої двовимірної графіки стилю Pixel art.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Енциклопедія сучасної України [Електронний ресурс] посилання на ресурс <https://esu.com.ua/article-4393>
2. FutureNow Technologies & Science Blog [Електронний ресурс] посилання на ресурс <https://futurenow.com.ua/yak-stvoryuyutsya-videoigry-protses-rozrobky-igor/>
3. ELARTU [Електронний ресурс] посилання на ресурс https://elartu.tntu.edu.ua/bitstream/lib/34719/2/AZST_2020v2_Yatsyshyn_V_V-Analysis_of_game_engines_79.pdf
4. Вікіпедія [Електронний ресурс] посилання на ресурс https://uk.wikipedia.org/wiki/Графіка_у_відеоіграх
5. Всеосвіта [Електронний ресурс] посилання на ресурс <https://vseosvita.ua/library/embed/01002pcs-27a1.docx.html>
6. fugas space [Електронний ресурс] посилання на ресурс <https://blog.fugas.space/gamedev-stages/>
7. Godot Docs [Електронний ресурс] посилання на ресурс https://docs.godotengine.org/ru/4.x/getting_started/step_by_step/index.html
8. GitHub [Електронний ресурс] посилання на ресурс <https://github.com/godotengine/godot-docs>
9. YouTube [Відео ресурс] посилання на ресурс <https://www.youtube.com/@GODOTru>

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»
Факультет (інститут) Навчально-науковий інститут математики та
інформаційних технологій

(повна назва)
Кафедра Інформаційних технологій та систем

(повна назва)

Програма та методика тестування
на виконання програмної розробки (ПР):
Розробка додатку з використанням ігрового рушія godot

Полтава – 2024

ЗМІСТ

| | |
|--------------------------------------|----|
| ТЕСТ ПЛАН..... | 36 |
| 1.Вступ..... | 36 |
| 1.1.Мета..... | 36 |
| 1.2. Вхідні дані..... | 36 |
| 1.3. Мета тестування..... | 36 |
| 2.Умови тестування..... | 37 |
| 3.Стратегія процесу тестування..... | 37 |
| 3.1. Тестування інтерфейсу..... | 37 |
| 3.1.2. функціональне тестування..... | 38 |
| 4. План робіт..... | 38 |
| 5. Кінцеві результати..... | 38 |
| ТЕСТОВІ ВИПАДКИ | 39 |

ТЕСТ-ПЛАН

1. Вступ

1.1. Мета

Мета даного тест плану - це опис додатку створеного за допомогою ігрового рушія Godot. Документ дозволяє отримати інформацію про план тестових робіт.

1.2 Вхідні дані

Додаток “платформер”- написаний на ігровому рушії Godot.

1.3 Мета тестування

Метою перевірки додатку є перевірка коректної роботи всіх його елементів. Більшу частину тестування буде відведена тестування правильності реалізації ігрових механік.

Підсумком процесу тестування будуть наступні матеріали:

- Висновок відносно загального стану, що надасть розробнику цього продукту картину відносно роботи програми.
- Звіт про результати тестування.
- Задokumentовані помилки при виявленні.

Тестування буде проводитися вручну, методом "неформального" тестування (ad-hoc testing) з позиції кінцевого користувача програми.

Таким чином, буде протестовано як ігрові механіки, так і інтерфейс програми.

Комп'ютерна система, що затверджена до перевірки:

Персональний комп'ютер із наступними характеристиками:

- Процесор Intel(R) Core(TM) i3-2120 CPU з потужністю 3.30 GHz
- 4,00 ГБ оперативної пам'яті
- Тип системи: 64-розрядна операційна система
- Операційна система Windows 8.1 x64 професійна

2. Умови тестування

Програма повинна правильно контролювати дії користувача. Система додатку система повинна правильно відповідати на введення користувача.

3.Стратегія процесу тестування

В процесі тестування додатку на ігровому рушії Godot буде використано структуроване тестування це методика тестування програмного забезпечення, в якій тест-кейси та сценарії тестування розробляються заздалегідь на основі вимог до програми.

Коп'ютерна система і ПО, на якому буде проводитися тестування:

- 4,00 ГБ оперативної пам'яті
- ОС Windows 8.1 Професійна x64.
- Процесор Intel(R) Core(TM) i3-2120, 4 ядра по 3.30 GHz .

3.1. Тестування правильності ігрових механік

- Перевірка на коректне реагування персонажа на команди про пересування
- Перевірка на коректний стрибок
- Перевірка на коректну реалізацію події програшу гравця
- Перевірка на коректну роботу меню програшу
- Перевірка на коректну роботу головного меню
- Перевірка на коректну роботу меню налаштувань
- Перевірка на коректний збір предметів

4. План робіт

| Задача | Об'єм робіт | Дата початку | Дата закінчення |
|----------------------|-------------|--------------|-----------------|
| Укладання тест-плану | 12 годин | 01.05.2023 | 02.05.2023 |
| Виконання тестування | 12 години | 02.05.2023 | 03.05.2023 |
| Аналіз тестування | 6 годин | 03.05.2023 | 03.05.2023 |
| Підведення підсумків | 6 годин | 03.05.2020 | 04.05.2023 |

5. Кінцеві результати

5.1 Підсумок

Підсумком проведення тестування є оформлений додаток «Тест-план» (цей документ) та «Тестові випадки», які містять результат процесу тестування.

ТЕСТОВІ ВИПАДКИ

Тестування інтерфейсу

Тестова задача:

| № | Призначення | Дії | Очікуваний результат | Отриманий результат | Висновок |
|---|---|---|--|---|----------|
| 1 | Перевірка на коректне реагування персонажа на команди про пересування | Використовуючи клавіші руху, пробігти частину рівня | Персонаж буде рухатися в належному напрямі, коректне програвання анімації | персонаж пересувався в потрібному напрямку, коректне відображення анімації | Виконано |
| 2 | Перевірка на коректний стрибок | використовуючи клавішу стрибка, підстрибнути кілька разів поспіль | Гравець може зробити стрибок з землі а також лише 1 раз в повітрі, коректне програвання анімації | персонаж здатний здійснити стрибок з землі, а також лише 1 раз в повітрі вірне виконання анімації | Виконано |
| 3 | Перевірка на коректну реалізацію | Дозволити противнику перемогти гравця | Коли ворог атакує гравця, програється анімація смерті | ворог атакує гравця, програється | Виконано |

| | | | | | |
|---|--|--|---|--|--------------|
| | події програшу гравця | | персонажу, з'являється меню вибору дії | анімація смерті персонажу, з'являється меню вибору дії | |
| 4 | Перевірка на коректну роботу меню програшу | 1) Програти та перезапустити рівень 2) Програти та вийти в головне меню | є можливість перезапустити рівень або вийти в головне меню. | залежно від вибору перезапускаєт ься рівень, або виходить у головне меню | Викона но |
| 5 | Перевірка на коректну роботу головного меню | 1) Запустити рівень 2) Зайти в налаштування 4) Вийти з гри | є можливість запустити рівень, зайти в налаштування або вийти з гри | Залежно від вибору запускається рівень, переходить в меню налаштувань та закривається гра | Викона но |
| 6 | Перевірка на коректну роботу меню налаштува нь | Зайти в меню налаштувань та обрати складність | При виборі складності вибір залишається на всю гру допоки гравець | Складність змінюється та зберігається допоки гравець не змінить її назад | Викона но |

| | | | | | |
|---|---|----------------------|--|---|--------------|
| | | | самостійно не змінить складність або не вийде з гри | | |
| 7 | Перевірка на коректний збір предметів | Підібрати предмет | при зборі колекціонован их об'єктів, зібраний об'єкт знищується, рахунок збільшується на 1 поінт | при зборі об'єкта, він зникає, і рахунок збільшується на 1 поінт | Викона но |

Результати тестувань розробленої двовимірної гри для ПК на рушії Godot Engine, на ПК, показали, що додаток відповідає усім встановленим вимогам.

Програмні скрипти

Скрипт персонажу

```

extends CharacterBody2D
@export var tilemap : TileMap
var SPEED = 180
var JUMP_VELOCITY = -300
var doble_jump = false
var die = false
@onready var anim: AnimatedSprite2D = $AnimatedSprite2D

var has_doble_jump : bool = true
var can_doble_jump : bool = false

var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")

func _ready() -> void:
    var r = tilemap.get_used_rect()
    var vp = tilemap.get_viewport_rect()
    var qs = tilemap.cell_quadrant_size
    $Camera2D.limit_left = r.position.x * qs
    $Camera2D.limit_top = r.position.y * qs
    $Camera2D.limit_right = $Camera2D.limit_left + r.size.x * qs
    $Camera2D.limit_bottom = $Camera2D.limit_top + r.size.y * qs
    GlobalVars.score

func _physics_process(delta):
    if is_on_floor():
        doble_jump = false
    if not is_on_floor():
        velocity.y += gravity * delta

    if Input.is_action_just_pressed("ui_accept"):
        if is_on_floor():
            can_doble_jump = true
            velocity.y = JUMP_VELOCITY

        if not is_on_floor() and has_doble_jump and can_doble_jump:
            doble_jump = true
            anim.play("doble_jump")

            can_doble_jump = false
            velocity.y = JUMP_VELOCITY * 0.95

    var direction = Input.get_axis("ui_left", "ui_right")
    if direction:
        velocity.x = direction * SPEED
    else:
        velocity.x = move_toward(velocity.x, 0, SPEED)
    update_animation()
    move_and_slide()

    if position.y > 480:
        on_death()

```

```

func update_animation():
    if die == false:
        if velocity.x < 0:
            anim.flip_h = true
        elif velocity.x > 0:
            anim.flip_h = false

        if doble_jump == false:
            if velocity.x:
                anim.play("run")
            else:
                anim.play("idle")

            if velocity.y < 0:
                anim.play("jump")
            elif velocity.y > 0:
                anim.play("fall")

        if GlobalVars.health <= 0:
            on_death()

func on_death():
    die = true
    anim.play("hit")
    #await anim.animation_finished
    await get_tree().create_timer(0.7).timeout
    self.queue_free()
    get_tree().change_scene_to_file("res://main menu/game_over.tscn")

func _on_pickup_area_entered(area: Area2D) -> void:
    if area.has_method("on_pickup"):
        area.on_pickup(self)

```

Скрипт врага

```

extends CharacterBody2D

@onready var anim: AnimatedSprite2D = $AnimatedSprite2D as AnimatedSprite2D
@export var points: int = 1
var alive = true
var SPEED = 50
var chase = false
var gravity: int = ProjectSettings.get_setting("physics/2d/default_gravity")

func _physics_process(delta: float) -> void:
    if not is_on_floor():
        velocity.y += gravity * delta
    var player = $"../player"
    var direction = (player.position - self.position).normalized()
    if chase == true:
        velocity.x = direction.x * SPEED

    move_and_slide()
    update_animation()

func update_animation():
    if alive == true:
        if velocity.x < 0:
            anim.flip_h = false

```

```

        elif velocity.x>0:
            anim.flip_h = true

    if velocity.x:
        anim.play("run")
    else:
        anim.play("idle")

    if position.y > 480:
        on_death()

func on_death():
    alive = false
    anim.play("death")
    GlobalVars.score += 1
    await anim.animation_finished
    self.queue_free()

func _on_area_body_entered(body: Node2D) -> void:

    if body.name == "player":
        if alive == true:
            GlobalVars.health -= 50
        if GlobalVars.difficulty == true:
            GlobalVars.health -= 50
        on_death()

func _on_detector_body_entered(body: Node2D) -> void:
    if body.name == "player":
        chase = true

func _on_area_2d_2_body_entered(body: Node2D) -> void:
    if body.name == "player":
        if body.velocity.y>0:
            body.velocity.y -= 250
        if body.velocity.y<0:
            body.velocity.y += 250
        $sound.play()
        await $sound.finished
        on_death()

```

Скрипт головного меню

```

extends CanvasLayer

var SCROLL_SPEED_H := 10

func _on_start_game_pressed() -> void:
    GlobalVars.score = 0
    GlobalVars.health = 100
    get_tree().change_scene_to_file("res://Free/world/world.tscn")

func _on_option_pressed() -> void:
    get_tree().change_scene_to_file("res://main menu/options.tscn")

func _on_exit_pressed() -> void:
    get_tree().quit()

func _process(delta: float) -> void:
    $ParallaxBackground.scroll_offset.x -= SCROLL_SPEED_H * delta

```

Скрипт налаштувань

```
extends CanvasLayer
var SCROLL_SPEED_H := 10
```

```
func _ready() -> void:
    if GlobalVars.hard == false:
        $MarginContainer/VBoxContainer/normal/normal1.visible = true
        $MarginContainer/VBoxContainer/hard/hard1.visible = false
    if GlobalVars.hard == true:
        $MarginContainer/VBoxContainer/hard/hard1.visible = true
        $MarginContainer/VBoxContainer/normal/normal1.visible = false
```

```
func _on_normal_pressed() -> void:
    GlobalVars.difficulty = false
    GlobalVars.hard = false
    if GlobalVars.hard == false:
        $MarginContainer/VBoxContainer/normal/normal1.visible = true
        $MarginContainer/VBoxContainer/hard/hard1.visible = false
```

```
func _on_hard_pressed() -> void:
    GlobalVars.difficulty = true
    GlobalVars.hard = true
    if GlobalVars.hard == true:
        $MarginContainer/VBoxContainer/hard/hard1.visible = true
        $MarginContainer/VBoxContainer/normal/normal1.visible = false
```

```
func _on_back_pressed() -> void:
    get_tree().change_scene_to_file("res://main menu/main_menu.tscn")
```

```
func _process(delta: float) -> void:
    $ParallaxBackground.scroll_offset.x -= SCROLL_SPEED_H * delta
```

Скрипт фінального босу

```
extends CharacterBody2D
```

```
@onready var anim: AnimatedSprite2D = $AnimatedSprite2D as AnimatedSprite2D
@onready var animation: AnimationPlayer = $AnimationPlayer as AnimationPlayer
@export var points: int = 1
var alive = true
var SPEED = 20
var chase = false
var hp = 100
var death = false
var gravity: int = ProjectSettings.get_setting("physics/2d/default_gravity")
```

```
func _physics_process(delta: float) -> void:
    if not is_on_floor():
        velocity.y += gravity * delta
    var player = $"../player"
    var direction = (player.position - self.position).normalized()
    if chase == true:
        velocity.x = direction.x * SPEED
```

```
    move_and_slide()
    update_animation()
```

func **update_animation**():

```
    if alive == true:
        if death == false:
            if velocity.x < 0:
                anim.flip_h = false
            elif velocity.x > 0:
                anim.flip_h = true

        if velocity.x:
            anim.play("run")
        else:
            anim.play("idle")
        if position.y > 480:
            on_death()
```

func **on_death**():

```
    alive = false
    death = true
    anim.play("death")
    GlobalVars.score += 1
    await anim.animation_finished
    self.queue_free()
```

func **_on_area_body_entered**(body: Node2D) -> **void**:

```
    if body.name == "player":
        if alive == true:
            GlobalVars.health -= 20
        if GlobalVars.difficulty == true:
            GlobalVars.health -= 30
```

func **_on_detector_body_entered**(body: Node2D) -> **void**:

```
    if body.name == "player":
        chase = true
```

func **_on_area_2d_2_body_entered**(body: Node2D) -> **void**:

```
    if body.name == "player":
        hp -= 20
        if hp <= 0:
            on_death()
        if body.velocity.y > 0:
            body.velocity.y -= 250
        if body.velocity.y == 0:
            body.velocity.y -= 250
        $sound.play()
        await $sound.finished
        alive = false
        $Area2D2/CollisionShape2D2.disabled = true
        if death == false:
            anim.play("take hit")
            await anim.animation_finished
        alive = true
        $Area2D2/CollisionShape2D2.disabled = false
```

func **_on_hit_2_body_entered**(body: Node2D) -> **void**:

```
    if body.name == "player":
        if death == false:
            alive = false
            SPEED = 0
            animation.play("hit")
```

```

        await anim.animation_finished
        SPEED = 20
        $"attack direction/hit2/CollisionShape2D".disabled= true
        alive = true
        await get_tree().create_timer(1).timeout
        $"attack direction/hit2/CollisionShape2D".disabled = false

func _on_area_2d_3_body_entered(body: Node2D) -> void:
    if body.name == "player":
        if death == false:
            alive = false
            SPEED = 0
            anim.flip_h = true
            animation.play("hit")
            await anim.animation_finished
            SPEED = 20
            $"attack direction/hit2/CollisionShape2D".disabled= true
            alive = true
            await get_tree().create_timer(1).timeout
            $"attack direction/hit2/CollisionShape2D".disabled = false

func _on_area_2d_body_shape_entered(body_rid: RID, body: Node2D, body_shape_index: int, local_shape_index:
int) -> void:
    if body.name == "player":
        if body.velocity.y>0:
            body.velocity.y -= 250
        if body.velocity.y==0:
            body.velocity.y -= 250
        GlobalVars.health -= 30
        if GlobalVars.difficulty == true:
            GlobalVars.health -= 70

```

Скрипт зілля лікування

```

extends Node2D
var is_picked : bool = false
func _process(delta: float) -> void:
    pass
func _on_area_2d_body_entered(body: Node2D) -> void:
    if is_picked:
        return
    is_picked = true
    var tween1 = get_tree().create_tween().set_parallel(true)
    tween1.tween_property($".", "position:y", position.y - 20, 0.4)
    tween1.tween_property($AnimatedSprite2D, "self_modulate:a", 0.0, 0.4)
    GlobalVars.health +=50
    $sound.play()
    await $sound.finished
    await tween1.finished
    get_tree().queue_delete(self)

```

