СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра інформаційних технологій та програмування

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
спеціальність 126 „Інформаційні системи та технології”
(шифр і назва спеціальності)

на тему „Розробка відеогри платформеру на ігровому движку Godot”

Виконав: студент групи ІСТ-20д _____ (підпис) Д. В. Євдокимов (ініціали і прізвище)

Керівник _____ (підпис) В.О. Лифар (ініціали і прізвище)

Завідувач кафедри _____ (підпис) О.І. Захожай (ініціали і прізвище)

Рецензент___Д.В. Ратов_____

Київ – 2024

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра інформаційних технологій та програмування
Освітній ступінь бакалавр
спеціальність 126 „Інформаційні системи та технології”
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Захожай О.І.
‘___” _____ 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ
Євдокимов Данило Віталійович
(прізвище, ім’я, по батькові)

1.Тема Роботи: Розробка відеогри платформеру на ігровому движку Godot

Керівник роботи: Лифар Володимир Олексійович, д.т.н., професор
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від “_06_” _травня_2024 року №171/15.15-С

2. Строк подання студентом роботи: 08.06.2024р.

3. Вихідні дані до роботи: Об’єктом даної роботи є процес розробки відеогри на ігровому движку Godot

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Вступ Аналітичний огляд, історії, можливостей, переваг та недоліків движку Godot, проектування концепції, геймдизайну, стилю графіки, розробка сцен, персонажів, логіки, анімації і тестування, Висновки. Перелік використаних джерел.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 30.03.2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Одержання завдання на виконання роботи	30.03.24	виконано
2	Укладання і погодження з керівником плану і етапів виконання роботи	06.04.24	виконано
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	13.04.24	виконано
4	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху	20.04.24	виконано
5	Укладання та тестування програмного продукту	27.04.24	виконано
6	Укладання, оформлення та погодження пояснювальної записки з керівником	04.05.24	виконано
7	Здача готової пояснювальної записки на кафедру	08.06.24	виконано
8	Укладання доповіді і презентації	10.06.24	виконано

Студент _____ Д.В. Євдокимов
підпис (ініціали і прізвище)

Керівник роботи _____ В.О. Лифар
підпис (ініціали і прізвище)

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

дипломної роботи студента гр. ІСТ-20д Євдокимов Д.В.

Науковий керівник:

Професор, д.т.н. _____ Лифар В.О.

Оцінка наукового керівника: _____

Рецензент Ратов Д.В.
ПІБ, місто роботи, посада

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту:

Голова ЕК
Професор кафедри ІТП
д.т.н. _____ Меняйленко О.С.
підпис

РЕФЕРАТ

Робота містить: 45 сторінок основного тексту, 36 рисунків, 10 використаних джерел.

Метою: випускної кваліфікаційної роботи є вивчення особливостей роботи ігрових движків та розробки відеоігор, методів їх реалізації, оптимальних рішень, в плані програмування і підбору продуктивних інструментів.

Було проаналізовано багато інструментів розробки ігор, ігрових движків, їх принцип роботи, перелік наданих можливостей і концепцій.

Багато часу було приділено технічній частини, а саме анімації та машині станів.

В результаті виконаної роботи, було реалізовано відеогру платформер, яка є гарною основою для подальшого розвитку та вдосконалення.

Відеогра реалізована відповідно всім вимогам технічного завдання.

Зроблено детальний опис процесу розробки відеогри, а також, продемонстрована її робота.

ЗМІСТ

ВСТУП

Актуальність теми:

Що стосується теми розробки відеоігор, з кількох основних причин, включаючи технології, фінанси, культуру та освіту, розробка відеоігор є надзвичайно важливою та актуальною темою. Ось детальний опис цих елементів:

Технологічний компонент, Інновації та технологічний розвиток: швидкий розвиток технологій: відеоігри є одним із найважливіших джерел прогресу в технологіях. Графічні процесори, штучний інтелект, обчислювальні технології та інтерактивні системи розвиваються завдяки їхньому впливу. Взаємодія з новими технологіями: Використання віртуальної реальності (VR), доповненої реальності (AR) та інших новітніх технологій часто є частиною процесу розробки ігор, що сприяє їх впровадженню в різні сфери життя.

Інструменти для розробки: доступність та потужність ігрових двигунів: навіть інді-розробники можуть зараз використовувати сучасні інструменти розробки, такі як Godot, Unity та Unreal Engine, що дозволяє створювати високоякісні ігри з меншими витратами.

Ком'юніті та підтримка: великі спільноти розробників діляться інформацією та ресурсами, що прискорює процес навчання та розробки.

Економічний підхід:

Зростання ринку відеоігор; вплив на економіку; відеоігровий ринок є однією з найбільш швидкозростаючих галузей розваг, і його прибутки перевищують прибутки кіноіндустрії та музичної індустрії, якщо взяти всі інші.

Робота: Оскільки індустрія відеоігор розвивається, виникає багато робочих місць у різних областях, таких як програмування, дизайн, звукорежисура, менеджмент і маркетинг.

Інді-розробка: Підтримка інновацій: Інді-розробники часто приносять інноваційні ідеї та нові методи створення ігор, що збагачує ринок.

Фінансові можливості: збільшення популярності цифрових платформ дистрибуції ігор, таких як Steam, Itch.io та мобільні магазини, створює нові можливості для монетизації та розвитку інді-студій.

Культурна характеристика Вплив на культуру: Ігри як феномен культури: відеоігри стали важливою частиною сучасної культури, впливаючи на кіно, літературу, мистецтво та інші форми медіа.

Соціальні аспекти: Відеоігри часто об’єднують людей з різних куточків світу, створюючи світові спільноти гравців.

Освіта та виховання: Ігрові ігри: Відеоігри все частіше використовуються як інструмент для навчання, допомагаючи вдосконалювати різні навички, включаючи креативне мислення та логіку.

Розвиток м’яких навичок: Ігри покращують командну роботу, спілкування та емоційний інтелект.

Ступінь освіт, участь у навчанні: багато навчальних закладів пропонують курси з розробки відеоігор, щоб підготувати спеціалістів для індустрії.

Пропозиції для студентів: Студенти можуть отримати практичні навички, які є надзвичайно цінними на ринку праці, створюючи відеоігри як дипломні проекти.

Доступ до інформації: онлайн ресурси: Завдяки наявності онлайн-курсів, туторіалів, форумів та інших навчальних ресурсів будь-хто, хто бажає навчитися створювати відеоігри, незалежно від того, де вони знаходяться та скільки грошей вони мають.

Багато причин роблять розробку відеоігор актуальною. Вона поєднує в собі новітні технології, значний економічний вплив, культурний вплив і освітні можливості. Відеоігри стають все більш важливою частиною нашого життя, оскільки вони пропонують нові можливості для творчості, навчання та розвитку професійних навичок.

Мета та Задачі дослідження:

Робота спрямована на комплексне дослідження процесу розробки відеогри. Вони охоплюють весь цикл розробки, починаючи від початкового аналізу та проектування та закінчуючи тестуванням і документацією. Завдяки цьому можна отримати глибоке розуміння специфіки розробки відеоігор, а також створити якісний кінцевий продукт. Обидва ці знання будуть важливими для майбутньої роботи професійних у цій галузі.

Метою: цієї дипломної роботи є створення платформерної відеогри на основі ігрового движка Godot. Ця гра буде демонструвати сучасні підходи до створення двовимірних ігор, такі як дизайн ігрових рівнів, механіка руху персонажів, інтерактивність та анімація. Дослідження буде вивчати процес розробки гри від ідеї до завершеної версії, зосереджуючись на можливостях движка Godot.

Задачі роботи:

Аналіз платформи Godot та його можливостей: - проаналізувати його історію та розвиток.

Дослідити основні функції та можливості движка, включаючи його модулі та інструменти.

Аналізувати переваги та недоліки движку Godot.

Спроектувати архітектуру гри, включаючи структуру ігрових сцен, класів і об'єктів.

Визначити необхідні ресурси для гри.

Розробка основних механік гри, таких як рух персонажа, стрибки, взаємодія з об’єктами.

Використовувати вбудовані можливості Godot для реалізації фізики гри.

Розробити анімаційні переходи та анімацію персонажів і об’єктів.

Створення та налаштування ігрового рівня.

Виявлення та усунення багів, підвищення продуктивності та зручності

використання.

Документування процесу розробки: підготувати докладну інформацію, що описує кожен етап розробки гри.

- Описати проблеми та труднощі, з якими зіткнулися під час розробки, а також про те, як вони були вирішені.

Огляд літератури:

Огляд літератури дозволяє зрозуміти поточний стан досліджень у галузі розробки відеоігор, зокрема на ігровому движку Godot. Крім того, він допомагає визначити основні концепції, методології та інструменти.

Книги та статті: Tracy Fullerton's "Game Design Workshop: A Playcentric Approach to Creating Innovative Games": Ця книга дає повне розуміння процесу розробки ігрового дизайну, починаючи від концепції та закінчуючи прототипуванням і тестуванням.

Katie Salen and Eric Zimmerman's "Rules of Play: Game Design Fundamentals" розглядає основи ігрового дизайну, включаючи теорію ігрових систем, гравців і взаємодію.

Chris Bradfield написав книгу "Godot Engine Game Development Projects", який містить практичний посібник з розробки різноманітних ігор на Godot, включаючи платформери.

Alan Thorn написав «GD Script: Godot 4.0 Beginner's Guide», яка допомагає початківцям навчитися використовувати GDScript, основну мову програмування Godot.

Офіційна документація Godot доступна за адресою https://docs.godotengine.org/. Там можна знайти докладний опис кожної функції та можливості движка, приклади використання GDScript і туторіали для створення ігор.

GDQuest — це онлайн-ресурс, який містить багато відеоуроків і туторіалами з різних частин розробки Godot, включаючи створення платформерів.

"Level Up! The Guide to Great Video Game Design" by Scott Rogers: Книга пропонує практичні поради та методи створення захоплюючих і добре спроектованих ігрових рівнів.

"The Art of Game Design: A Book of Lenses" by Jesse Schell обговорює ігровий дизайн з різних точок зору, що допомагає створювати захоплюючі та добре збалансовані ігрові світи.

"Game Testing: All in One" by Charles P. Schultz: Посібник з методів і інструментів тестування ігор на всіх етапах розробки.

"Game Engine Architecture" by Jason Gregory: охоплює архітектурні аспекти розробки ігрових движків і оптимізації ігор.

Одним із найпопулярніших інструментів для розробки ігор у двох і трьох димах є ігровий движок Godot. У ньому багато функцій, які дозволяють створювати ігри різного рівня складності, а також безкоштовність і відкритий вихідний код.

1 ОГЛЯД ІГРОВОГО ДВИЖКУ GODOT

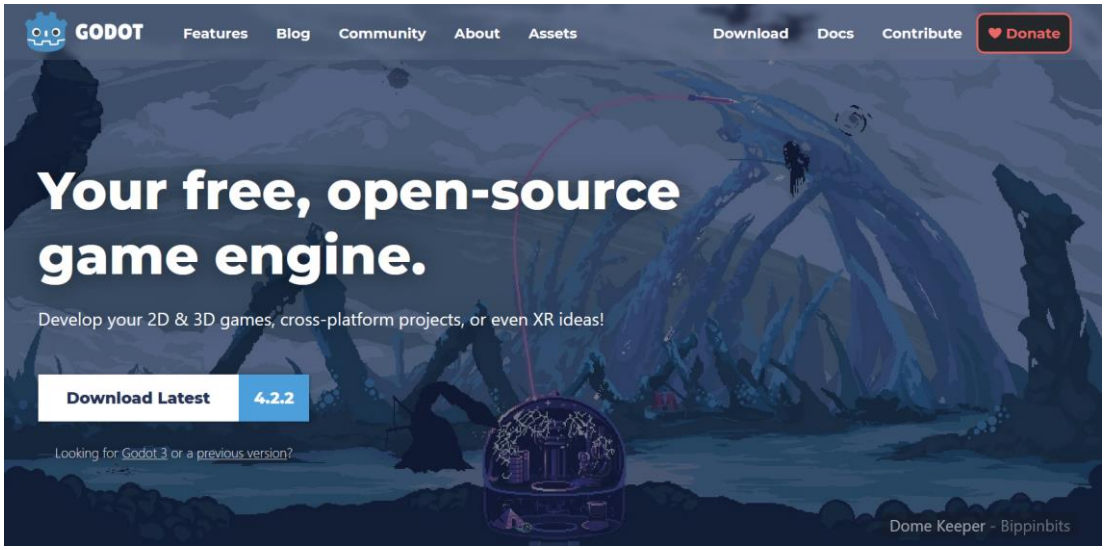


Рисунок 1.1 Godot Engine.

Одним із найпопулярніших інструментів для розробки ігор у двох і трьох димах є ігровий движок Godot. У ньому багато функцій, які дозволяють створювати ігри різного рівня складності, а також безкоштовність і відкритий вихідний код.

1.1 Історія та розвиток Godot

Від свого заснування в 2007 році до сьогодні він еволюціонував завдяки підтримці розробників і спільноти. Детально розглянемо основні віхи розвитку Godot.

Заснування та початковий розвиток (2007-2013):

У 2007 році, розробка Godot розпочалася Аріелем Маніелем (Ariel Manzur) та Хуаном Ліньєцці (Juan Linietsky). Спочатку движок був внутрішнім інструментом для компанії, в якої вони працювали. Він не мав назви і був розроблений як інструмент для швидкого створення продуктів для клієнтів.

З 2008-2013 років движок використовувався виключно для внутрішніх проектів і був закритий для широкого загалу. За цей час він набув багато функцій, які були використані для створення комерційних ігор, включаючи підтримку 2D та 3D графіки, а також вбудовану систему скриптування.

Відкритий вихідний код і публічний реліз:

Січень 2014 року: Godot був офіційно випущений як проект з відкритим вихідним кодом на GitHub під ліцензією MIT. Це було важливим кроком, оскільки це дозволило розробникам з усього світу використовувати, змінювати та покращувати движок. Версія 1.0 була номером першого публічного релізу.

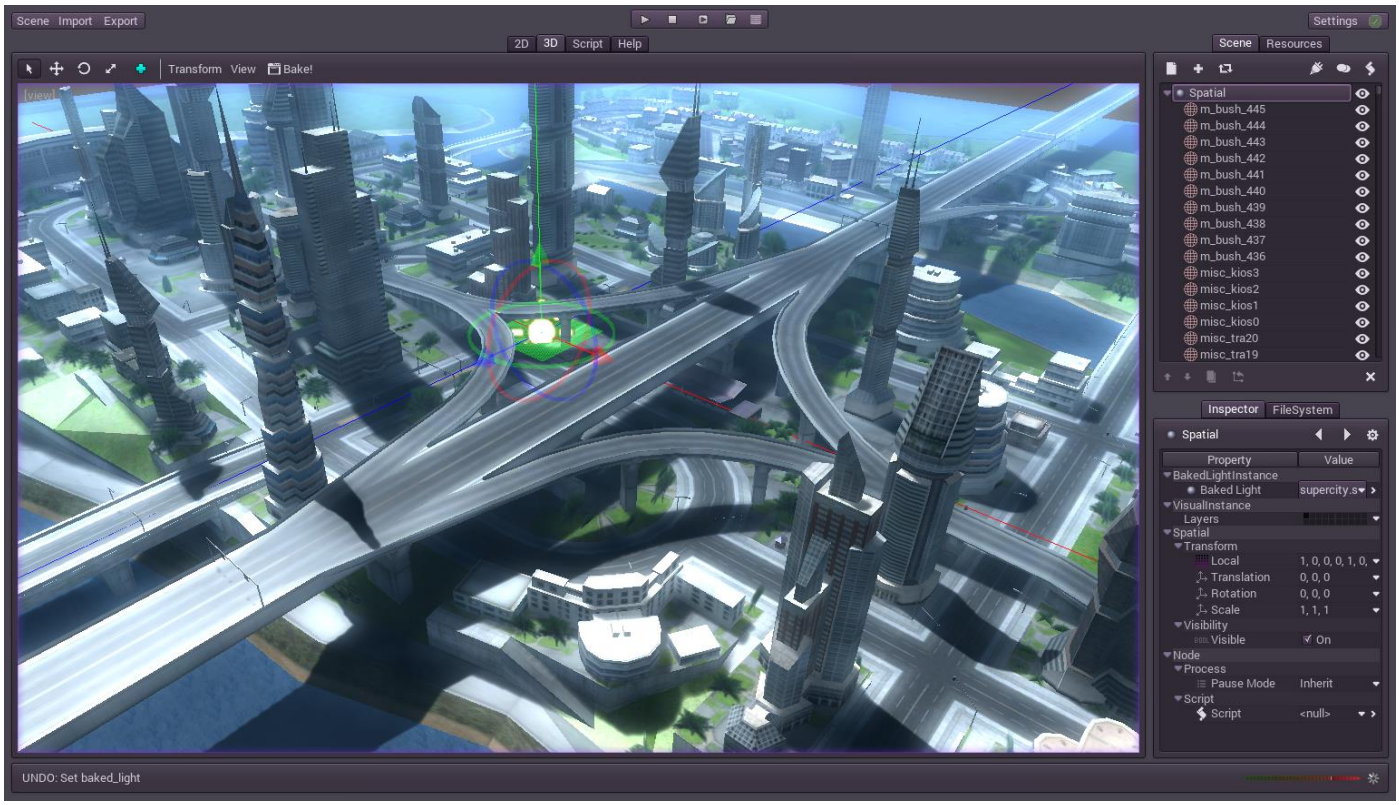


Рисунок 2.1 Godot Engine версії 1.0.

Версія 1.0 включала основні функції розробки 2D та 3D, а також підтримку GDScript (власної мови скриптування Godot), систему нодів і редактор сцени. Це була досить стійка платформа, яка містила всі необхідні інструменти для створення ігор.

Удосконалення та розвиток (2014–2017):

Версія 1.1 (2015) включає нові інструменти для редагування, підвищення продуктивності та нові функції для розробки 2D та 3D. Значним доповненням стала система анімації.

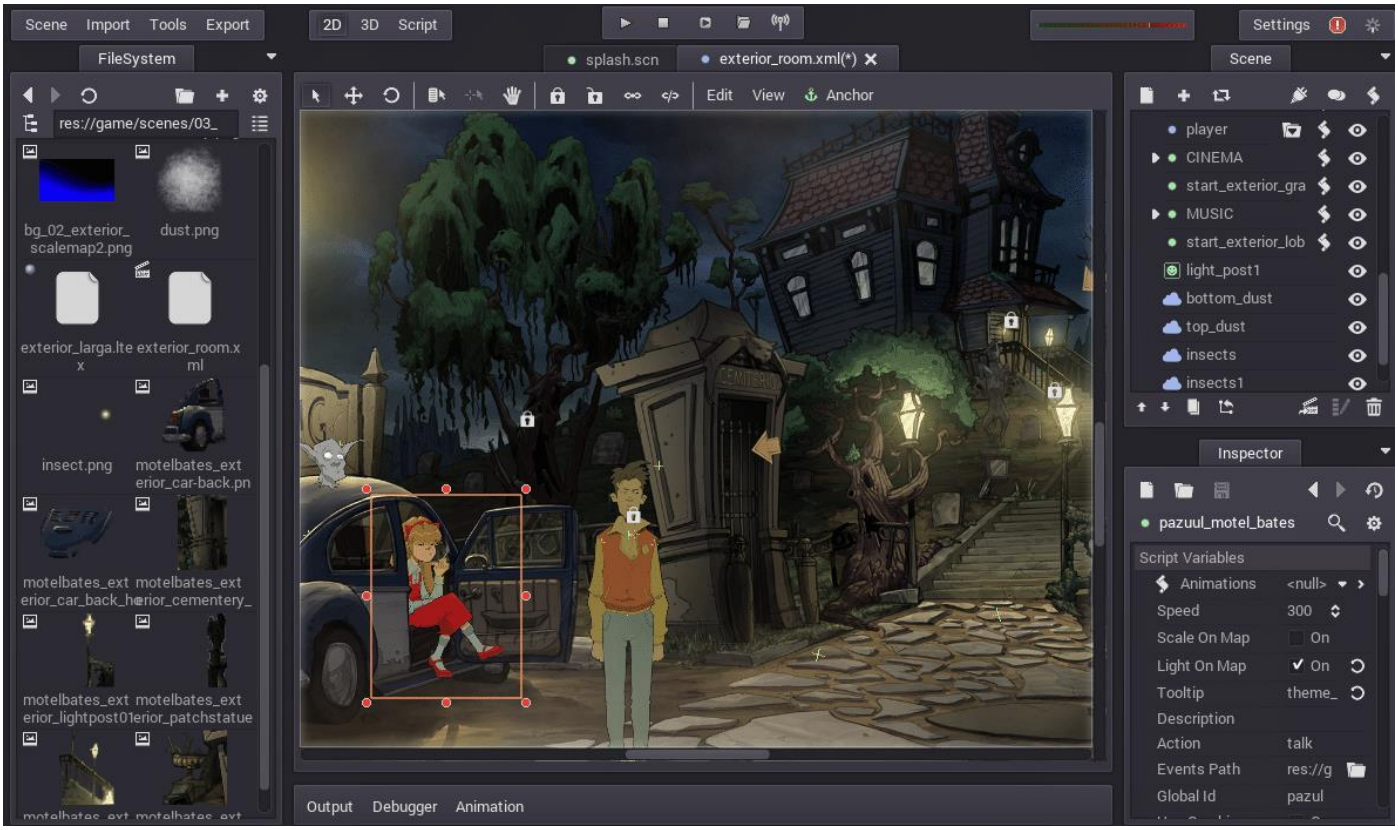


Рисунок 2.2 Godot Engine версії 2.0.

Версія 2.0 (2016): Функціонал і користувацький інтерфейс значно покращилися. Зокрема, була додана підтримка візуального скриптування, нові ноди для UI, а також покращена підтримка експорту ігор на різні платформи.

Версія 2.1 (2016) зосередилася на покращенні стабільності, розширюваності та зручності використання. Додана підтримка плагінів, це дозволило розробникам створювати власні інструменти для движка.

Підтримка та значне оновлення 3D (2018–2020):

Версія 3.0 (2018): отримала значне оновлення з численними новими функціями. Підтримка Vulkan, новий 3D рендерер, інтеграція з фізичним рушієм Bullet і підтримка C# були найважливішими нововведеннями.

Це зробило Godot серйозним конкурентом для інших ігрових движків, таких як Unity та Unreal Engine.

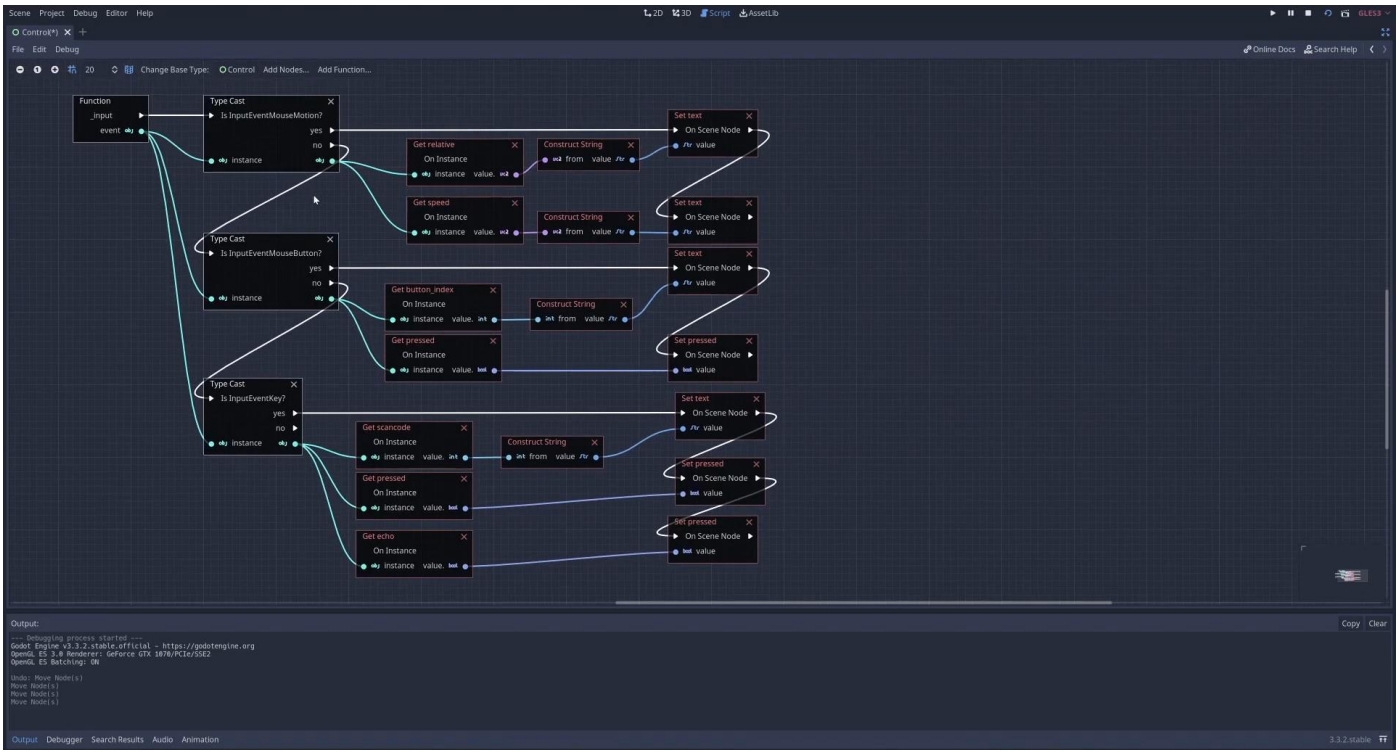


Рисунок 2.3 Godot Engine версії 3.0, візуальне скриптування.

Версія 3.1 (2019) додала покращення для 2D і 3D, нові інструменти для анімації та користувацького інтерфейсу, а також підтримку OpenVR для розробки віртуальних віртуальних додатків.

Версія 3.2 (2020): продовжила тенденцію до вдосконалення, додавши нові можливості для 2D-анімації, підтримку WebAssembly, покращення для рендеринга та продуктивності. У цій версії також були додані нові інструменти, що покращило процес розробки ігор.

Актуальні зміни (2021 та далі):

Версії 3.3 та 3.4 (2021): Були внесені оптимізації та нові функції,

включаючи підтримку нових платформ, нові функції роботи з мережею та покращення інструментів для розробки 2D-ігор.

Версія 3.5 (2022) включає нові функції для роботи з анімаціями та скриптуванням, а також додаткові покращення продуктивності та стабільності.

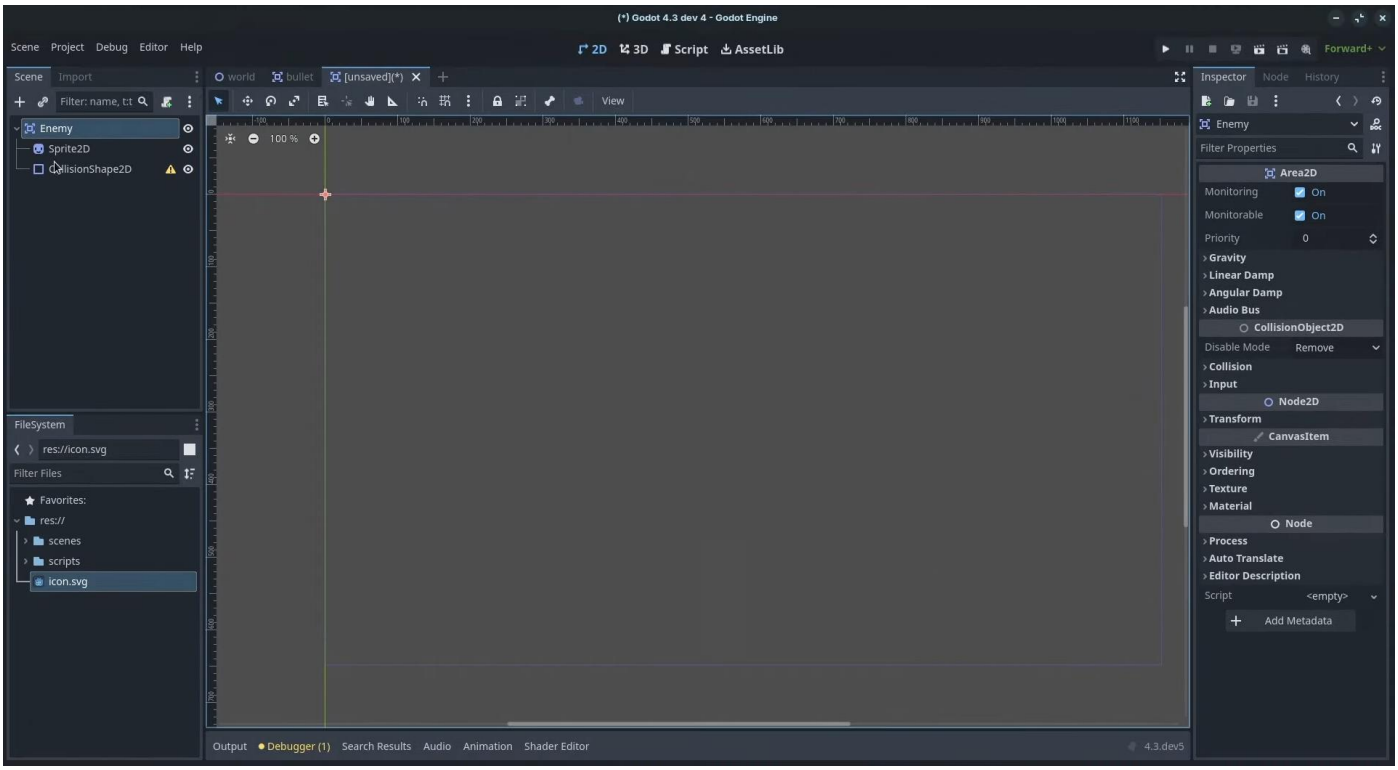


Рисунок 2.4 Godot Engine версії 4.0.

Версія 4.0 (2023) це повна інтеграція з рендерингом Vulkan, покращення редактора, нові можливості для 3D-рендеринга, покращена система анімацій та багато іншого.

Версія 4.1 (2023) взяла фокус на стабільності, продуктивності та поліруванні. Покращено уникнення навігації ШІ та можливість від'єднувати редактори коду та розміщувати їх на інших дисплеях.

Версія 4.2 (2023) додала необхідні виправлення помилок і захоплюючі нові функції, які перетворюють Godot 4 на ще кращий і досконаліший інструмент для втілення ідей щодо ігор і програм. Відновлення та покращення усіх існуючих систем.

Від початкового внутрішнього інструменту до сьогодні ігровий движок Godot став одним із найвідоміших відкритих ігрових движків у світі. Він є важливим інструментом для розробників ігор усіх рівнів завдяки своєму постійному розвитку, відкритому вихідному коду та спільноті. Godot постійно розвивається завдяки постійним оновленням і покращенням, що дозволяє йому залишатися актуальним і ефективним інструментом для створення ігор.

1.2 Основні можливості та функції движка

Godot Engine пропонує розробникам ігор широкий вибір інструментів для створення 2D та 3D ігор. Розглянемо детально основні функції та можливості цього движка.

Архітектура сцени, та нодів:

Сцени: Це важлива структурна одиниця Godot. Сцена може бути будь-чим, від звичайного об’єкта до складної ігрової сцени з кількома елементами.

Щоб легко створювати ієрархічні структури, сцени можуть взаємодіяти з іншими сценами.

Ноди: є основними будівельним блоками Godot. Кожна сцена складається з нодів, які виконують різні функції. Ці функції включають рендеринг, обробка фізики, анімацію, управління інтерфейсом тощо. Тривимірні та двовимірні ігри мають різні види нодів.

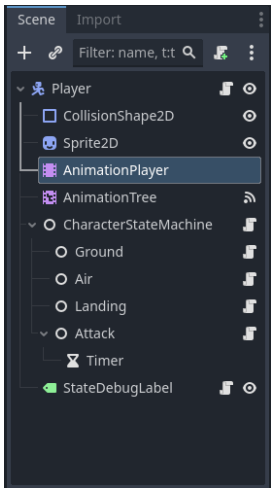


Рисунок 2.5 Різноманітні ноди, у редакторі сцен Godot Engine.

Приклади нодів:

Node2D/Node3D: Базові ноди для 2D і 3D сцен відповідно.

Sprite: Нод для відображення 2D зображень.

RigidBody/StaticBody: Ноди для фізичних об'єктів у 3D світі.

Label: Нод для відображення тексту.

Програмування та скриптування:

Мови програмування:

GDScript: Основна мова програмування в Godot, схожа на Python. Вона спеціально створена для швидкого та ефективного написання ігрової логіки.

C#: Підтримка C# дозволяє використовувати потужні можливості .NET екосистеми.

VisualScript — це візуальна мова програмування, яка дозволяє без написання коду створювати логіку гри.

Інші мови: Завдяки модулям є можливість інтегруватися з іншими мовами.

Характеристики скриптування:

Інтуїтивний синтаксис: GDScript має простий синтаксис, що робить навчання та написання коду простішим.

Інтеграція з редактором: Скрипти легко інтегруються з редактором сцени, що дозволяє швидко створювати та налаштовувати поведінку об’єктів.

Автоматичне управління пам'яттю: Godot звільняє розробників від необхідності контролювати видалення об'єктів, керуючи пам'яттю автоматично.

Розробка 2D ігор:

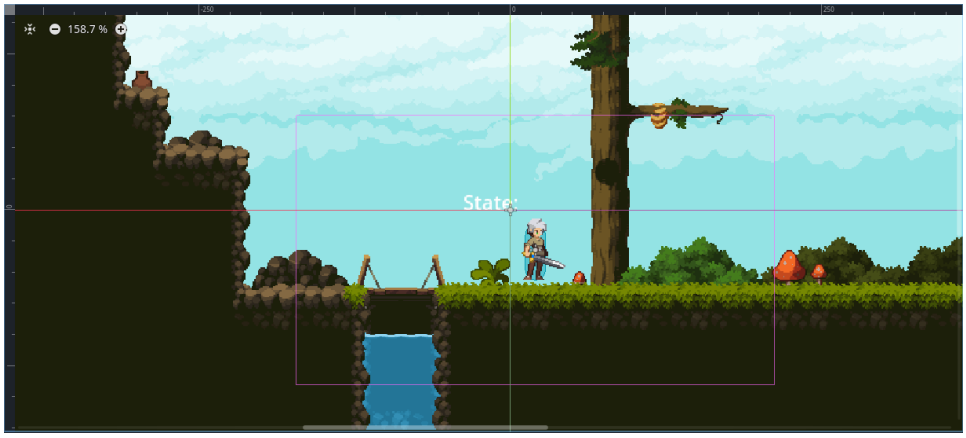


Рисунок 2.6 2D Редактор Godot Engine.

Основні можливості:

Рендеринг: потужний двовимірний рендерер з шейдерами, освітленням, тінями та ефектами постобробки.

Анімація: анімаційні дерева та твінинг є частиною вбудованої системи для створення та управління анімаціями.

Фізика: тіла RigidBody, StaticBody, KinematicBody та Area2D підтримуються

за допомогою фізики.

Тайлмапи: інструмент для створення рівнів із тайлами. Він підтримує автотайли та має редактор тайлсетів.

Партіклі: система, яка створює ефекти частинок, такі як пожежі, дим і вибухи.

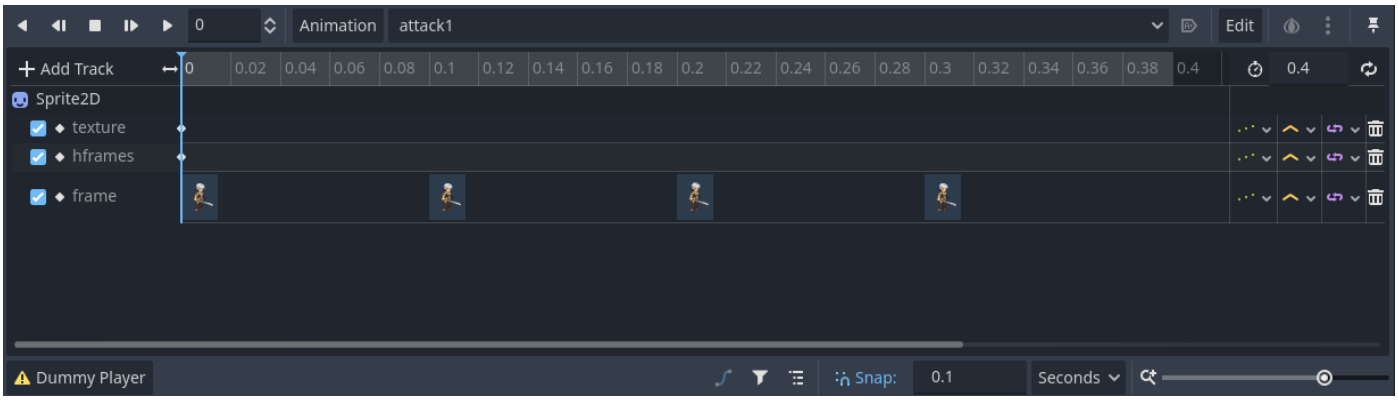


Рисунок 2.6 Панель Анімації Godot Engine.

Приклад функцій:

CanvasLayer — це нод, який використовується для керування шарами графіки 2D.

Polygon2D — це нод для створення складних форм і полігонів у 2D.

TiLeMap — це додаток, який дозволяє створювати та керувати тайловими картами.

Розробка ігор у 3D:

Основні перспективи:

Рендеринг: Vulkan це сучасний 3D рендерер з можливостями освітлення, тіней, шейдерів, постобробки та PBR (фізико-базований рендеринг).

Фізика: Інтеграція з фізичним рушієм Bullet, підтримка RigidBody, StaticBody, KinematicBody, SoftBody, а також обробка зіткнень та обмежень.

Анімація: Потужна система для створення скелетних анімацій, підтримка анімаційних дерев та твінингу.

Террейн: Інструменти для створення і управління 3D ландшафтами.

Імпорт: можна імпортувати моделі з відомих форматів, таких як FBX, OBJ і Collada.

Приклади функцій:

Spatial: Базовий нод для 3D сцен.

MeshInstance: Нод для відображення 3D моделей.

Camera: Нод для налаштування камери у 3D світі.

Інструменти для розробки:

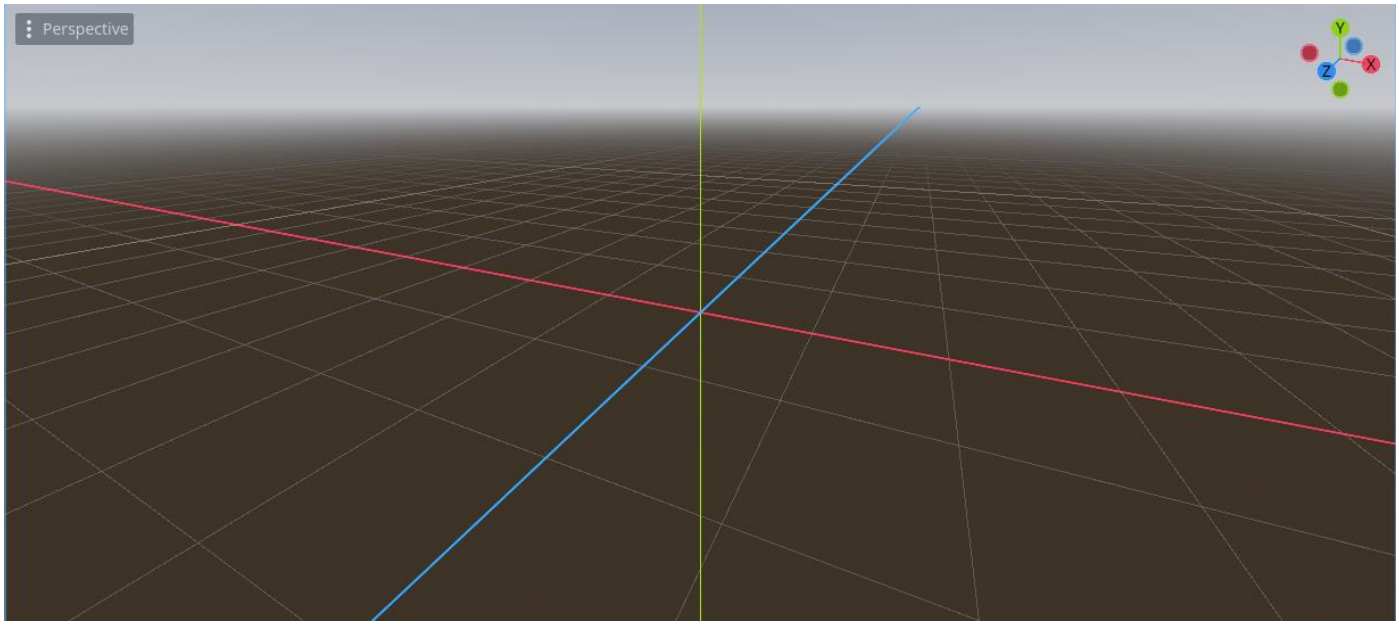


Рисунок 2.7 3D Редактор Godot Engine.

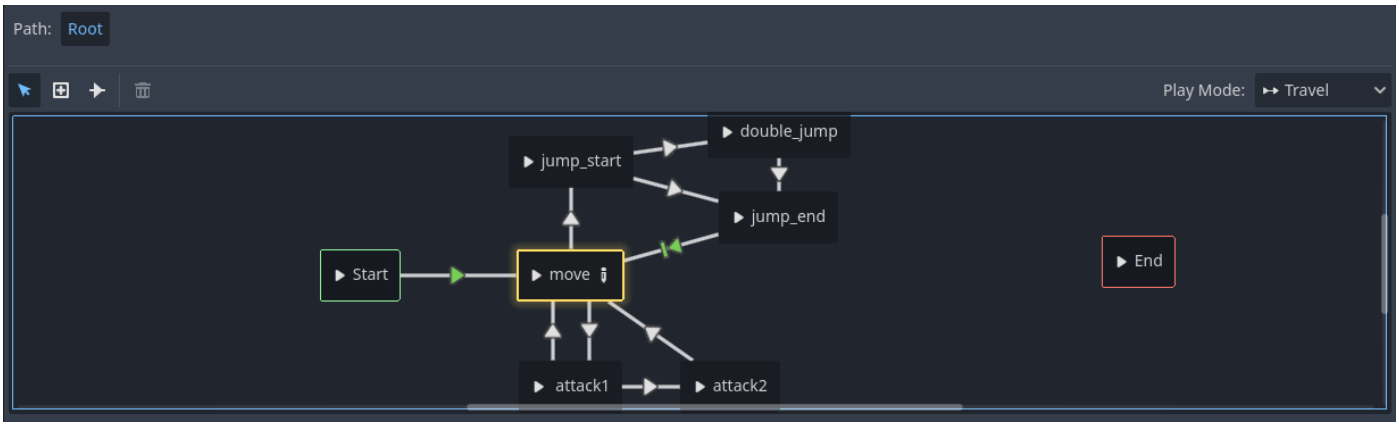
Редактор сцени:

Інтуїтивний інтерфейс: Простий графічний інтерфейс редактора сцени

робить створення та налаштування сцени простим завданням.

Ієрархії нодів: можливість створення складних ієрархій нодів для структурованої розробки.

Тулсет: різноманітні інструменти для розміщення об’єктів, налаштування



властивостей і керування сценами.

Рисунок 2.8 Animation Tree у Godot Engine.

Анімаційний редактор:

Ключові кадри: Підтримка ключових кадрів для створення анімацій.

Animation Tree — це інструмент, який можна використовувати для створення складних анімаційних послідовностей.

Можливість плавного переходу між анімаціями відома як Твінинг.

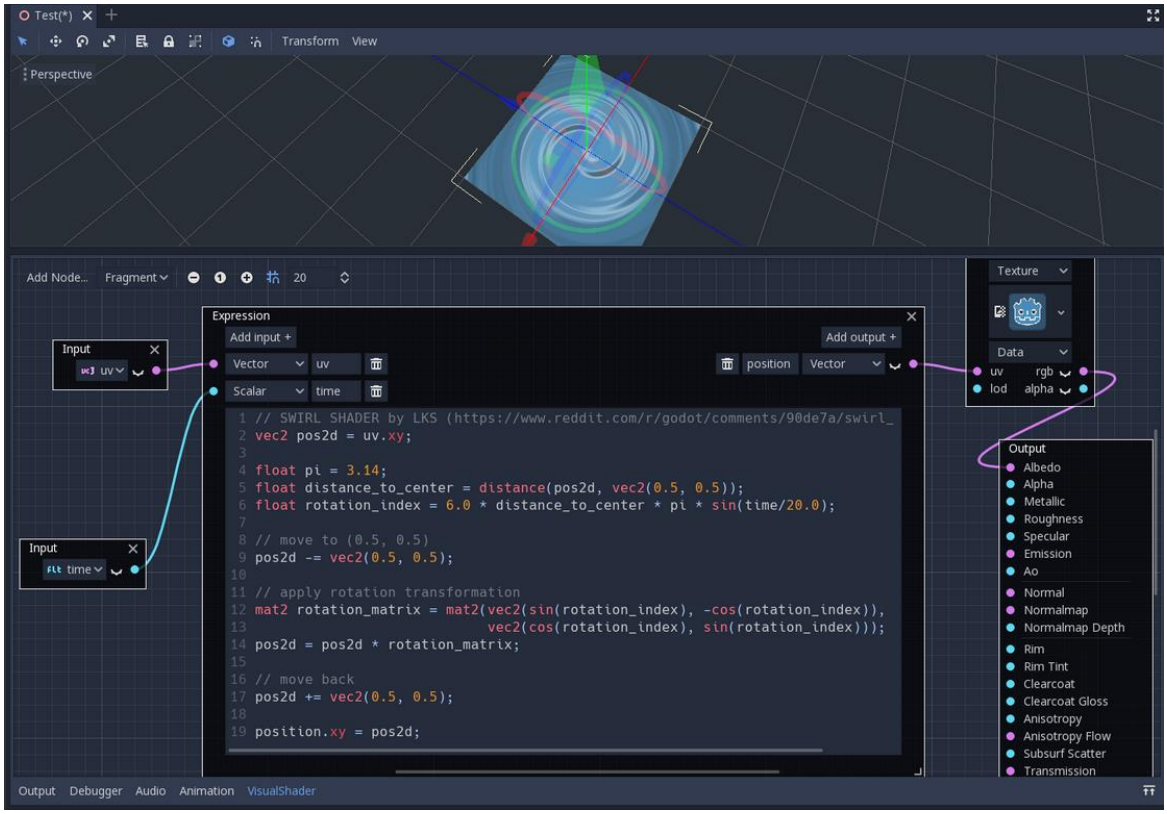


Рисунок 2.9 Редактор шейдерів Godot Engine.

Редактор шейдерів:

Візуальний редактор: Інструмент для написання та редагування шейдерів у режимі реального часу.

Підтримка GLSL: Можливість написання шейдерів на мові GLSL для створення складних графічних ефектів.

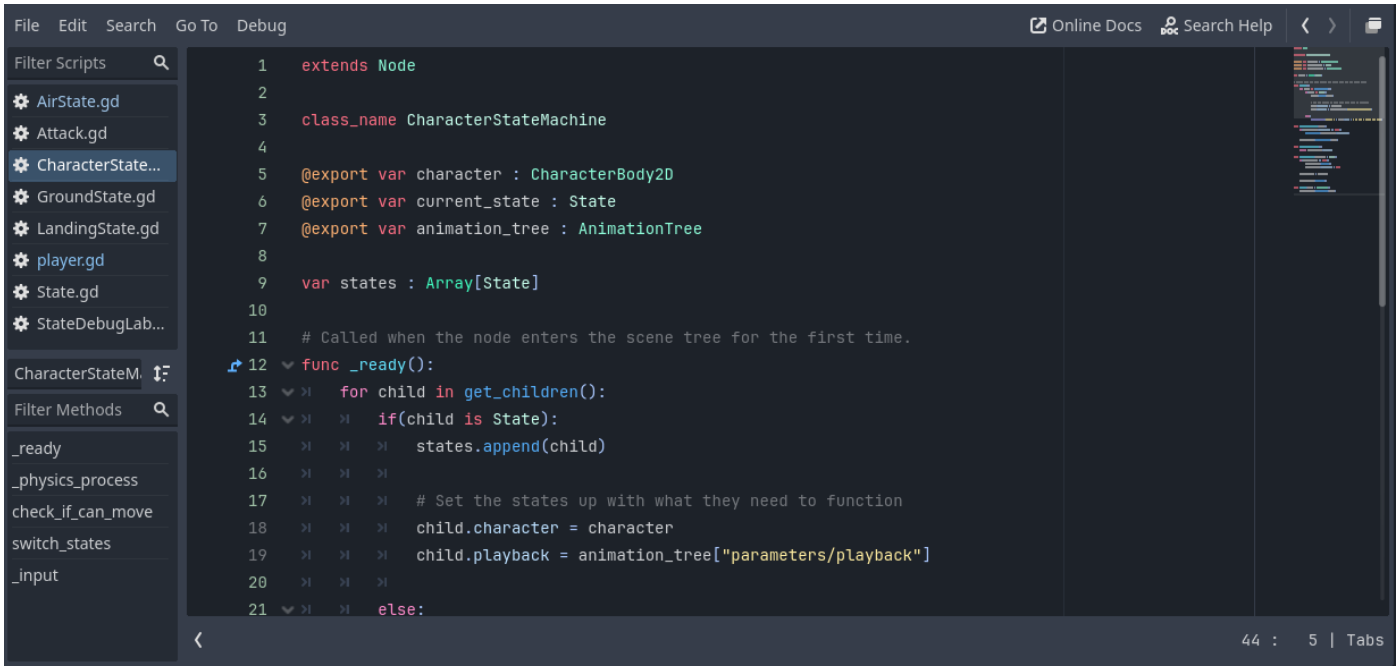


Рисунок 2.10 Редактор Скриптів Godot Engine.

Інтеграція та розширення:

Модулі та плагіни:

Розширюваність: можливість додавати нові функції за допомогою модулів і плагінів.

Спільнота: активна спільнота розробників, яка створює та ділиться плагінами, щоб зробити Godot більш функціональним.

Експорт на різні платформи:

Платформи: Підтримуються ігри на Windows, macOS, Linux, Android, iOS, HTML5 та інших платформах.
Настроювані налаштування експорту: Інструменти для налаштування параметрів експорту, таких як розмір пакету, налаштування графіки та інші.

Godot Engine пропонує багатий набір функцій та можливостей, які дозволяють розробникам створювати як 2D, так і 3D ігри. Його приваблива архітектура сцени та нодів, потужні інструменти для скриптування та програмування, а також широкий вибір інструментів для розробки та розширення приваблюють розробників ігор з різним рівнем досвіду. Godot постійно розвивається, надаючи розробникам більше інструментів і

можливостей для створення ігор найвищої якості.

Інше:

Відтворення відео з вбудованою підтримкою Ogg Theora.

Режим Movie Maker для запису відео з поточного проекту із синхронізованим аудіо та ідеальною частотою кадрів.

Низькорівневий доступ до серверів, що дозволяє обійти накладні витрати дерева сцен, коли це необхідно.

Інтерфейс командного рядка для автоматизації.

Експортуйте та розгортайте проекти за допомогою платформ постійної інтеграції.

Сценарії завершення оболонки доступні для Bash, zsh і fish.

Друк кольорового тексту на стандартний вихід на всіх платформах за допомогою print_rich.

Підтримка модулів C++, статично пов’язаних у бінарний файл двигуна.

Двигун і редактор написані на C++17.

Можна скомпілювати за допомогою GCC, Clang і MSVC. MinGW також підтримується. Доброзичливе ставлення до пакувальників. У більшості випадків можна використовувати системні бібліотеки замість тих, які надає Godot. Система збірки нічого не завантажує. Конструкції можна повністю відтворити.

Ліцензовано відповідно до дозволу MIT.

Відкритий процес розробки з внесками.

1.3 Переваги та недоліки Godot

Godot Engine, як і кожен інший ігровий движок, має свої сильні сторони та слабкі сторони. Разом ми розглянемо основні переваги та недоліки цього

движка.

Переваги Godot Engine:

Відкритий вихідний код і безкоштовність: Godot є проектом з відкритим вихідним кодом, розповсюдженим під ліцензією MIT. Це дозволяє розробникам використовувати, змінювати та поширювати движок без обмежень.

Безкоштовність: Всі функції Godot доступні безкоштовно, і вам не потрібно платити роялті чи ліцензійні збори.

Широкий набір функцій для розробки двовимірних ігор: потужний двовимірний рендерер: Godot є одним із найкращих інструментів для розробки двовимірних ігор з підтримкою освітлення, тіней, шейдерів і ефектів постобробки.

Інструменти для анімації: вбудована система анімації, яка включає анімаційні дерева, твінинг і підтримку ключових кадрів.

Гнучкість вибору мов програмування: GDScript — це власна мова програмування, схожа на Python, яка спеціально створена для швидкого та ефективного написання ігрової логіки.

C#: Підтримка C# дає розробникам можливість використовувати потужні можливості.системи NET.
GDExtension (C, C++, Rust, D, ...) дозволяє використовувати майже будь-яку мову з Godot Engine.

VisualScript — це візуальна мова програмування, призначена для людей, які люблять працювати без написання коду.

Інтуїтивний і зручний редактор: Інтерфейс: простий у використанні інтерфейс редактора дозволяє легко створювати та редагувати сцени.

Ієрархія нодів: можливість створення складних ієрархій нодів для структурованої розробки.

Інтеграція інструментів: включає інструменти для управління ресурсами, анімації та редагування шейдерів.

Підтримка кількох платформ: Godot може експортувати ігри на різні платформи, включаючи Windows, macOS, Linux, Android, iOS, HTML5 та інші, що дозволяє легко розповсюджувати ігри на різних пристроях.

Активна спільнота та документація: Спільнота — це активна, дружня спільнота розробників, яка створює велику кількість ресурсів, туторіалів і плагінів.

Документація: чудова документація, яка включає всі аспекти роботи з Godot, включаючи приклади коду та детальні пояснення.

Недоліки Godot Engine:

Обмеження в розробці 3D:

Можливості для графіки, хоча Godot має потужний 3D рендерер, він все ще не має інструментів і деталізації, які доступні в таких движках, як Unity або Unreal Engine. Зокрема, це стосується високотехнологічних графічних ефектів і роботи з великими сценами.

Інструменти: Порівняно з конкурентами, деякі інструменти 3D розробки можуть бути менш розумними або розвиненими.

Перформанс:

Оптимізація для деяких компонентів продуктивності можна знадобитися більше зусиль для оптимізації, особливо коли працює з великими 3D-сценами або складними фізичними об’єктами.

Мобільні платформи: через широкий спектр пристроїв і технічних характеристик, оптимізація мобільних платформ може бути складною.

Молодий проект:

Godot є досить молодим проектом, і хоча він активно розвивається, йому все ще бракує деяких функцій і інструментів, які є у більш зрілих ігрових движках.

Розширюваність: Інтеграція з деякими зовнішніми інструментами та сервісами може бути менш розвиненою.

Проблеми з сумісністю:

Переходи між версіями, іноді оновлення до нових версій Godot викликають проблеми з сумісністю, які вимагають переробки частини проекту.

Менша спільнота, хоча членство в спільноті продовжує зростати, кількість плагінів, бібліотек і туторіалів, доступних для використання, все ще менша, ніж у більш відомих движках, таких як Unity та Unreal Engine.

Godot Engine має багато переваг, які приваблюють розробників, особливо тих, хто працює над двовимірними іграми або бажає використовувати безкоштовні інструменти та відкритий вихідний код. Тим не менш, він має деякі обмеження, особливо щодо розробки та оптимізації 3D.

2 ПРОЕКТУВАННЯ ГРИ

Першим і одним із найважливіших етапів у розробці будь-якої відеогри є розробка концепції гри. Розробка основної ідеї гри, жанру, механік, візуального стилю та інших важливих елементів є частиною цього. Розглянемо детальніше основні етапи розробки концепції гри.

2.1 Концепція гри

Для початку ми оберемо просту концепцію для першої гри, щоб навчитися використовувати інструменти та зробити початковий проект який можливо буде доповнити у майбутньому новими механіками, рівнями, або графічними елементами. Тому Основна концепція цього проекту це простий платформер у якому гравець керує персонажем.

2.2 Розробка геймдизайну

Всі аспекти створення ігрового досвіду — від механік і рівнів до взаємодії з користувачем і наративу — охоплюються геймдизайном. Це складний процес, і ретельне планування та увага до деталей є необхідними. Поговоримо про основні етапи розробки геймдизайну.

Рух персонажа: переміщення визначає швидкість руху персонажа, а також його здатність бігати, стрибати та переміщатися іншими способами.

Стрибки включають стрибки від стін, подвійні стрибки, стрибки на висоту та довжину тощо. У нашому проекті ми реалізуємо подвійні стрибки.

Розробка рівнів: спочатку розробимо тестовий рівень, у якому не буде преград, але ми зробимо деякі естетичні елементи на цьому рівні.

2.3 Вибір стилю графіки

Вибір стилю графіки для відеогри є важливою частиною процесу розробки, оскільки він значно впливає на сприйняття гри, її атмосферу та

цільову аудиторію. Для того, щоб вибрати найкращий стиль графіки, необхідно врахувати багато важливих факторів.

Жанр гри:
Платформери: піксель-арт або мультяшний стиль часто використовується, щоб підкреслити простоту та веселість гри.

RPG: Для створення глибшого ігрового світу може використовуватися більш реалістичний стиль.

Шутери: Вимагатимуть реалістичного або футуристичного стилю, щоб підкреслити динамічність та напруженість гри.

Цільова аудиторія:

Діти: Прості та дружні персонажі з веселим, яскравим і мультяшним стилем.

Підлітки: Може бути більш стильний та екшн-орієнтований, з акцентом на деталі та візуальні ефекти.

Дорослі: Реалістичний або художній стиль, що підкреслює глибину сюжету та серйозність теми.

Також слід враховувати ресурси та можливості команди розробників, наприклад якщо команда розробників дуже мала, тоді піксель-арт або простий 2D стиль може бути більш досяжним, оскільки вони вимагають менше ресурсів та часу на розробку. Або якщо команда розробників велика, то і можливості в неї теж будуть великі, з детальною 3D графікою, анімаціями та спецефектами.

Ще треба враховувати цільові платформи на яких буде роспростронятися гра. Якщо це мобільні пристрої, для них потрібно менш ресурсів, тобто простий стиль або піксель арт підійде найкраще. Якщо це Консолі та Персональні Комп'ютери, вони можуть підтримувати більш складну графіку, тому 3D або високоякісний 2,5D стиль може бути доцільним.

Які саме є стилі графіки?

Піксель-арт:

Переваги: Ностальгічний стиль, легкість створення та оптимізація, висока продуктивність.

Недоліки: Може виглядати старомодно, обмеження у деталізації.

Приклад: Stardew Valley, Celeste.

Рисунок 3.1 Приклад Піксель Арту, на прикладі інді ігор Celeste, Stardew Valley.

Ручна анімація або мультяшний дизайн:

Переваги: барвистий і привабливий, підходить для широкого загалу.

Недоліки: Для створення анімацій потрібно багато зусиль, і це може бути важко підтримувати.

Hollow Knight і Cuphead є двома прикладами цього.

Рисунок 3.2 Приклад мультяшної/мальованої графіки з іграми Cuphead, Hollow Knight.

Реалістичний стиль:

Переваги: Висока деталізація, підходить для серйозних ігор з глибоким сюжетом.

Недоліки: оптимізація складна та вимагає багато ресурсів.

Приклади: Red Dead Redemption 2 і The Last of Us.

Рисунок 3.3 Приклад Реалістичного стилю графіки з іграми RDR 2 та TLOU.

Художній стиль:

Переваги: унікальний вигляд, можливість виділитися, ідеальний для створення особливої атмосфери.

Недоліки: можуть не підійти для всіх жанрів, вимагають високої майстерності.

Journey, Ori and the Blind Forest, є прикладом.

Рисунок 3.4 Приклад Художнього стилю, Journey та Ori and the Blind Forest.

Low Poly:

Переваги: стильний вигляд, менші вимоги до продуктивності та простота створення моделей.

Недоліки: Для серйозних або реалістичних ігор може бути неприйнятно занадто мало деталей.

Приклади включають Bomb Rush Cyberfunk і Risk Of Rain 2.

Рисунок 3.5 Приклад Low Poly стилю, Bomb Rush Cyberfunk та Risk Of Rain 2.

Cel-shading:

Переваги: Стиль, який виглядає як живий комікс або аніме, має яскравий і виразний вигляд.

Недоліки: може не підійти для всіх жанрів, вимагає певних навичок.

Приклади: The Legend of Zelda: Breath of the Wild, Borderlands.

Рисунок 3.6 Приклад Cel-Shading стилю, Borderlands 3 та TLoZ:BotW.

Жанр гри, цільова аудиторія, ресурси команди та технічні можливості повинні бути враховані під час вибору стилю графіки. Є переваги та недоліки кожного стилю графіки, і вибір правильного стилю графіки може суттєво вплинути на успіх гри. Враховуючи всі ці елементи, під наш перший проект підходить стиль піксель арт, тому зупинимося не нему. Для цього ми будемо використовувати безкоштовний набір ассетів під назвою Legacy Fantasy Bundle під авторством Anakolisa на Itch.io.

Рисунок 3.7 Ассети Legacy fantasy game asset by Anakolisa.

3 РОЗРОБКА ГРИ

3.1 Створення ігрових сцен

Використання сцен для організації контенту та логіки гри є однією з основних концепцій Godot. Сцени можуть містити будь-які елементи гри, включаючи персонажів, ворогів, предмети, інтерфейс користувача тощо.

Створення ігрових сцен для Godot складається з кількох основних етапів: створення нової сцени, додавання та налаштування вузлів, створення окремих сцен для персонажів і інтеграція цих сцен у основну сцену. Налаштування управління персонажем і камери, що стежить за його рухом, також є важливими. Цей процес дозволяє організувати структуру гри таким чином, щоб було легко керувати проектом і його продовжувати.

У Godot сцена складається з набору вузлів, які розташовані в ієрархії. Кожен вузол виконує свою функцію та призначення. Складні структури можна створити, зберігаючи сцени як окремі файли та вкладаючи їх одна в одну.

Вузли:

Node — це базовий елемент, який може мати дочірні вузли та виконувати певні функції.

Node2D: використовується для двовимірних ігрових елементів, таких як спрайти, області зіткнень тощо.

Control — це модуль, який використовується для створення елементів UI.

Sprite — це вузол для відображення зображень.

CollisionShape2D — це вузол, який можна використовувати для визначення форми зіткнення в двомірному просторі.

Після швидкої настройки проекту ми можемо почати створювати над першою сценою:
Першою сценою буде наш тестовий рівень, тому створюємо новий вузол “Node” та називаємо його testlevel, додаємо до нього дочірній вузол “TileMap”

У властивостях вузлу TileMap додаємо текстуру відповідаючу за набір тайлів, з яких ми потім будемо робити об’єкти у нашому рівні, з нашим набором ассетів цей файл називається Tiles.png, додаємо його та ще один файл під назвою Background.png.

Рисунок 4.1 Меню TileSet у Godot Engine.

Далі ми редагуємо розмір тайлів у властивостях тайлсету, на 16x16 пікселів. Після цього ми можемо обрати будь який тайл у меню TileSet та у властивостях кожного тайлу, у вкладці Physics ми можемо додавати свої колізії, щоб інші об’єкти чи гравець могли на них стояти, або впиратися.

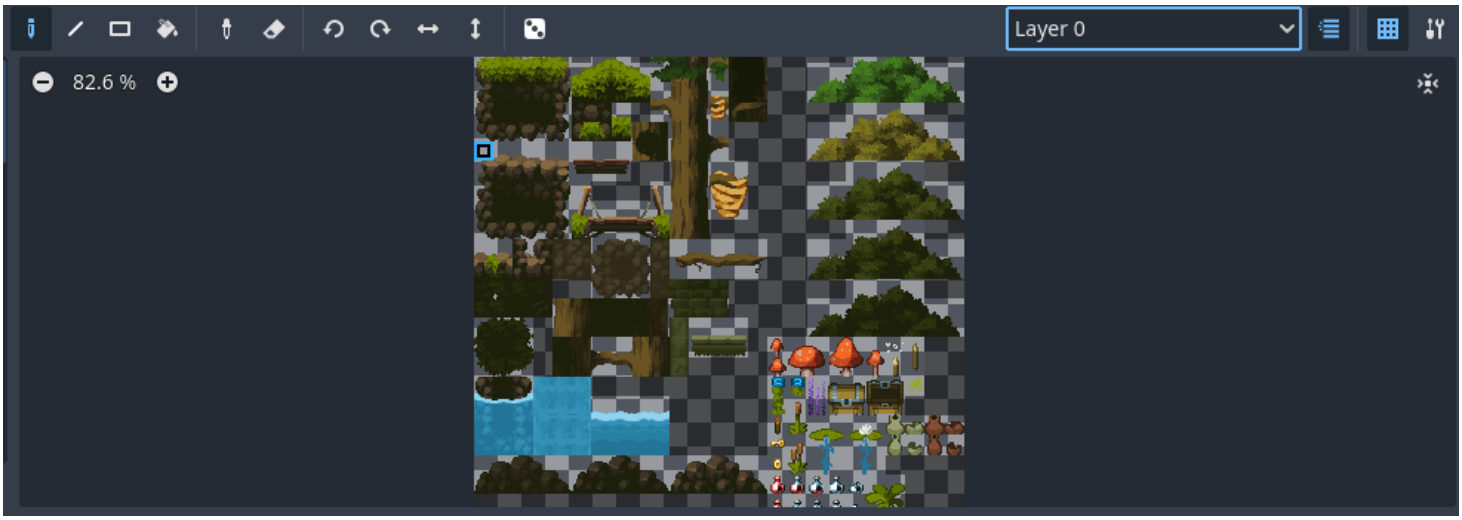


Рисунок 4.2 Меню TileMap у Godot Engine.

Після цього ми можемо перейти у меню TileMap та обрати один, чи декілька або навіть паттерн тайлів, та малювати їх прямо на нашій новій сцені, наверху цього меню ми також можемо обирати шари на яких ми малюємо ці тайли. У цьому набору тайлів є дуже багато візуальних об’єктів фону з яких ми можемо створити приємний очам пейзаж лісу між горами.

Тайлами з файлу Background.png ми можемо намалювати фон нашого рівню.

Таким чином ми створили тестовий рівень у якому може знаходитись наш персонаж.

Рисунок 4.3 Огляд готового тестового рівня у 2D сцені.

Далі наша ціль це створення персонажу яким буде керувати гравець.

3.2 Розробка персонажа

Другою сценою буде персонаж керований гравцем, для цього ми в редакторі сцен створимо новий вузол “CharacterBody2D” та перейменуємо його на Player, та додамо до нього Child Node під назвою Sprite.

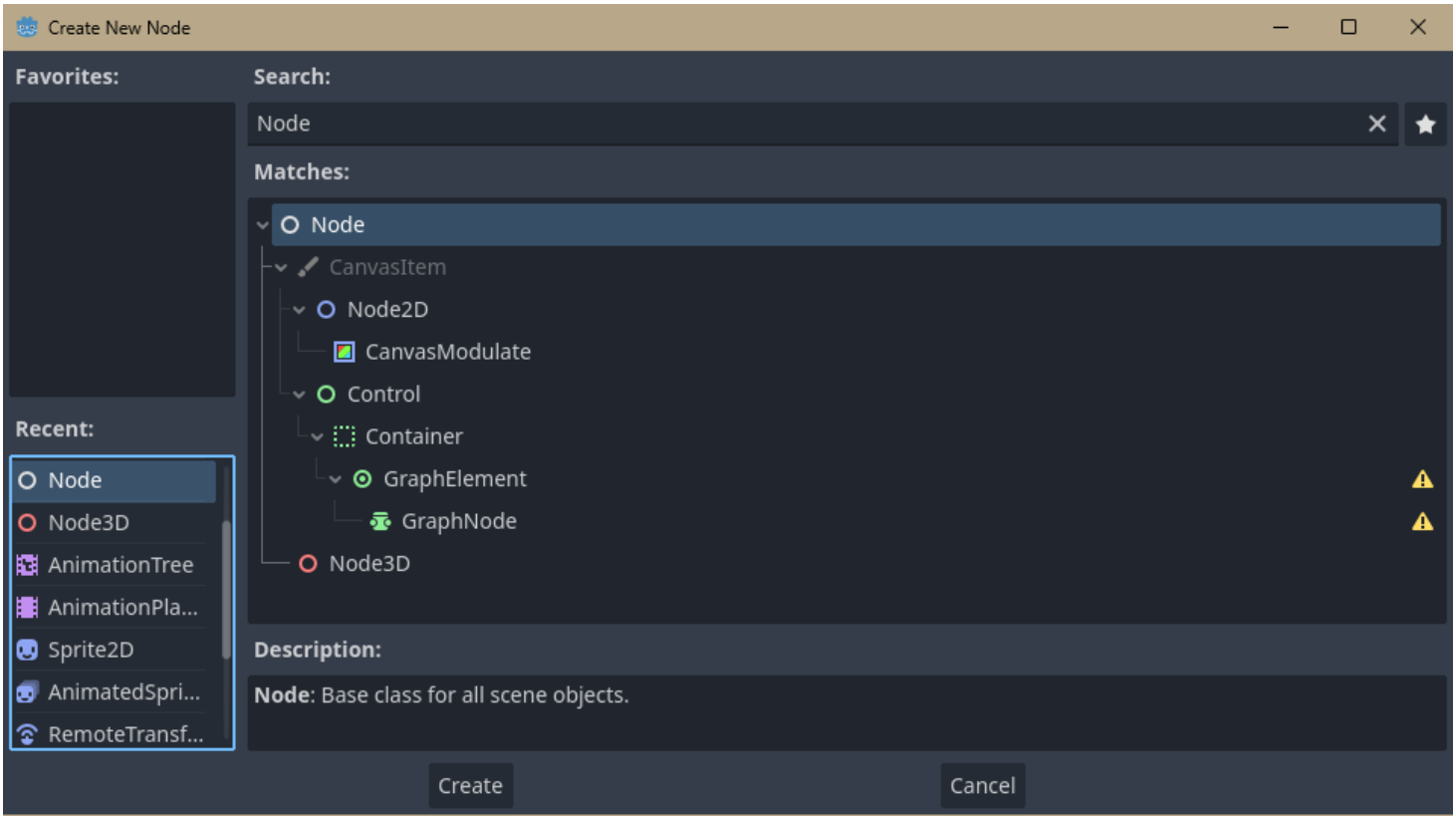


Рисунок 4.4 Редактор сцен Godot Engine.

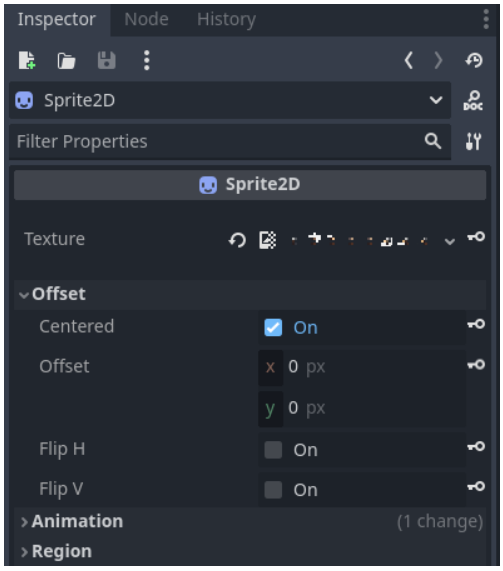


Рисунок 4.5 Властивості вузла Sprite у вікні Inspector, Godot Engine.

У властивостях вузла Sprite додаємо текстуру нашого персонажу. До вузлу Sprite додаємо дочірній вузол CollisionShape2D, цей вузол відповідає за колізію нашого гравця з другими об’єктами у світі. У властивостях CollisionShape2D обираємо форму, до нашої текстури краще всього підійде CapsuleShape2D.

3.3 Реалізація ігрової логіки та фізики

Додамо камеру яка буде слідкувати за нашим персонажем, для цього створимо новий дочірній вузол Camera2D та включемо параметр Current у властивостях цього вузла, також налаштуємо як плавно вона буде слідкувати, та призначимо їй нашого персонажа як ціль.

Далі нам потрібно розробити логіку персонажу.

Для початку додамо скрипт до нашої сцени персонажу, для цього у редакторі сцен натискаємо кнопку додати скрипт на обраній сцені Player.

```
1  extends CharacterBody2D
2
3
4  @export var speed  : float = 200.0
5
6  @onready var sprite : Sprite2D = $Sprite2D
7  @onready var animation_tree : AnimationTree = $AnimationTree
8  @onready var state_machine : CharacterStateMachine = $CharacterStateMachine
9
```

Рисунок 4.6 початковий код скрипту Player.

команда @export дозволяє нам змінювати змінну за допомогою Inspector, не міняючи код програми. Створюємо нову змінну Speed та надаємо їй значення. Це значення буде відповідати кількості пікселів які гравець пройде за секунду.

```
func _ready():
    screen_size = get_viewport_rect().size
```

Рисунок 4.7 Функція _ready.

Також ми маємо функцію _ready, яка оновлюється під час того як движок загружає Node, тому ми можемо використовувати її для того щоб змінити розмір вікна гри.

Для наступного кроку нам потрібно назначити нашому проекту Input, Godot має свій стандартний Input для тестування, але нам потрібно зробити свої унікальні методи для кожного елементу, будь то рух або атака. У налаштуваннях проекту у вкладці Input Map записуємо up, down, left, right, jump та кожен раз натискаємо кнопку Add. Натискаємо на + поряд з методом та натискаємо відповідні клавіші для цього вводу, наприклад давайте використовуємо WASD та Space для цього.

```
18  func _physics_process(delta):
19      # Add the gravity.
20      if not is_on_floor():
21          velocity.y += gravity * delta
22
23      # Get the input direction and handle the movement/deceleration.
24      direction = Input.get_vector("left", "right", "up", "down")
25
26      if direction.x != 0 && state_machine.check_if_can_move():
27          velocity.x = direction.x * speed
28      else:
29          velocity.x = move_toward(velocity.x, 0, speed)
```

Рисунок 4.8 Функції фізики та руху гравця.

Тут ми також можемо додати команди play() або stop() але ми не використовуємо AnimatedSprite2D та будемо робити анімацію окремо у Animation Tree.

```
10    # Get the gravity from the project settings to be synced with RigidBody nodes.
11    var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")
12    var direction : Vector2 = Vector2.ZERO
```

Рисунок 4.9 Змінні гравітації та синхронізація налаштувань проекту.

Додамо змінну гравітації та надамо їй значення налаштувань проекту, щоб ми могли у будь який момент змінити її без зміни коду.

3.4 Робота з анімацією

Підготовка анімацій: спочатку ми зробимо базові анімації за допомогою AnimationPlayer, щоб потім додати та зв’язати їх у AnimationTree.

Почнемо з Idle анімації, анімація яка буде грати коли персонаж стоїть на місці та нічого не робе.

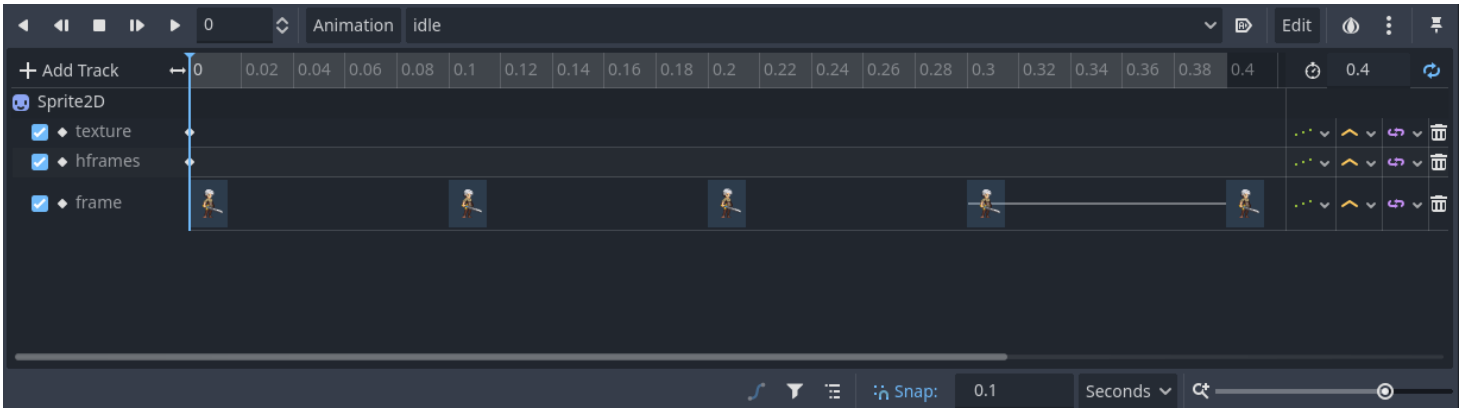


Рисунок 4.10 AnimationPlayer з прикладом анімації Idle.

Для початку додаємо необхідні треки для анімації, це texture, hframes, та frame.

Texture відповідає за текстуру анімації, hframes відповідає за кількість горизонтальних кадрів у анімації, та frame відповідає за кожен кадр анімації

та коли він буде програватись.

Додаємо ключ до треку текстури та додаємо текстуру нашої анімації.

Також додаємо ключ до hframes та задаємо кількість горизонтальних кадрів нашої анімації, у випадку з Idle, це 4 кадри. задаємо довжину анімації, це також 4 кадри, та додаємо кожен кадр анімації окремо через Inspector з обраним Sprite2D. Таким чином ми зробили Idle анімацію, по цьому прикладу ми також робимо анімації під назвами: jump_start, jump_end, run, double_jump. В деяких випадках нам потрібно натиснути кнопку Loop щоб наша анімація повторювалася наприкінці. Також зміною довжини анімації ми можемо керувати її скоростью.

Ми можемо подивитися та перевірити кожну анімацію завдяки плееру анімацій щоб вдосконалюватися результатом нашої роботи.

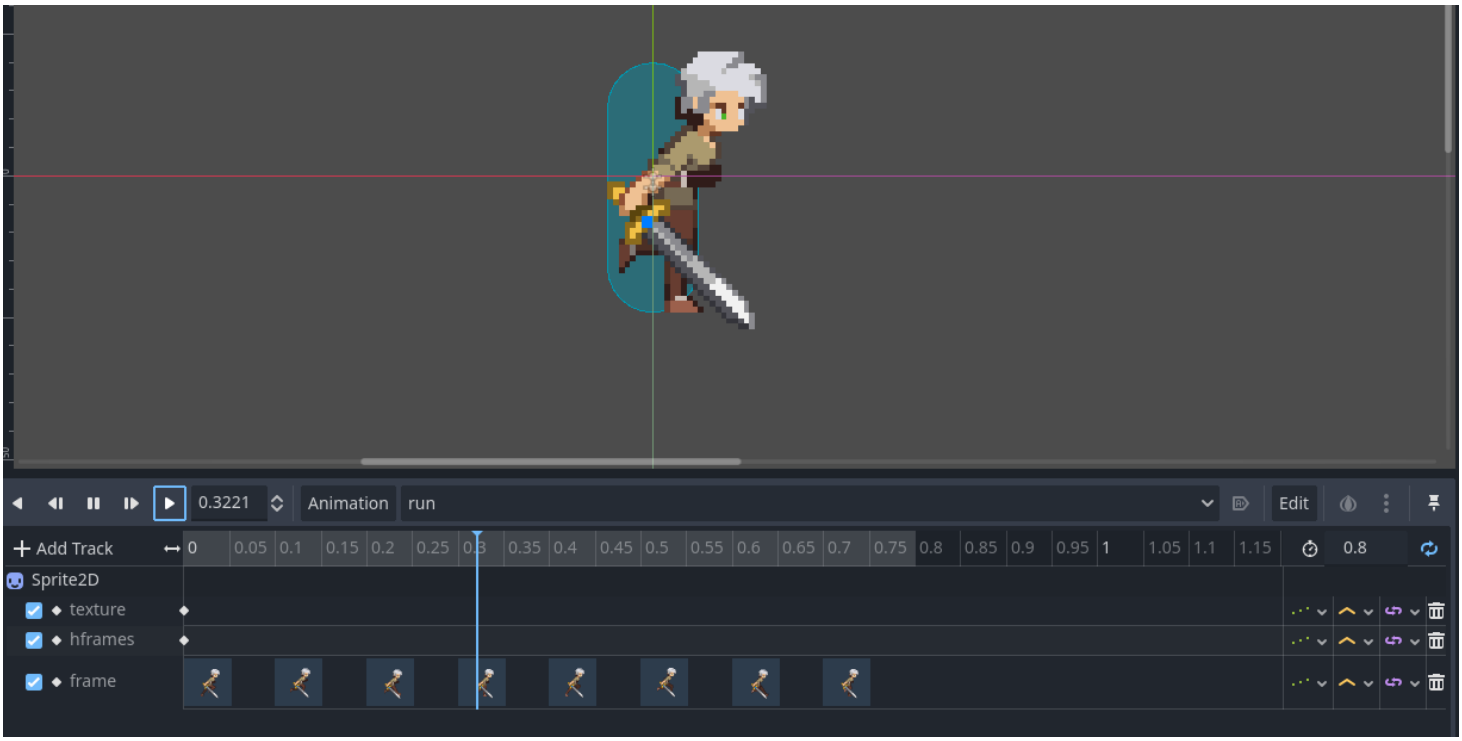


Рисунок 4.11 плеер анімації з прикладом анімації run.

так як наш персонаж може рухатися в дві різні сторони нам потрібно зробити функцію яка буде перегортати спрайт нашого персонажа у правильну сторону. Відповідно значенню змінної направлення ми перегортаємо спрайт.

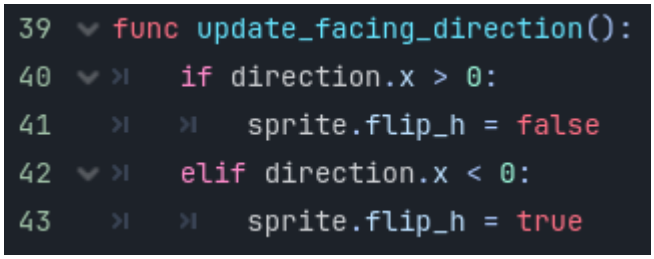


Рисунок 4.12 Функція перегортання спрайту персонажа.

Налаштуємо AnimationTree, для цього додамо вузол AnimationTree до нашого персонажа, та зв’яжемо його з AnimationPlayer у властивостях.

Активуємо наш AnimationTree та тепер нам потрібно зробити CharacterStateMachine, це вузол який допомогає AnimationTree зрозуміти статус персонажа, та відповідно змінювати його.

Обираємо режим AnimationNodeStateMachine у властивостях Tree Root.

У верхній частині екрану відкриваємо редактор AnimationTree, натиснувши на вузол AnimationTree і вибравши вкладку AnimationTree.

Натикаємо правою кнопкою миші в редакторі станів і додаємо нові стани. Називаємо їх jump_start, jump_end, double_jump.

У властивостях кожного стану обираємо відповідну анімацію з AnimationPlayer.

З'єднаємо стани, натиснувши правою кнопкою миші на вузлі одного стану і перетягнувши до іншого стану, щоб створити перехід.

Налаштуємо умови переходу у властивостях переходу (вибравши стрілку між станами).

Додаємо BlendSpace1D, ця функція дозволяє переходити нам між станами.

Перейменовуємо його на move, редагуємо клацнув на іконку малівця, створюємо точки анімацій у центрі та на краях, щоб при значенні 0 програвалися анімація idle, а на 1 та -1 анімація run.

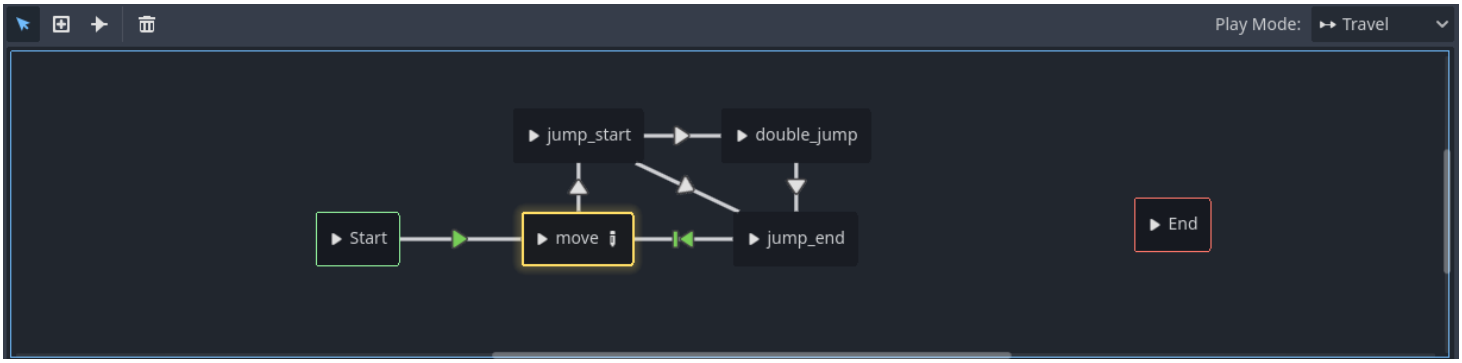


Рисунок 4.12 AnimationTree з прикладом налаштування базових анімацій.

Рисунок 4.13 Налаштування переходів анімацій та статусів персонажа.

Виставляємо параметр Blend на другу опцію.

Повернемося до скрипту нашого гравця, та додаємо функції анімації та CharacterStateMachine.

```
@onready var animation_tree : AnimationTree = $AnimationTree
@onready var state_machine : CharacterStateMachine = $CharacterStateMachine
func _ready():
    animation_tree.active = true
```

Рисунок 4.14 Необхідні функції для роботи AnimationTree.

Створимо новий дочірній Node та назвемо його CharacterStateMachine, та додаємо скрипт до нього.

Створимо змінну states до якої додамо масив State, створимо скрипт State, та назвемо його клас.

Тепер нам потрібно відтворити різні states для кожної позиції ігрока, сперше це буде GroundState який дозволить нам перевірити чи знаходиться гравець на землі, далі створимо AirState та LandingState.

```
1   extends Node
2
3   class_name State
4
5   @export var can_move : bool = true
6
7   var character : CharacterBody2D
8   var next_state : State
9   var playback : AnimationNodeStateMachinePlayback
10
11  func state_process(delta):
12      pass
13
14  func state_input(event : InputEvent):
15      pass
16
17  func on_enter():
18      pass
19
20  func on_exit():
21      pass
22
```

Рисунок 4.15 Конструкція файлу State.gd

Заповнюємо файл State.gd як вказано у малюнку 4.15, та

інші по прикладу з рисунку.

```
@export var landing_animation_name : String = "landing"
@export var ground_state : State

func _on_animation_tree_animation_finished(anim_name):
    if(anim_name == landing_animation_name):
        next_state = ground_state
```

Рисунок 4.16 Структура LandingState.gd

Заповнюємо скрипт CharacterStateMachine функціями switch_states; check_if_can_move; _input; та у функції _ready() робимо перевірку дочірніх скриптів та доєднуємо анімацію. Перевіряємо наші анімації.

3.5 Тестування

Для того щоб краще розуміти роботу нашого CharacterStateMachine зробимо DebugLabel який буде показувати нам у якому state зараз находиться гравець.

Для цього додаємо новий дочірній до сцени Player, Node під назвою Label, та називаємо його StateDebugLabel. Додаємо змінну state_machine та прикрепляємо до функції _process текст “state: ”.

```
@export var state_machine : CharacterStateMachine



# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
    text = "state: " + state_machine.current_state.name
```

Рисунок 4.17 StateDebugLabel

тепер під час тестування ми будемо бачити стан нашого персонажу в коді у реальному часі.

Рисунок 4.18 Мірка стану персонажа CharacterStateLabel

ВИСНОВКИ

В ході виконання дипломної роботи на тему "Розробка відеогри платформера на ігровому движку Godot" було виявлено, що розробка відеоігор є однією з найперспективніших і найбільш динамічно розвиваються сфер індустрії розваг. Це підтверджується постійним зростанням ринку, зростаючою популярністю відеоігор серед широкої аудиторії та розвитком технологій, які дозволяють інді-розробникам створювати ігри.

Розроблено концепцію гри, механіки, візуальний стиль. Це послужило основою для подальшої розробки гри, а також дозволило визначити цілі та завдання проекту.

Створено детальний геймдизайн, це дозволило забезпечити цілісність ігрового процесу та уникнути суперечностей у механіках гри.

Обрано та встановлено стиль графіки, який відповідає тематиці та жанру гри. Піксельна графіка дозволяє досягти високої естетичної якості при збереженні мінімальних апаратних ресурсів. У Godot було створено ігрові сцени, які включали середовище, персонажів і об’єкти. Розробка системи станів персонажа (State Machine): Система станів персонажа ефективно керує анімаціями та переходами між різними діями персонажа, такими як ходьба, біг і стрибок. Інтеграція анімації та логіки: використання AnimationTree у поєднанні зі скриптами для керування анімацією та логікою персонажа забезпечило гнучкість і простоту управління ігровим процесом. Це дало можливість створювати складні анімаційні системи та забезпечувати швидкі переходи між станами.

Godot Engine виявив себе потужним і універсальним інструментом для розробки відеоігор, який підходить як досвідченому, так і початківцям розробникам. Він привабливий для створення як інді-ігор, так і більш масштабних проектів завдяки своєму відкритому коду, багатому функціоналу та активній спільноті. Таким чином, дипломна робота розповіла про те наскільки важливо використовувати структурований підхід для проектування та реалізації ігрових проектів, гра є хорошою основою для подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Game Design Workshop: A Playcentric Approach to Creating Innovative Games" by Tracy Fullerton
2. "Rules of Play: Game Design Fundamentals" by Katie Salen and Eric Zimmerman
3. "Godot Engine Game Development Projects" by Chris Bradfield
4. "GD Script: Godot 4.0 Beginner's Guide" by Alan Thorn
5. Офіційна документація Godot - https://docs.godotengine.org/
6. Онлайн ресурс GDQuest - https://www.gdquest.com/
7. "Level Up! The Guide to Great Video Game Design" by Scott Rogers
8. "The Art of Game Design: A Book of Lenses" by Jesse Schell
9. "Game Testing: All in One" by Charles P. Schultz
10. "Game Engine Architecture" by Jason Gregory