

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
(повне найменування факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка
до дипломного проєкту (роботи)
бакалавра
(ступінь вищої освіти (освітній ступінь))

на тему **РОЗРОБКА ГРИ НА БАЗІ GODOT ENGINE**

Виконав: студентка 4 курсу групи КНТ 511сп
спеціальності _____

123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація) _____
«Комп'ютерна інженерія»

ЕЩЕКГУЛЄВА Г.П.
(ПРІЗВИЩЕ та ініціали)

Керівник **ТЯГУНОВА М.Ю.**
(ПРІЗВИЩЕ та ініціали)

Рецензент **МАЛИЙ О.Ю.**
(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій
 Кафедра «Комп'ютерні системи та мережі»
 Ступінь вищої освіти бакалаврський
 Спеціальність 123 Комп'ютерна інженерія
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерна інженерія
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри КУДЕРМЕТОВ Р.К.

“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Ещектулієва Гозель Піркуліївна

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка гри на базі Godot Engine

керівник проєкту (роботи) к. т. н., доцент, Тягунова Марія Юріївна

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

затверджені наказом вищого навчального закладу від “ 01 ” квітня 2024 року № 105

2. Строк подання студентом проєкту (роботи) 10 травня 2024 року

3. Вихідні дані до проєкту (роботи) інструмент Godot Engine, ігрові механіки, класифікація ігор

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) _____

Аналіз предметної області

Функціонал Godot Engine

Розробка гри

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Тестування гри, Розміщення спрайтів, Вузли для функціонування ігрока, сцена з кнопками, Сцена з доріжкою анімації, Сцена вузлів анімації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ТЯГУНОВА М.Ю., к.т.н., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.03.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Ознайомлення з завданням	01.03.2024 р.	
2	Збір матеріалу для проєкту	03.03.2024 р.	
3	Вивчення варіантів реалізації та вибір варіанту для розробки на базі Godot Engine	12.04.2024 р.	
4	Збір всіх комплектуючих для розробки гри	23.04.2024 р.	
5	Розробка самої гри	25.04.2024 р.	
6	Тестування гри	25.05.2024 р.	
7	Оформлення графічного матеріалу	28.05.2024 р.	
8	Оформлення допуску до захисту	30.05.2024 р.	

Студент (ка)


 (підпис)

Гозель ЕШЕКГУЛІЄВА
 (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)


 (підпис)

Марія ТЯГУНОВА
 (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

ПЗ: 61 стр., 6 табл, 40 рис., 15 джерел.

GDSCRIPT, GODOT ФУНКЦІОНАЛ, СТВОРЕННЯ ГРИ НА БАЗІ GODOT ENGINE, ОСВІТНІ ІГРИ

Об'єкт розробки – гра на базі Godot Engine.

Метою роботи є створити гру на базі Godot Engine.

Проект складається з трьох розділів.

Перший розділ присвячено теоретичній частині, де проведено аналіз завдання та сучасних освітніх ігрових середовищ. У цьому розділі детально розглянуто різні технології для створення ігор, досліджено різновиди ігрових движків для різних платформ та проаналізовано інструментарій Godot.

Другий розділ зосереджено на функціоналі Godot Engine. Тут детально розглянуто можливості та особливості цього движка, що дозволяє зрозуміти його потенціал для створення освітніх ігор. Аналіз функціоналу є критичним для вибору правильного інструменту для реалізації поставленої мети.

Третій розділ описує процес збору ассетів для гри та їх розпакування. У цьому розділі також показано детальне створення гри та написання скриптів на Godot Engine. Виконання цього етапу дозволило створити працездатний прототип освітньої гри, що є ключовим результатом роботи.

ЗМІСТ

Вступ.....	10
1 Аналіз предметної області.....	12
1.1 Цілі розробки гри	13
1.2 Вибір платформ та програмної мови.....	14
1.3 Вибір двигунів для ігор.....	19
1.3.1. Unity	19
1.3.2 Unreal Engine.....	19
1.3.3 Godot	20
2 Функціонал Godot	21
2.1 Можливості Godot.....	22
2.2 Основи GDScript.....	27
2.3 Переваги Godot.....	34
3 Розробка гри	36
3.2 Побудова дочірніх вузлів та розміщення спрайтів.....	41
3.3 Написання скрипта.....	51
3.4 Тестування гри та позбавлення багів	55
Висновки.....	58
Перелік джерел посилання.....	59

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ФС	-	Файлова система;
TCP	-	АНГЛ. Transmission Control Protocol;
HTT	-	АНГЛ. HyperText Transfer Protocol.
CV	-	АНГЛ. Comma-Separated Values;
VCS	-	АНГЛ. Version Control System;
OTF	-	АНГЛ. Open Type File;
TTF	-	АНГЛ. True Type File.

ВСТУП

За короткий час в Україні, відбулися глибокі зміни в різних сферах життя. З нововведень, що змінили стиль життя та й інтереси осіб різного віку, можна виокремити комп'ютеризацію. Крім практичного використання в роботі та побуті, спостерігається захоплення комп'ютерними іграми, що швидко поширюється, особливо серед підлітків і дітей. У зв'язку з цим комп'ютерні ігри потрібно розглядати як своєрідний соціально-психологічний феномен, що сьогодні посідає все помітніше місце в житті людини.

Давно відомо, що в процесі розвитку мислення та формуванні особистості важливу роль відіграє не тільки освітній фактор, але й характер ігрової діяльності. Для дитини гра є провідною діяльністю, тому що саме під час гри вона засвоює значення і способи вживання предметів, а також різні варіанти соціальних відносин.

Гра залишається важливим елементом у житті людини будь-якого віку. Психологи давно та активно вивчають ігрову діяльність.

В ігровій діяльності підліток не тільки заміщає реальні предмети, але й приймає на себе ту чи іншу роль і починає діяти відповідно з нею. Роль у сюжетній грі полягає саме в тому, щоб виконувати обов'язки, що покладаються нею, і реалізовувати права стосовно інших учасників гри. Досвід ігрових та реальних взаємин у сюжетно-рольовій грі лягає в основу особливої властивості мислення, що дозволяє передбачити майбутню поведінку людей і залежно від цього будувати власну поведінку.

Гра є тим видом діяльності, що сприяє практичному освоєнню реального соціального простору - у символічних діях і заміщеннях дитина грає варіанти відносин людей, символічно ідентифікуючи себе з одними персонажами та відокремлюючи від інших.

Ігрова діяльність впливає на формування довільних психічних процесів. Умови гри вимагають зосередженості на змісті дій і сюжеті, на діючих особах чи

предметах, включених до ігрової ситуації. Ігрова ситуація впливає на мислення і психіку дитини, підлітка і дорослого. Гра сприяє розвитку рефлексії, оскільки в цьому процесі виникає реальна можливість контролювати, як виконується будь-яка дія, що входить у процес спілкування. У рольовій грі формується здатність осмислювати свої власні дії, передбачати реакцію інших людей.

В останні десятиліття поступово все більшу частку ігор складають ігри з комп'ютером, в яких людина реалізує діяльність щодо керування різними процесами ігрового простору. Вплив комп'ютерних ігор на поведінку людей не викликає сумніву.

Позитивний потенціал більшості комп'ютерних ігор реалізується далеко не завжди. І це залежить здебільш не від самої гри, а від людини, що грає, від того, який мотив переважає при включенні до гри. Крім основного мотиву розваги, гра може реалізувати інші мотиви. У залежності від мотивів другого плану можуть формуватися абсолютно різні для різних гравців навички та уміння. Реалізація мотиву тренінгу призводить до формування навичок у сфері, що тренується, а мотив компенсації внутрішніх проблем скоріш матиме результатом формування механізмів психологічного захисту. У зв'язку з розмаїттям мотивів гра може сприяти як підготовці до зустрічі з реальністю, так і втечі від неї.

Мета роботи – створити гру на базі Godot Engine. Для досягнення поставленої мети необхідно виконати наступні завдання:

- проаналізувати предметну область;
- обрати необхідно програмне забезпечення для розробки;
- розробити безпосередньо гру на базі Godot Engine.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Розробка комп'ютерних ігор - процес створення комп'ютерних ігор (відеоігор). Розробкою відеоігор займається розробник, який може бути представлений як через одного чоловіка, так і фірмою. Зазвичай великомасштабні комерційні ігри розробляються командами розробників в межах компанії, що спеціалізується на іграх для персонального комп'ютера або консолей. Як правило, розробку фінансує інша, більш велика компанія-видавець, яка по закінченню розробки займається виданням гри і пов'язаними з ним витратами. Рідше компанії-видавці можуть містити внутрішні команди розробників, або ж компанія-розробник може розробляти гри за свій рахунок і поширювати їх без участі видавців, наприклад, засобами цифрової дистрибуції казуальні ігри

Розробка найбільш крупнобюджетних ігор («AAA-гри») може коштувати десятки мільйонів доларів США, причому протягом останніх десятиліть ці бюджети безперервно росли, як і чисельність команд розробників і терміни розробки. Середній бюджет AAA-проекту, як правило, випускаються найбільшими компаніями-видавцями, що продаються на фізичних носіях і нерідко входять до складу відомої серії з декількох ігор - коливається від 18 до 24 млн. Дол. Крупнобюджетного гра для двох платформ - Xbox 360 і PlayStation 3 - обходилася в 2012 році в середньому в 20 мільйонів доларів, і для того, щоб вона окупилася, потрібно було продати близько двох мільйонів копій.

Перш ніж заглиблюватися в секрети маркетингу і розробки, давайте поговоримо трохи докладніше про те, що ж таке гіпер-казуальна гра і які у неї особливості.

Гіпер-казуальна гра - це жанр, який має на увазі миттєве занурення в ігровий процес, з дуже простої повторюється механікою дій гравця і без тимчасових обмежень.

У гіпер-казуалки є й інші ігрові механіки:

- таймінгова механіка (торкання екрану в певний час);

- механіка укладання блоків (збір об'єктів в вежу або стіну);
- механіка спритності (робиться наголос на швидкість реакції);
- механіка сходження / падіння (управління рухом об'єкта);
- механіка ухилення.

Гіпер-казуальні ігри стали трендом на ринку мобільних додатків. У 2019 число активних користувачів в день в топових гіпер-казуалки склало 94 тисяч. І це найвищий показник серед інших мобільних ігор. У 2020 році гіпер-казуальні ігри, судячи з усього, не втрачають популярність: тільки в першому кварталі число установок по всьому світу виросло на 103%. А ще гравці стали проводити в грі на 72% більше часу, ніж раніше.

Адже, на перший погляд, можна подумати, що створення гіпер-казуалки - справа всього декількох тижнів. Але незважаючи на уявну простоту, гіпер-казуальні ігри таять в собі чимало підводних каменів, про які краще знати заздалегідь. В іншому випадку ви ризикуєте витратити час і сили даремно, так і не показавши гру вашої цільової аудиторії.

Даний проєкт відноситься до казуальної гри. Зважаючи на своє призначення такі ігри мають достатньо прості правила й не вимагають від користувача добрих навичок роботи з комп'ютером.

Багато подібних ігор характеризуються також яскравою привабливою графікою та мінімумом тексту. Простими казуальними іграми можливо назвати ті, які входять у стандартне постачання дистрибутивів операційних систем.

1.1 Цілі розробки гри

Основна ідея гіпер-казуальних ігор - запропонувати гравцям миттєву можливість почати гру, відразу ж після завантаження з стору. Не потрібно входити в систему, реєструватися або створювати персонаж - просто натискаєш «Грати» і починаєш проходити рівні.

Гіпер-казуальні ігри, як правило, мають мінімалістичний і привабливий дизайн, щоб ще більше зацікавити користувача. Крім того, вони припускають прогресуючий і швидкий геймплей, щоб користувачеві хотілося продовжувати грати якомога довше.

Знання цільової аудиторії - це не тільки інформація про те, якою мовою говорять користувачі. Потрібно враховувати й інші фактори, такі як рівень інтересу до гіпер-казуальних ігор і бажані ігрові жанри в цілому.

А ще переваги всередині самого жанру гіпер-казуалок теж можуть відрізнятися в різних країнах. Наприклад, гравцям в одному регіоні може подобається механіка "тапов" по екрану, а іншим - механіка укладання блоків.

Також важливо знати про локалізації - не просто випустити гру, потрібно щоб аудиторія полюбила її. Тому важливо витратити час і сили на професійну локалізацію - і реакція геймерів вас точно порадує.

Гіпер-казуальні ігри не містять великої кількості тексту. І це може посіяти думку про те, що їх локалізація не дуже важлива. Але насправді це зовсім не так.

Незважаючи на мінімалістичний дизайн і відносно невеликий обсяг текстів в гіпер-казуальній грі, все одно доведеться витратити певну кількість часу на всілякі переклади. І якщо взяти ідеальну бізнес-модель, в якій ви створюєте більше десятка ігор в надії, що одна або дві з них принесуть дохід і окупляться, це означає, що внутрішньоігрові тексти доведеться переводити дуже оперативно.

1.2 Вибір платформ та програмної мови

Інший важливий параметр при виборі платформи - гри. Велика частина проєктів доступна на Xbox One, PlayStation 4 і ПК.

Також є ігри, які виходять тільки на конкретних платформах: наприклад, в екшен про богів скандинавської міфології God of War можна пограти тільки на PlayStation 4, а в фантастичний шутер Halo 5 - тільки на Xbox One.

Крім цього, деякі ігри випускають тільки для консолей (як бандитський вестерн Red Dead Redemption 2), а інші виходять на певних консолях і ПК (в основному це проєкти, що випускаються Microsoft, на зразок гонки Forza Horizon 4).

Ексклюзиви на ПК - це в основному стратегії і гри різних жанрів від незалежних розробників. Наприклад, фантастичні стратегії Homeworld: Deserts of Kharak і Wahrammer 40000: Dawn of War 3 вийшли тільки на комп'ютерах. Як і мультиплеєрний шутер World War 3 або тактичний екшен Insurgency.

Ігри, доступні тільки на PlayStation 4, - це, як правило, екшени від третьої особи з упором на сюжет і геймплей. За п'ять років існування консолі Sony накопичила багато подібних проєктів.

Серед ексклюзивів Xbox One - серії шутерів Gears of War і Halo, гоночна франшиза Forza, екшени від третьої особи Quantum Break і Sunset Overdrive. Правда, практично всі нові ігри для Xbox доступні також і на ПК, наприклад Forza Horizon 4 і Quantum Break.

Виключно на Switch виходять в основному гри від Nintendo: Super Mario Odyssey, Mario + Rabbids Kingdom Battle, The Legend of Zelda: Breath of the Wild, Super Smash Bros., Mario Kart 8 Deluxe.

Ці ігри відрізняються один від одного жанрами, персонажами і сюжетами, але їх об'єднує одне: приголомшливе увагу Nintendo до деталей в ігровому процесі. Кожну свою гру компанія намагається зробити максимально веселою і захоплюючою, керуючись при розробці філософією «гра повинна приносити задоволення і дивувати». А ще практично всі проєкти від цієї студії мають рейтинг 3+ або 6+.

По першу чергу слід вибирати не мова, а область, в якій ви збираєтеся будувати кар'єру. Ігри, в бізнес-процеси, розробка мобільних додатків, робота з великими даними - кожен напрямок вимагає свого набору мов. Вибирайте один з популярних мов в області.

При цьому важливо розуміти, що однією мовою програмування справа не обмежиться: з часом доведеться освоїти два-три близьких, які допоможуть

ефективніше вирішувати «дорослі» завдання по розробці. Якщо володієте англійською, тут можна пройти класний тест для підбору ідеально підходить вам мови.

Новачки часто не звертають уваги на важливі чинники і вже в ході вивчення мови розуміють, що краще було б вибрати інший.

Знайомство з програмуванням зручно починати з простих мов з доброзичливим синтаксисом. Команди в такій мові задаються англійськими словами, зрозумілі без словника і легко запам'ятовуються, а код нагадує пропозиції тексту.

Чим далі від повсякденного англійської логіка мови і назви команд, тим складніше буде втягнутися в роботу. Лайфхак: якщо назва мови знайоме людині без досвіду в ІТ, він, швидше за все, досить простий для вивчення.

Крім цього, важливо щоб вибрана мова подобався вам візуально - саме його ви будете бачити більшу частину дня. Уже згаданий вище тест пропонує порівняти зразки коду на популярних і вибрати той, до якого більше лежить душа

Ще один важливий момент - кількість доступної інформації про конкретній мові. Якщо документації мало, навчання може стати дуже складною. Детальніше про це розповім в пункті про рівень розвитку мови.

У процесі вивчення мови у вас обов'язково виникнуть питання. Тому важливо заздалегідь з'ясувати, наскільки відкрито до спілкування співтовариство розробників навколо конкретного мови. Для adeptів старіших мов буває характерна деяка зарозумілість і уїдливість, які ускладнюють шлях новачка.

Тому, перш ніж ставати частиною ком'юніті, рекомендую приміряти його на себе. Почитайте, поспостерігайте, сходіть на кілька зустрічей: підходяще по духу оточення буде підтримувати і мотивувати вас в процесі навчання.

```
//параметры нашего хождения<br/>
var speed = 6.0;<br/>
var jumpSpeed = 8.0;<br/>
var gravity = 20.0;<br/>
<br/>
//внутренние переменные<br/>
private var moveDirection = Vector3.zero;<br/>
private var grounded : boolean = false;<br/>
<br/>
//обновление по таймеру<br/>
function FixedUpdate() {<br/>
    //мы на земле<br/>
    if (grounded) {<br/>
        //строим вектор перемещения<br/>
        moveDirection = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));<br/>
        //переносим в глобальные координаты<br/>
        moveDirection = transform.TransformDirection(moveDirection);<br/>
        //увеличиваем пропорционально скорости<br/>
        moveDirection *= speed; <br/>
    }<br/>
    //если велено прыгать - прыгаем<br/>
    if (Input.GetButton ("Jump")) {<br/>
        moveDirection.y = jumpSpeed;<br/>
    }<br/>
}<br/>
<br/>
```

Рисунок 1.1 – Мова програмування C#

```
import sys
for n in range(int(sys.argv[1]), int(sys.argv[2])):
    if n % 3 == 0 and n % 5 == 0:
        print("fizzbuzz")
    elif n % 3 == 0:
        print("fizz")
    elif n % 5 == 0:
        print("buzz")
    else:
        print(n)
```

Рисунок 1.2 – Скрипт Python

```

extends Node

# Declare the class name here
class_name ScriptName, "res://path/to/optional/icon.svg"

func _ready():
    var this = ScriptName          # reference to the script
    var cppNode = MyCppNode.new()  # new instance of a class named MyCppNode

    cppNode.queue_free()

```

Рисунок 1.3 – Скрипт Godot

```

#!/bin/bash

msg_file=/dev/null
#msg_file=/dev/stdout

tmp_path=$HOME"/.cache/c++/"
mkdir -p $tmp_path
tmp_file=$1".c++"
exe_file=$1".bin"
if test $1 -nt $tmp_path$exe_file; then
    echo "Need to recompile.." > $msg_file
    tail -n +2 $1 > $tmp_path$tmp_file
    eval "g++ -o $tmp_path$exe_file $tmp_path$tmp_file > /dev/null 2>&1"
    if [ $? -eq 0 ]
    then
        echo "Compiled ok" > $msg_file
    else
        echo "Compile error" > $msg_file
        exit 255
    fi
fi
eval "$tmp_path$exe_file $@"

```

Рисунок 1.4 – Скрипт C++

1.3 Вибір двигунів для ігор

Список сучасних двигунів:

- unity;
- unreal Engine;
- godot.

1.3.1 Unity

Перша перевага, на мій погляд - сама мова C #. Дана мова високорівнева і дозволяє програмісту легко увійти в розробку гри. Це важливий момент, тому що на відміну від інших движків, де використовується мова C ++, в C # є багато елементів і прийомів, які вже реалізовані, і програмісту потрібно тільки скористатися ними.

Друга перевага - кроссплатформенність, тобто один і той же код, написаний на движку Unity, з мінімальними змінами може бути перенесений на різні платформи (PC, Mac, Android, iOS, Web, ігрові консолі). Це величезний плюс, який скорочує час на розробку гри в кілька разів.

Третя перевага - гарне Community. Це означає, що у різних функцій движка є чіткий опис з прикладами на сайті розробника, звернутися до якого можна в будь-який момент. Якщо щось все ж залишилося незрозумілим, служба підтримки обов'язково відповість на виниклі питання.

І четвертим перевагою є Asset Store, де є величезна кількість різних плагінів і ресурсів для створення гри. Зрозуміло, якісь з них безкоштовні, якісь платні, але всі вони зібрані в одному місці зі зручним пошуком і можливістю завантажити, інтегрувати і отримати відразу робочий функціонал.

1.3.2 Unreal Engine

Unreal Engine надає кращі інструменти для створення мультяшної або реалістичною графіки. Програма відома тим, що саме її використовують відомі студії для випуску масових AAA-ігор.

В Unreal Engine є схеми Blueprints (функції для візуального програмування за допомогою перетягування об'єктів і створення зв'язків між ними): навіть художники без академічних знань зможуть використовувати движок для базового програмування моделей, освітлення і анімації. Також ця функція підходить для прототипування рівнів (реалізації базових ігрових локацій за допомогою примітивних моделей). Але для більш складного функціоналу буде потрібно мову програмування C ++.

У Unreal Engine великий набір інструментів і функцій, яких немає в інших движунах аналогічного типу. Наприклад, тут є візуальні редактори (перегляд сцен в режимі реального часу).

1.3.3 Godot

У Godot в цілому репутація швидше 2д движка, яка опрацьована досить добре, але і не так давно з'явилися 3д можливості дозволяють робити тривимірні ігри.

Що стосується мов, то якщо не розглядати низькорівневий спосіб, найбільш ходові варіанти - це скриптові мови GDScript і C #, а також візуальний скриптинг для чогось простого. GDScript краще інтегрований в движок, має більше прикладів, не вимагає запуску зовнішнього середовища, а в плані загальної структури тут буде відбуватися все те ж, що у варіанті C # - оголошення змінних, виклик функції ініціалізації, цикл процесів, цикл фізичних процесів і так далі.

Так що вибір GDScript в якості основного скриптового мови розробки має сенс. Пересідаючи на місцевий C # отримаємо хіба що більшу акуратність і подробиця записи, втрачаючи в лаконічності, але посилюючи розбірливість і контроль над ситуацією - фігурні скобочки замість використання табуляції, позначки кінця рядка, більш формальну типізацію.

2 ФУНКЦІОНАЛ GODOT

Godot Engine - багатофункціональний, крос-платформний ігровий движок, для створення 2D і 3D ігор. Він надає широкий набір поширених інструментів, таким чином, що користувачі можуть зосередитися на створенні ігор без необхідності винаходити колесо. Ігри можуть бути експортовані в один клік на ряд платформ, включаючи головні настільні платформи (Linux, macOS, Windows), а так само мобільні (Android, IOS) і веб-орієнтовані платформи (HTML5).

Godot абсолютно безкоштовний, з відкритим вихідним кодом згідно з ліцензією MIT. Ніяких ліцензійних платежів тощо. Розробка Godot повністю незалежна і підтримується спільнотою, даючи можливість користувачам створити свій інструмент, щоб він відповідав їхнім очікуванням. Це підтримується некомерційною організацією Software Freedom Conservancy.

Для подальшого вивчення можливостей ми рекомендуємо більш докладне вивчення документації, особливо навчального розділу Покрокове навчання.

Спільнота постійно допрацьовує, виправляє, редагує і переробляє цю документацію. Процес виглядає так: текстові файли на мові розмітки reStructuredText редагуються, а потім, за допомогою інструментів Sphinx і ReadTheDocs перетворюються в статичний веб-сайт або окремий документ.

Документація розбита на п'ять розділів різного обсягу по досить зрозумілим принципам:

- секція Основне містить вступ, інформацію по движку, його історію, інформацію про ліцензії, авторів, і інше. Вона також містить Часті питання;
- розділ Початок роботи є головною причиною для існування даної документації, оскільки він містить всю необхідну інформацію про використання движка для створення ігор. Він починається з покрокового навчання, яке повинно стати відправною точкою для всіх нових користувачів;

- секцію Уроки можна читати в міру необхідності, вбуди-якому порядку. Вона містить керівництва і документацію по конкретним можливостям движка;
- секція Розробка призначена для просунутих користувачів і учасників розробки движка, і включає в себе інформацію про те, як компілювати движок, розробляти модулі на C ++ або плагіни для редактора;
- секція Спільнота дає інформацію щодо участі в розробці движка і життя його спільноти, наприклад як повідомляти про помилки, допомагати писати документацію, та інше. Вона також направляє на різні канали спільноти такі як IRC і Discord і містить список рекомендованих посібників від третіх осіб поза основною документації;
- класи движка це документація по Godot API, яка також доступна безпосередньо з редактора скриптів у двигуні. Вона генерується автоматично з файлу в основному кодовому сховища, згенеровані файли документації не повинні бути змінені. Дивіться Contributing to the class reference для уточнення.

2.1 Можливості Godot

Платформи:

- може запустити як редактор, так і експортовані проекти;
- windows 7 і вище (64-бітна і 32-бітна);
- macOS 10.12 і вище (64-бітна, x86 і ARM);
- linux (64-бітна або 32-бітна, x86 або ARM);
- двійкові файли статично пов'язані і можуть працювати в будь-якому дистрибутиві, якщо вони скомпільовані в досить старому базовому дистрибутиві;
- офіційні виконавчі файли скомпільовані на Ubuntu 14.04;
- запускає експортовані проекти;
- android 4.4 і вище;

- ios 10.0 і вище;
- html5 за допомогою WebAssembly (Firefox, Chrome, Edge, Opera);
- консолі;
- godot прагне бути максимально незалежним від платформи і його відносно легко переносити на нові платформи;
- редактор;
- редактор дерева сцени;
- редактор скриптів;
- підтримка зовнішніх редакторів скриптів, таких як Visual Studio Code або Vim;
- gsdscript відгадчик;
- не підтримується налагодження в потоках;
- інструменти моніторингу продуктивності;
- жива перезавантаження скриптів;
- редагування сцени на льоту;
- зміни будуть відображені в редакторі і будуть збережені після закриття запущеного проекту;
- віддалений інспектор;
- зміни не будуть відображені в редакторі і не будуть збережені після закриття запущеного проекту;
- реплікація камери в реальному часі;
- перемістіть камеру в редакторі і подивіться результат в запущеному проєкті;
- використовуйте редактор на десятках мов, наданих спільнотою;
- модулі редактора можна завантажити з бібліотеки ресурсів для розширення функціональності редактора;
- створіть свої власні плагіни, використовуючи GDScript, щоб додати нові функції або прискорити робочий процес;
- завантажте проєкти з бібліотеки ресурсів в диспетчер проєктів і імпортуйте їх безпосередньо;

- доступні два рендерера:
 - openGL ES 3.0 рендерер (на настільних платформах використовується OpenGL 3.3);
 - високоякісне зображення. Рекомендується для настільних платформ;
 - openGL ES 2.0 рендерер (на настільних платформах використовується OpenGL 2.1);
 - рекомендується на мобільних і веб-платформах;
 - візуалізація ліній, полігонів і спрайтів;
 - інструменти високого рівня для малювання ліній і полігонів, такі як Polygon2D і Line2D;
 - animatedSprite в якості помічника для створення анімованих спрайтів;
 - підтримка псевдо-3D шляхом автоматичного дублювання шару кілька разів;
 - 2D освітлення з картами нормалей;
 - тверді або м'які тіні;
 - візуалізація шрифтів з використанням растрових зображень (BitmapFont) або растеризація з використанням FreeType (DynamicFont);
 - растрові шрифти можна експортувати за допомогою таких інструментів, як BMFont;
 - dynamicFont підтримує як монохромні, так і кольорові шрифти. Підтримуються формати TTF і OTF;
 - dynamicFont підтримує додаткові контури шрифтів з регульованою шириною і кольором;
 - підтримка передискретизації шрифтів для збереження різкості шрифтів при більш високій роздільній здатності;
 - частинки на базі GPU з підтримкою призначених для користувача шейдерів частинок;
 - частинки на базі процесора;
- 2D-камера з вбудованою функцією згладжування і перетягування полів:
- вузол Path2D для подання шляху в 2D просторі;

- шляхи можуть бути намальовані в редакторі або згенеровані процедурно;

- вузол PathFollow2D, щоб змусити вузли слідувати по Path2D;
- допоміжний клас 2D геометрії;
- вузол Line2D для малювання текстурованих 2D ліній;
- фізичні тіла;
- статичні тіла;
- жорсткі тіла (rigid bodies);
- кінематичні тіла (kinematic bodies);
- суглоби;
- області для виявлення тіл, що входять або виходять з них;
- виявлення зіткнень;
- вбудовані фігури: лінія, коробка, коло, капсула;
- полігони зіткнення (можуть бути намальовані вручну або згенеровані

з спрайту в редакторі) ;

- об'єктно-орієнтована шаблон проектування зі скриптами, які розширюють вузли;

- сигнали і групи для обміну даними між скриптами;
- підтримка міжмовних сценаріїв;
- багато типів даних 2D і 3D лінійної алгебри, такі як вектори і

трансформації;

- високорівнева інтерпретована мова з необов'язковою статичної типізацією;

- синтаксис, натхненний Python;
- підсвічування синтаксису передбачена на GitHub;
- застосування потоків для виконання асинхронних дій або

використання декількох процесорних ядер;

- упакований в окремий двійковий файл для зменшення розмірів файлів і залежностей;

- використовує Mono 6.x;

- повна підтримка синтаксису і можливостей C # 7.0 ;
- підтримує всі платформи;
- для використання функціональних можливостей IDE рекомендується використовувати зовнішній редактор;
- мова візуальних сценаріїв на основі графів;
- найкраще працює при використанні для певних цілей (наприклад, логіки для конкретних рівнів), а не в якості мови для створення цілих проектів;
- вузли;
- кнопки;
- прапорці, кнопки-перемикачі, перемикачі;
- введення тексту за допомогою LineEdit (один рядок) і TextEdit (кілька рядків) ;
- випадають меню за допомогою PopupMenu і OptionButton;
- смуги прокрутки;
- мітки;
- richTextLabel для тексту, відформатованого за допомогою використання BBCode;
- дерева (також можуть використовуватися для подання таблиць).
- контейнери (горизонтальні, вертикальні, сітка, центр, поля, перетаскується роздільник) ;
- елементи управління можна обертати і масштабувати;
- локалізація;
- повна підтримка Unicode, включаючи емодзі;
- збереження рядків локалізації, використовуючи CSV або gettext
- використовуйте локалізовані рядки в своєму проекті автоматично в елементах графічного інтерфейсу або за допомогою функції;
- підтримка письма справа наліво і верстка тексту планується в Godot 4.0 ;
- вихід моно, стерео, 5.1 і 7.1;
- непозиційних і позиційне відтворення в 2D і 3D;

- додатковий ефект Доплера в 2D і 3D;
- підтримка перенаправляє audio buses і ефектів з десятком внесених змін;
- вузол Listener3D для прослуховування з позиції, відмінної від позиції камери в 3D;
- аудіо вхід для запису мікрофонів;
- midi вхід.

2.2 Основи GDScript

GDScript це високорівнева, динамічно типізований мову програмування. Він використовує синтаксис схожий на Python (блоки виділяються табуляцією і безліч ключових слів ідентичні). Його мета бути оптимізованим і щільно інтегрованим в движок Godot, дозволяючи досягти високої гнучкості в створенні контенту і інтеграції.

Приклад GDScript

Деяким людям простіше навчитися, глянувши на синтаксис, тому ось простий приклад того, як виглядає GDScript.

```

# A file is a class!

# Inheritance

extends BaseClass

# (optional) class definition with a custom icon

class_name MyClass, "res://path/to/optional/icon.svg"

# Member variables

var a = 5
var s = "Hello"
var arr = [1, 2, 3]
var dict = {"key": "value", 2: 3}
var typed_var: int
var inferred_type := "String"

```

Рисунок 2.1 – Приклад GDScript

```

var a = 5
var s = "Hello"
var arr = [1, 2, 3]
var dict = {"key": "value", 2: 3}
var typed_var: int
var inferred_type := "String"

# Constants

const ANSWER = 42
const THE_NAME = "Charly"

# Enums

enum {UNIT_NEUTRAL, UNIT_ENEMY, UNIT_ALLY}
enum Named {THING_1, THING_2, ANOTHER_THING = -1}

# Built-in vector types

var v2 = Vector2(1, 2)
var v3 = Vector3(1, 2, 3)

```

Рисунок 2.2 – Приклад GDScript

Якщо у вас вже був досвід роботи з статично типізованими мовами, такими як C, C++ або C#, але ви ніколи раніше не використовували динамічно типізовану мову, рекомендується прочитати це керівництво: `doc_gdscript_efficient`.

Далі слід загальний огляд GDScript. Детальну інформацію, наприклад, про доступні методи для масивів або інших об'єктів, слід шукати в описах пов'язаних класів.

Будь-яка рядок, яка обмежує себе алфавітними символами (від а до z і від А до Z), цифрами (від 0 до 9) і `_`, кваліфікується як ідентифікатор. Крім того, ідентифікатори не повинні починатися з цифри. Ідентифікатори чутливі до регістру (`foo` відрізняється від `FOO`).

Нижче наведено список ключових слів, які підтримуються мовою. Оскільки ключові слова є зарезервованими словами (токенами), вони не можуть використовуватися в якості ідентифікаторів. Оператори (наприклад, `in`, `not`, `and` або `or`) і імена вбудованих типів, перераховані в наступних розділах, також зарезервовані.

Ключові слова визначені в токенизаторе GDScript, якщо ви хочете поглянути під капот

Вбудовані типи розподілені по стека. Вони передаються як значення. Це означає, що копія створюється на кожному привласненні або при передачі їх в якості аргументів функцій. Єдиним винятком є `Array` і `Dictionaries`, які передаються по посиланню і є загальними.

(Не `PoolArray` типу `PoolByteArray`, але вони теж передаються як значення, тому враховуйте це при вирішенні, що саме потрібно використовувати для проекту)

Вбудовані типи контейнерів

Загальна послідовність будь-яких типів об'єктів, включаючи інші масиви або словники (див. Нижче). Масив може змінюватися динамічно. Масиви індексуються, починаючи з індексу 0. Починаючи з Godot 2.1, індекси можуть бути негативними, як і в Python, вважаючи з кінця.

```

var arr = []
arr = [1, 2, 3]
var b = arr[1] # This is 2.
var c = arr[arr.size() - 1] # This is 3.
var d = arr[-1] # Same as the previous line, but shorter.
arr[0] = "Hi!" # Replacing value 1 with "Hi!".
arr.append(4) # Array is now ["Hi!", 2, 3, 4].

```

Рисунок 2.3 – Array

Масиви в GDScript виділяються в пам'яті лінійно для підвищення швидкості роботи. Однак великі масиви (більш десятків тисяч елементів) можуть привести до фрагментації пам'яті. Якщо це проблема, то доступні спеціальні типи масивів. Вони приймають тільки один тип даних. Вони уникають фрагментації пам'яті і також споживають менше пам'яті, але є атомними і, як правило, працюють повільніше, ніж звичайні масиви. Тому їх рекомендується використовувати тільки для великих наборів даних:

- poolByteArray: Масив байтів (цілі числа від 0 до 255).
- poolIntArray: Масив цілих чисел.
- poolRealArray: Масив чисел з плаваючою точкою.
- poolStringArray: Масив рядків.
- poolVector2Array: Масив об'єктів типу Vector2.
- poolVector3Array: Масив об'єктів типу Vector3.
- poolColorArray: Масив об'єктів типу Color.

```

extends Node

# A signal named health_depleted.
signal health_depleted

```

Рисунок 2.4 – Signal

Можливо підключати ці сигнали до методів точно так же, як ви підключаєте вбудовані сигнали вузлів типу Button або Rigidbody.

У прикладі нижче ми з'єднали сигнал `health_depleted` від вузла `Character` з вузлом `Game`. Коли вузол `Character` відправляє сигнал, викликається ігровий вузол `_on_Character_health_depleted`:

```
# Game.gd

func _ready():
    var character_node = get_node('Character')
    character_node.connect("health_depleted", self, "_on_Character_health_depleted")

func _on_Character_health_depleted():
    get_tree().reload_current_scene()
```

Рисунок 2.5 - Character з вузлом Game

Дозволяє відправляти разом з сигналом стільки аргументів, скільки хочете.

Ось приклад, де це може бути корисним. Припустимо, ми хочемо, щоб смуга здоров'я на екрані реагувала на зміни здоров'я анімацією, але ми хочемо, щоб користувальницький інтерфейс був відділений від гравця в нашому дереві сцен.

У скрипті `Character.gd` ми визначаємо сигнал `health_changed` і відправляємо його разом з `Object.emit_signal()`, і вузол `Game` вище по дереву сцен ми з'єднуємо з `Lifebar` використовуючи метод `Object.connect()`:

```

# Character.gd

...
signal health_changed

func take_damage(amount):
    var old_health = health
    health -= amount

    # We emit the health_changed signal every time the
    # character takes damage.
    emit_signal("health_changed", old_health, health)

...

```

Рисунок 2.6 - Сигнал health_change

Сигнали це спосіб відправки повідомлень від об'єкта, на які можуть реагувати інші об'єкти. Щоб створити власні сигнали для класу, використовуйте ключове слово `signal`.

Співпрограми з проміжним поверненням (`yield`)

GDScript пропонує підтримку співпрограми через вбудовану функцію `yield`. Виклик `yield ()` негайно поверне з поточної функції з поточним замороженим станом цієї ж функції як повертається значення. При виклику `resume ()` на даному результаті об'єкт продовжить виконання і поверне все, що повертає функція. Після відновлення стан об'єкта стає недійсним. Ось приклад:

```

func my_func():
    print("Hello")
    yield()
    print("world")

func _ready():
    var y = my_func()
    # Function state saved in 'y'.
    print("my dear")
    y.resume()
    # 'y' resumed and is now an invalid state.

```

Рисунок 2.7 - Виклик yield

Також можна передавати значення між функціями `yield ()` і `resume ()`, наприклад:

```
func my_func():
    print("Hello")
    print(yield())
    return "cheers!"

func _ready():
    var y = my_func()
    # Function state saved in 'y'.
    print(y.resume("world"))
    # 'y' resumed and is now an invalid state.
```

Рисунок 2.8 - Між функціями `yield ()` і `resume ()`

Справжня міць використання `yield` полягає в їх поєднанні з сигналами. `yield` може приймати два аргументи, об'єкт і сигнал. Коли сигнал буде отриманий, виконання буде відновлено. Ось кілька прикладів:

```
# Resume execution the next frame.
yield(get_tree(), "idle_frame")

# Resume execution when animation is done playing.
yield(get_node("AnimationPlayer"), "animation_finished")

# Wait 5 seconds, then resume execution.
yield(get_tree().create_timer(5.0), "timeout")
```

Рисунок 2.9 - Співпрограми і сигнали

2.3 Переваги Godot

З метою розробки власної кооперативної комп'ютерної гри використано Godot Engine – безкоштовний рушій з відкритим кодом, що дозволяє створювати ігри для більшості сучасних популярних платформ з використанням спільної кодової бази та має широкий спектр ефективних інструментів для роботи в трьох (3D) та двовимірних (2D) просторах.

Головною перевагою Godot Engine є його повна безкоштовність. Рушій Godot Engine для розробки мережевої частини гри має невелику кількість варіантів. Наразі існують два доступні варіанти: використання вбудованого функціоналу – Godot Engine має вбудоване API для роботи з мережевими функціями через HTTP, UDP, TCP та SSL протоколи; створення власної мережевої системи на мові C#.

Компактний двигун з величезними можливостями але, знову ж таки, поки поступається тому ж Unity в плані різноманіття готових рішень і можливостей. Хоча, різних неофіційних рішень для Годо вже написано досить багато і завдяки відносній простоті / швидкості імплементації нового функціоналу, не сказати, щоб це було якоюсь проблемою на даний момент.

У Годо більше репутація 2д-движка, завдяки різноманіттю опрацьованих інструментів саме для 2д, але вони є додатковим плюсом для 3д - куди простіше робити ігровий UI. На поточний момент Годо в своєму розвитку дійшов до стабільної 3.2.2 версії, ось ось вийде 3.2.3, яка поки в стані реліз кандидатів.

Движок не вимагає потужного заліза для 3д-графіки і видає цілком пристойну картинку. Готових тривимірних інструментів не величезна кількість, але реалізовані як раз одні з найбільш потрібних, корисних і універсальних. Приблизно те ж стосується і оптимізацій. Наприклад, в движку реалізований звичайний frustrum culling, що відтинає геометрію поза зоною видимості камери.

А ось для якого-небудь occlusion culling (щоб не брати до уваги закриті стінами об'єкти) реалізацію доведеться придумувати самостійно (що не так вже й складно, особливо в якихось точкових місцях, та й не в кожній грі потрібно).

На відміну від Unity движок не буде за вас Батче геометрію (принаймні в gles3 режимі цього поки немає) і нарізати террейн на чанкі (тому як офіційної реалізації террейн в Годо поки немає, але в майбутніх версіях хочуть включити існуюче неофіційне рішення до складу релізу).

Але ви можете робити це самостійно - заздалегідь зшити потрібні шматки в 3д-пакеті в один меш або побити свій террейн на дрібні частини, щоб камера не малювала його цілком, навіть коли в кадрі видно лише його край. Або підшукати якусь реалізацію в місцевому невеликому сторі, наприклад, додати в свій проєкт готове рішення для террейн, де все саме розіб'ється на шматки-чанкі, як в стандартному Unity-террейнов.

Інтерфейс движка, до речі, досить продуманий і кастомізуємий (хоча є деякі негнучкі елементи). Користуватися ним в цілому зручно. Підтримуваних мов досить, для різного рівня занурення. Тут і C ++ і C # і досить зручний внутрішній GDScript, який запускається прямо всередині редактора, не вимагаючи запуску окремої середовища. Візуальний скриптинг теж присутній (плюс дуже класний інструмент для запису анімацій), так що без знання програмування в Годо теж цілком можна жити - якусь мінімальну логіку сконструювати, щось заанімувати.

Мала вага додатки, мультиплатформеність, швидкість розробки, простота імплементації різних сторонніх рішень - теж важливі плюси движка. Є два варіанти рендеру - gles2 і gles3, обидва підтримують 3д, але в першому воно простіше і в цілому він більше підходить для 2д і мобілок. Gles3 дає більш просунутий рівень графіки, якась частина мобільних пристроїв його теж підтримує.

До кінця року з'явиться 4-я версія Godot, що використовує vulkan. Там графіка зробить крок ще далі, з'явиться більш просунуте реалтайм освітлення і

додаткові оптимізації. Власне, неофіційні білди четвертої версії вже є і можна їх пробувати, хоча працюють частково і нестабільно.

3 РОЗРОБКА ГРИ

3.1 Етапи створення гри

Ігровий Ассет (англ. Game asset) - це будь-який готовий елемент, який можна використовувати для створення гри. Наприклад, до Ассет відносяться: текстури, фонові музика, 2D або 3D-моделі, анімації, ефекти частинок і багато іншого.

Основна функція Ассет.

Готових елементів гри, - заощадити час і ресурси розробника на створення ігрового контенту. Наприклад, якщо потрібно, щоб бос страхотливо гарчав на початку битви, то набагато простіше взяти вже готовий записаний рик, ніж намагатися зобразити і записати його самому.

Види ігрових Ассет

- звук: фонові музика, звукові ефекти, голоси.
- персонажі: 2D або 3D-моделі, кастомізуємо елементи зовнішності і костюма.
- навколишнє середовище: геометричні моделі, зображення (спрайт), спецефекти (вода, вогонь, дим), що руйнуються об'єкти, будь реквізит.

Приклад ігрового Ассет - навколишнє середовище, кімната геймера.

- всі види транспорту.
- всі види вогнепальної, холодної і будь-якого іншого зброї.
- анімації.
- сюжетні і текстові Ассет: сценарій, діалоги, репліки.
- навколишнє середовище, в тому числі рівні і місії, ландшафт, карти.

- службові Ассет: фрагменти коду, модулі, повні проекти для імпорту в ігровий движок.
- основні вбудовані Ассет
- анімаційні кліпи - файли з даними про те, як змінюються координати об'єктів.
- аудиомикшер - файли з настройками гучності звуку і звукових ефектів.
- примітиви - моделі для створення примітивних геометричних фігур, наприклад, куба, сфери.
- guisskin - файли зі стилями призначеного для користувача інтерфейсу.
- префаб - файли з ігровими шаблонами, наприклад, шаблонами зброї.
- файли з настройками сцени і рендеринга об'єктів на ній.
- файли з настройками спрайтів - 2D-картинок, які формують 3D-моделі.

Одне з головних пунктів для початку реалізації створення гри - дизайн , якщо вирішили не створювати дизайн гри з нуля. Наприклад гравця, задній фон, будівлі і т.д. То в інтернеті є безліч сайтів з готовими ассетами.

Здебільшого ассети не безкоштовний. Так як будь-яка праця і справді повинен бути оцінений.

Зазвичай безкоштовні ассети дуже прості і не сильно заморочені по виду. Але якщо добре пошукати, то можна знайти щось зі своїми потребами.

В свій роботі брала ассети с двух сайтів:

Перший сайт: opengameart.org

Вичерпний ресурс для розробників ігор з відкритим вихідним кодом, Open Game Art можна назвати найбільшим зібранням ігрових Ассет з вільною ліцензією. Всі матеріали - від спрайтів до іконок - можна завантажити за ліцензіями GNU або Creative Commons. Це відмінний сайт для новачків, але багато Ассет відрізняються по візуальному стилю, тому їх доведеться ретельно відбирати (рисунок 3.1.)



Рисунок 3.1 – Перший сайт

Другий сайт: **craftpix.net**

Можна купити різноманітні мультяшні Ассет для 2D платформер, включаючи анімованих персонажів, монстрів, зброя, візуальні ефекти і набори елементів інтерфейсу. (рисунок 3.2.)

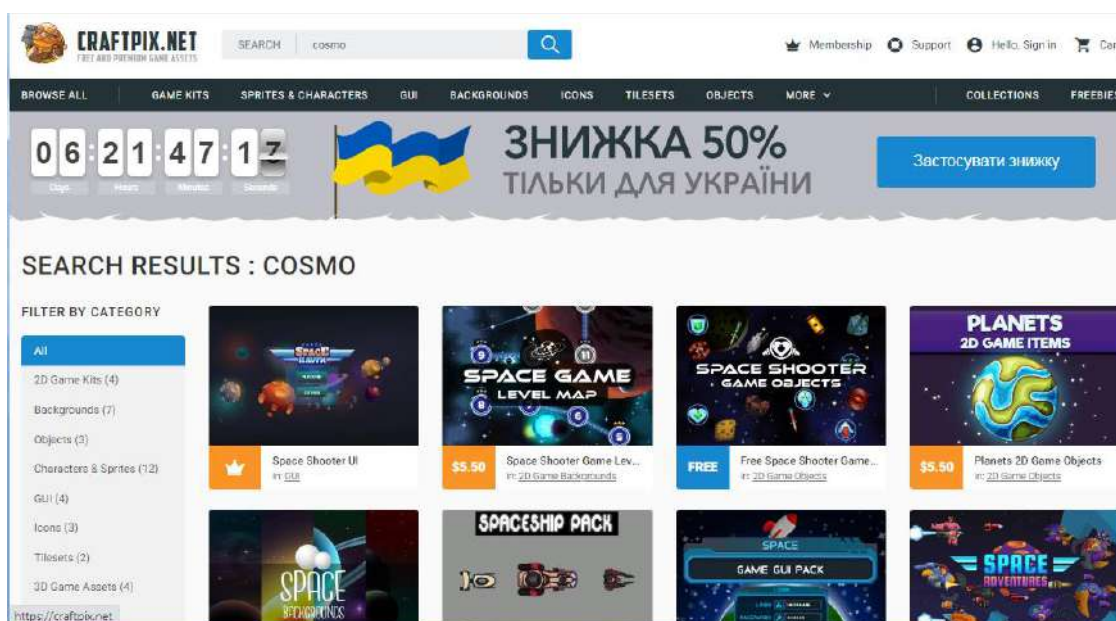


Рисунок 3.2 – Другий сайт

Для мене і мого рівня знання в написання скрипта і нового вивчення движка, потрібно було знайти казуальні спрайти.

Для початку Спрайт - це прості 2D-об'єкти, на яких є графічні зображення (так звані текстури). Всякий раз, коли Godot створює новий спрайт, він використовує текстуру. Потім ця текстура застосовується до свіжого Player, і до нього прикріплюється компонент collistionShape2D.(рисунок 3.3-3.4-3.5)



Рисунок 3.3 – Скачування спрайтів

..		
armor.png	388	388
button_hover.png	3 150	229
button_normal.png	3 150	229
button_pressed.png	3 150	226
enemy_clever.png	622	622
enemy_kamikaze.png	383	383
explosion.png	794	794
flare.png	136	133
icon.png	2 761	452
laser_enemy.png	140	134
laser_ship.png	104	101
powerup_armor.png	192	190
powerup_laser.png	166	165
rectangle.png	178	152

Рисунок 3.4 – Распокуваання

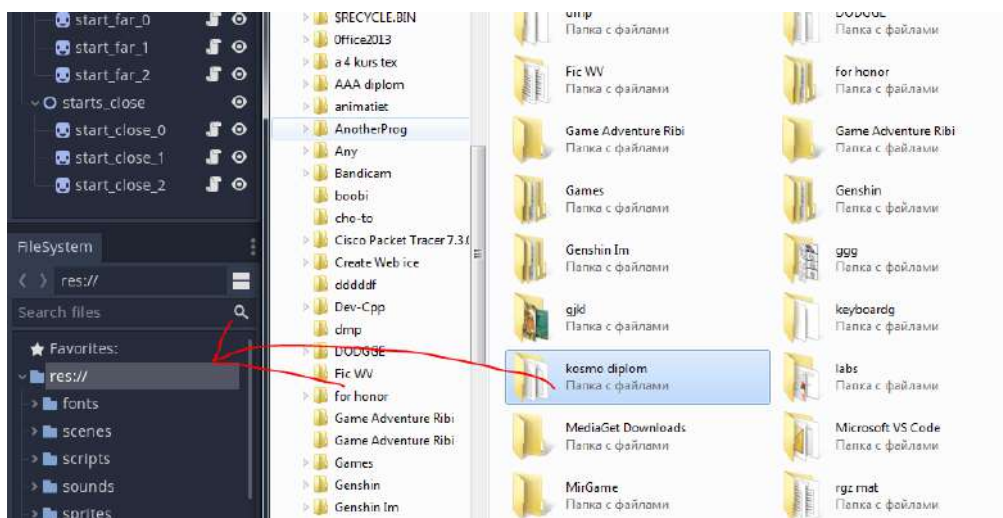


Рисунок 3.5 – Завантаження спрайтів

Файлова система управляє тим, як зберігаються ресурси і як до них здійснюється доступ. Добре продумана файлова система також дозволяє декільком розробникам редагувати одні і ті ж вихідні файли і ресурси під час спільної роботи. Godot зберігає всю бібліотеку ресурсів у вигляді файлів у своїй файлової системи.

Файлова система зберігає ресурси на диску. Все, що завгодно, від скриптів до сцен або зображень PNG, є ресурсом для движка. Якщо ресурс містить властивості, які посилаються на інші ресурси на диску, шляхи до них також включаються. Якщо ресурс має вбудовані суб-ресурси, то він зберігається в одному файлі разом з усіма включеними в комплект суб-ресурсами. Наприклад, ресурс шрифтів часто поставляється в комплекті з текстурами шрифтів.

Файлова система Godot уникає використання файлів метаданих. Існуючі менеджери активів і VCS найкраще, чим все, що ми можемо реалізувати, тому Godot намагається всіма силами працювати разом з SVN, Git, Mercurial, Perfce і т.д.

Недоліки

У цьому простому дизайні файлової системи є деякі недоліки. Перша проблема полягає в тому, що переміщення Ассет(перейменування або переміщення їх з однієї колії на іншу всередині проекту) порушить існуючі

посилання на ці Ассет. Ці посилання повинні бути перевизначені, щоб вказати на нове місце розташування Ассет.

Щоб уникнути цього, виконайте всі свої дії з переміщення, видалення та перейменування з Godot, в панелі файлової системи. Ніколи не рухайте активи поза Godot, інакше залежно доведеться виправляти вручну. (рисунок 3.6)

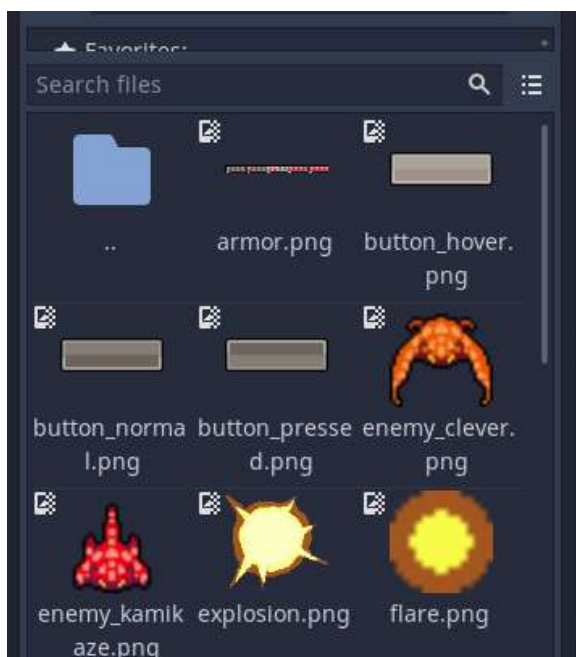


Рисунок 3.6 – Вигляд спрайтів в ФС

3.2 Побудова дочірніх вузлів та розміщення спрайтів

Вузли (ноди)

Але давайте почнемо з основ. Ноди є фундаментальними будівельними блоками для створення гри. Як згадувалося раніше, нода може виконувати безліч спеціалізованих функцій. Однак будь-яка використовувана нода завжди має такі атрибути:

- у неї є ім'я.
- він має редаговані властивості.

- він може отримувати зворотний виклик для обробки кожного кадру.
- він може бути розширений (щоб мати більше функцій).
- він може бути доданий до іншого вузла (ноду) в якості дочірнього.

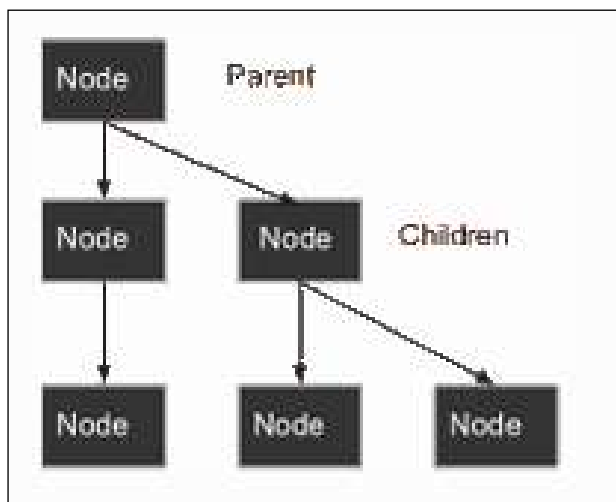


Рисунок 3.1 – Древо

Ще одне важливе зауваження. У вузлів можуть бути інші вузли в якості дочірніх. При розташуванні таким чином вузли утворюють дерево.

Здатність розташовувати вузли таким чином створює потужний інструмент для організації проєктів. Оскільки різні вузли мають різні функції, об'єднання їх дозволяє створювати більш складні функції.

Не турбуйтеся, якщо не розібралися в цьому. Ми продовжимо вивчення цього в наступних декількох розділах. Найважливіше, що потрібно запам'ятати на даний момент - це те, що вузли існують і можуть бути організовані таким ось чином.

Сцени

Тепер, коли концепція вузлів визначена, наступним логічним кроком є пояснення того, що таке Сцена.

Сцена складається з групи вузлів, організованих ієрархічно (у вигляді дерева). Крім того, сцена:

- завжди має тільки один кореневий нод.
- може бути збережена на диск і завантажена назад.

- може бути інстанційовати (т. е. використана кілька разів через створення її примірників, докладніше про це пізніше).

Запуск гри означає запуск сцени. Проект може містити кілька сцен, але для початку гри одна з них повинна бути обрана в якості головної сцени.

Тепер, щоб додати вузол мітки в цю сцену, ви можете натиснути на кнопку "Other Node" (Інший вузол) або кнопку "Add Node" (Додати вузол) у верхній частині. У непустих сценах використовується кнопка "Add Node" (Додати вузол) для створення кожного дочірнього вузла.(рисунок 3.2)

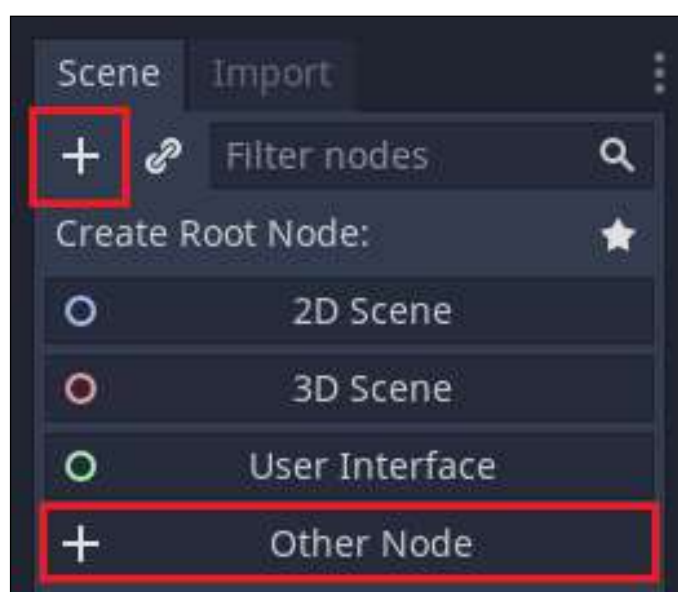


Рисунок 3.2 – Початок роботи

Відкриється діалогове вікно Створення Вузла, що показує довгий список вузлів, які можуть бути створені(рисунок 3.3):

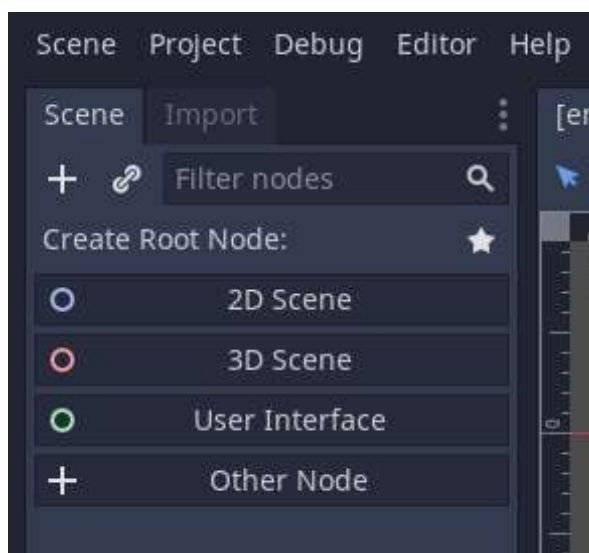


Рисунок 3.3 – Вікно з усіма варіантами вузлів

Надалі я створю потрібні мені вузли для моєї казуальної гри.

Мені потрібні були кілька вузлів.

Area2D - 2D область, яка виявляє **CollisionObject2D** вузли, що перекриваються, входять або виходять. Також може змінювати або замінювати місцеві фізичні параметри (сила тяжіння, демпфування).

AnimatedSprite - Користуючись справою, яка може використовувати кілька текстур для анімації. Анімації створюються за допомогою ресурсу **SpriteFrames**, який можна налаштувати в редакторі через панель **SpriteFrames**.

CollisionShape2D - Вузол, що представляє дані форми зіткнення у 2D просторі.

Редактор для створення та редагування форм зіткнень у двовимірному просторі. Ви можете використовувати цей вузол для представлення всіляких форм зіткнень, наприклад, додати його до **Area2D**, щоб надати йому форму виявлення, або додати до **PhysicsBody2D**, щоб створити твердий об'єкт.

Це допоміжний редактор лише для редагування фігур. (рисунок 3.4-3.5)

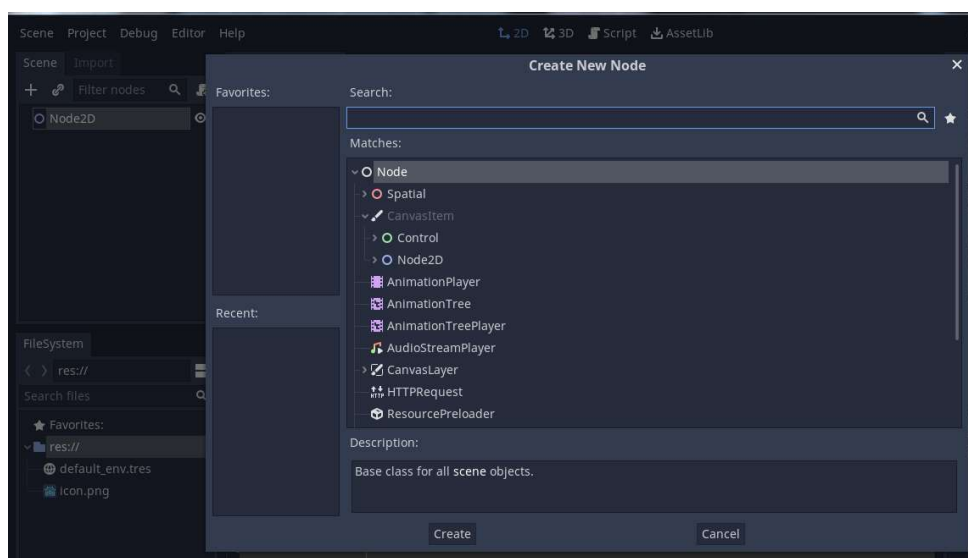


Рисунок 3.4 – Вузли для функціонування ігрока

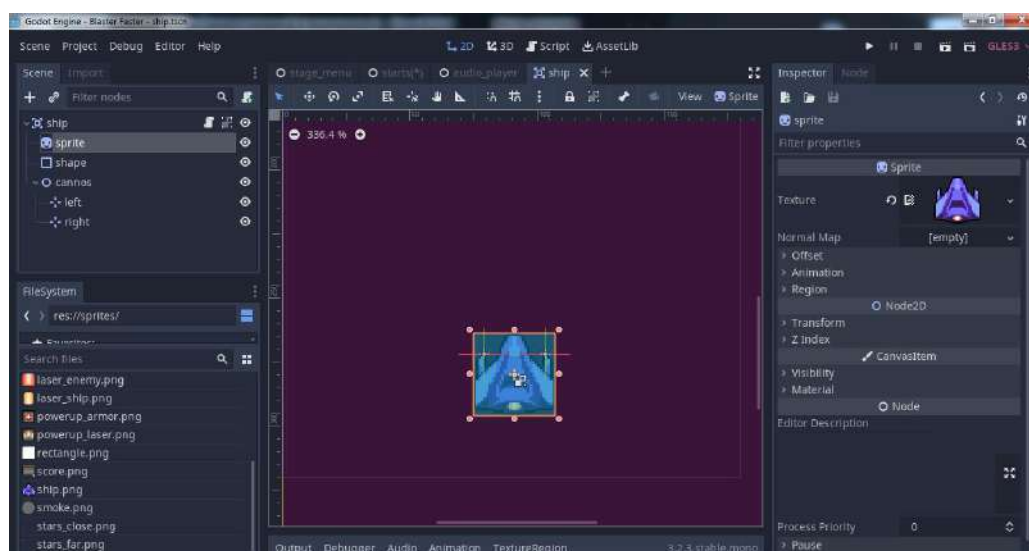


Рисунок 3.5 – Розміщення спрайтів

Практично для Моб, котрій виступає ворогом для гравця під час гри. Мі майже створюємо ті ж узляу тільки з поправкою. що у мобу є `VisibilityNotifier2D` визначає, коли він видно на екрані. Він також повідомляє, коли його обмежує прямокутник входить або виходить з екрану або області перегляду.

Якщо ви хочете, щоб вузли автоматично відключалися при виході з екрану, використовуйте замість цього `VisibilityEnabler2D`.

Примітка. З міркувань продуктивності `VisibilityNotifier2D` використовує приблизну евристику з точністю, яка визначається

ProjectSettings.world/2d/cell_size. Якщо вам потрібна точна перевірка видимості, використовуйте інший метод, наприклад додавання вузла Area2D як дочірнє вузла вузла Camera2D.(рисунок 3.6)

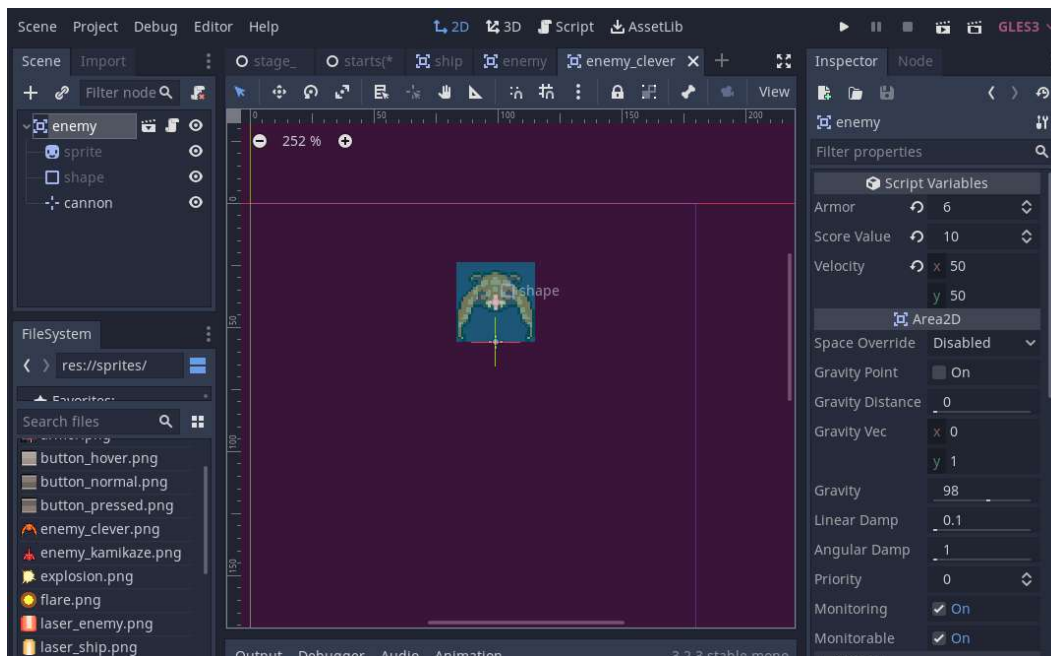


Рисунок 3.6 – Вузли для ворога гріка

Node --це будівельні блоки Годо. Їх можна призначити дочірнім елементом іншого вузла, що призведе до розташування дерева. Даний вузол може містити будь-яку кількість вузлів як дочірніх з вимогою, щоб усі брати і сестри (прямі дочірні вузли) мали мати унікальні імена.

Дерево вузлів називається сценою. Сцени можна зберігати на диску, а потім інстансувати до інших сцен. Це забезпечує дуже високу гнучкість в архітектурі та моделі даних проєктів Годо.

Sprite - Вузол, що відображає 2D-текстуру. Відображена текстура може бути областю від більшої текстури атласу або рамкою з анімації листів спрайтів.

Path2B - Може мати дочірні вузли PathFollow2D, що рухаються вздовж Curve2D. Докладніше про використання див.

У PathFollow2D -Цей вузол бере свій батьківський Path2D і повертає координати точки всередині нього, задану відстанню від першої вершини.

Це корисно для того, щоб інші вузли йшли шляхом, без кодування шаблону руху. Для цього вузли повинні бути дочірніми цього вузла. Потім нащадки-вузли рухатимуться відповідно при встановленні зміщення в цьому вузлі.

Timer - Зворотний відлік здійснює заданий інтервал і видає сигнал при досягненні 0. Може бути встановлений для повторення або режиму «один постріл»

RigidBody2D - Цей вузол реалізує імітаційну 2D-фізику. Ви не керуєте RigidBody2D безпосередньо. Натомість ви застосовуєте до нього сили (сила тяжіння, імпульси тощо), і фізичне моделювання обчислює результуючий рух на основі його маси, тертя та інших фізичних властивостей.

RigidBody2D має 4 режими поведінки: жорсткий, статичний, символічний та кінематичний.

Position2D - Загальний підказка про позицію 2D для редагування. Це як звичайний Node2D, але він відображається як хрест у 2D-редакторі завжди. Ви можете встановити перехресний візуальний розмір, використовуючи штучовину у 2D-редакторі, поки вибрано вузол.

AudioStreamPlayer - Відтворює аудіо без позиції.(рисунок 3.7.)

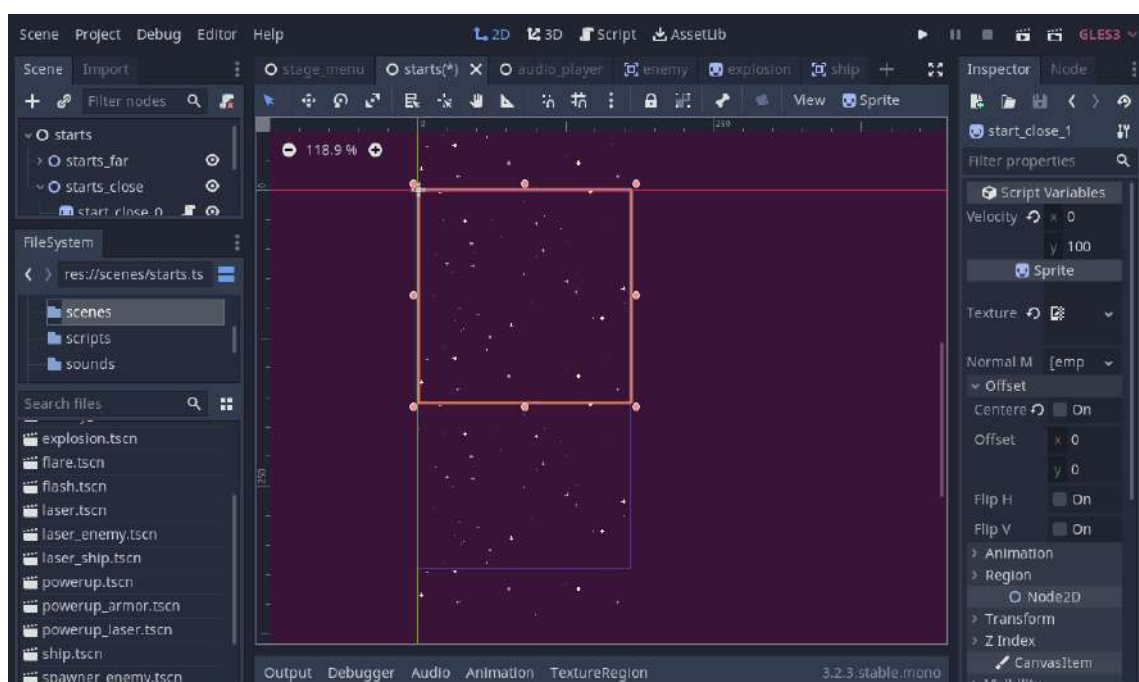


Рисунок 3.7 – Основна сцена з доп всіма вузлів

CanvasLayer - Шар для малювання полотна. У цьому шарі будуть намальовані вузли CanvasItem, які є прямими або непрямими дочірніми елементами CanvasLayer. Шар - це числовий індекс, який визначає порядок розіграшу. Двовимірна сцена за замовчуванням відображається з індексом 0, тому внизу буде намальовано CanvasLayer з індексом -1, а вище - індекс 1. Це дуже корисно для HUD (у шарі 1+ або вище) або фонів (у шарі -1 або нижче).

Label - Етикетка відображає на екрані звичайний текст. Це дає вам змогу контролювати горизонтальне та вертикальне вирівнювання та може обернути текст усередині обмежувального прямокутника вузла. Він не підтримує жирний шрифт, курсив чи інше форматування. Для цього використовуйте RichTextLabel.

Button - Кнопка - це стандартна тематична кнопка. Він може містити текст та піктограму та відображатиме їх відповідно до поточної теми.

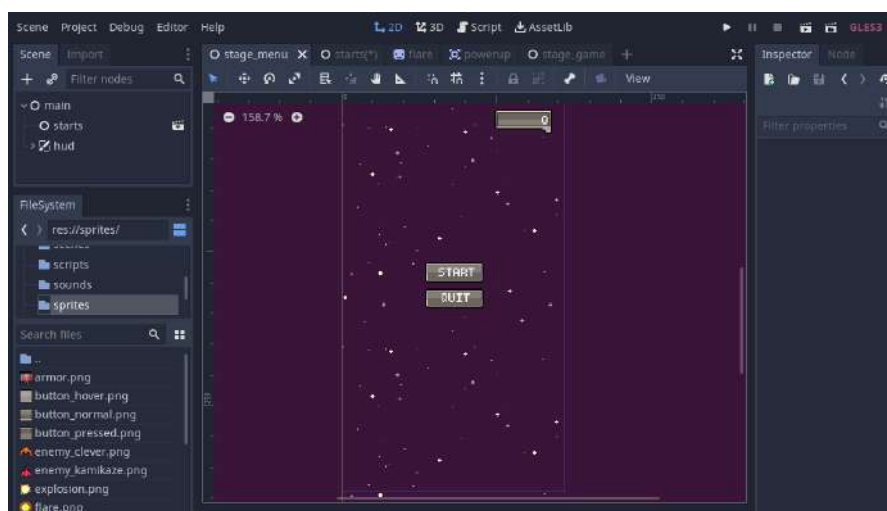


Рисунок 3.8 – Сцена с кнопками

Аудіосистема Godot дозволяє створювати будь-яку кількість аудіо шин та додавати до кожної шини будь-яку кількість ефектів. Тільки потужність пристрою на якому запущена ваша гра, може обмежити кількість шин і ефектів, що використовується до того, як продуктивність почне падати.

Шкала децибел створено спеціально для звукового дизайну. Для цього він використовує шкалу децибел.

Можна знайти на нижній панелі редактора Godot:

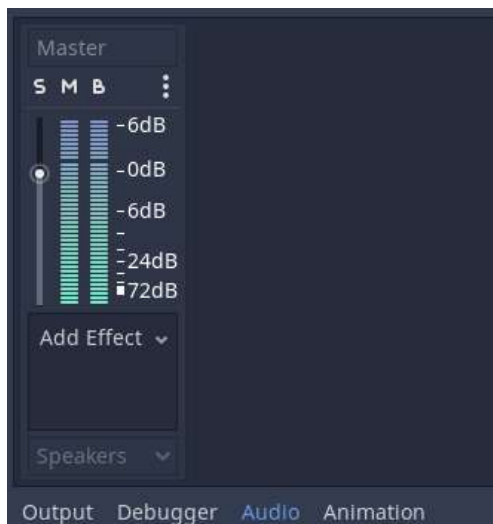


Рисунок 3.9 – Сцена с ауді доріжкою

Відтворення аудіо через шину. Створити вузол `AudioStreamPlayer`, завантажити аудіодані та вибрати цільову шину для відтворення:

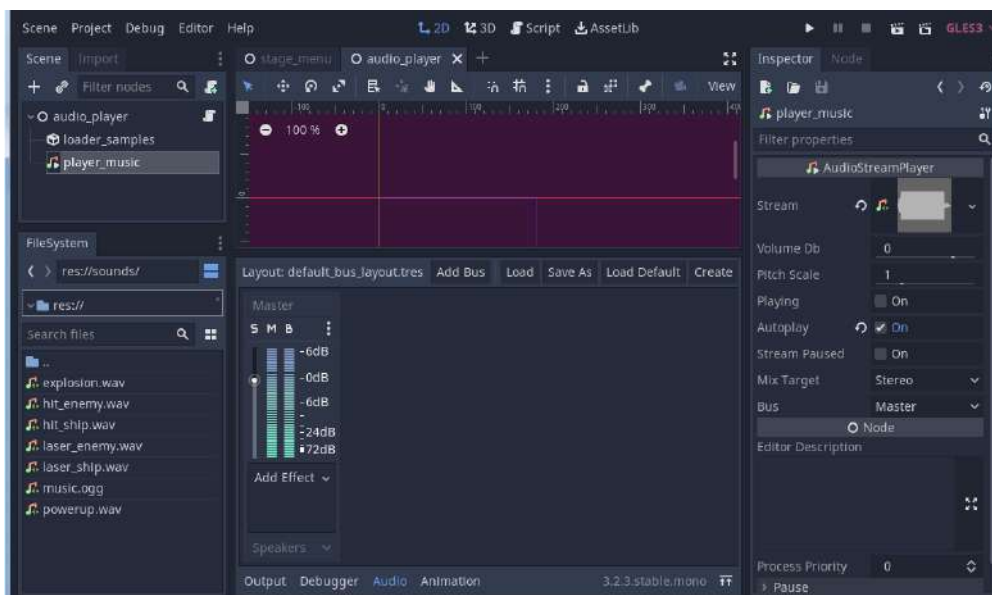


Рисунок 3.9 – Сцена с ауді доріжкою в крупному обзорі

Спрайтовий лист за допомогою `AnimationPlayer`. Інший спосіб можна створити анімації з використовуючи спрайт-лист, - це використовувати стандартний вузол `Sprite` для відображення текстури, а потім анімувати заміну текстури на текстуру за допомогою `AnimationPlayer`. (рисунок 3.10)

`AnimationPlayer` і натиснути кнопку "Анімація", буде "Новий".. Довжина анімації на 0.6 та кнопку "Зациклювання анімації", щоб анімація повторювалася. (рисунок 3.11)

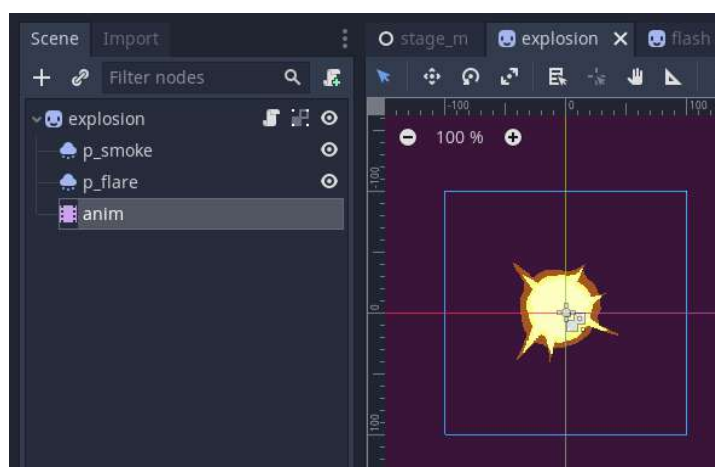


Рисунок 3.10 – Сцена вузлів анімації

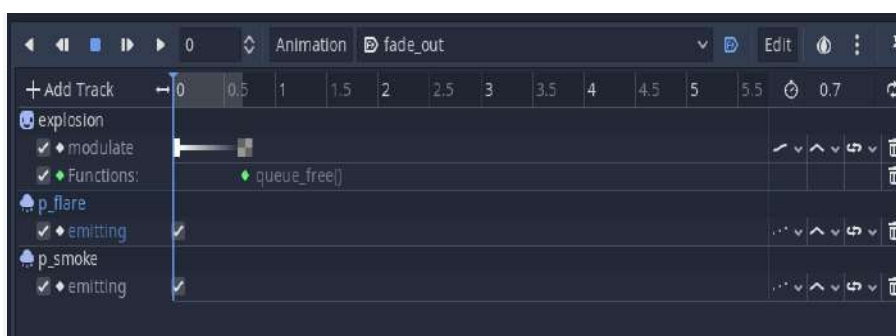


Рисунок 3.11 – Сцена с доріжкою анімації

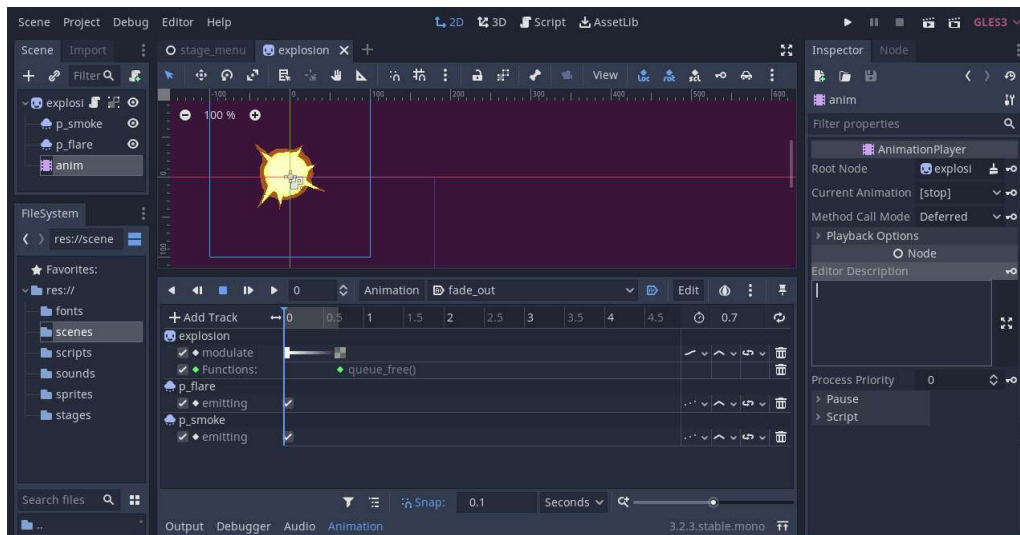


Рисунок 3.12 – Сцена анімації в крупному огляді

3.3 Написання скрипта

Написання скрипту для Ship. (рисунок 3.13.)

```

1  # script: ship
2
3  extends Area2D
4
5  var armor = 4 setget set_armor
6  var is_double_shooting = false setget set_double_shooting
7
8  const scn_laser = preload("res://scenes/laser_ship.tscn")
9  const scn_explosion = preload("res://scenes/explosion.tscn")
10 const scn_flash = preload("res://scenes/flash.tscn")
11
12 signal armor_changed
13
14 func _ready():
15     add_to_group("ship")
16
17     yield(utils.create_timer(0.5), "timeout")
18     shoot()
19     pass
20
21 func _process(delta):
22     # tracking mouse
23     var motion = (utils.mouse_pos.x - position.x) * 0.2
24     translate(Vector2(motion, 0))

```

Рисунок 3.13 - Ship

Написання скрипту для Ship. (рисунок 3.14.)



Рисунок 3.14 – Ship signal

Написання скрипту для Ship. (рисунок 3.15.)

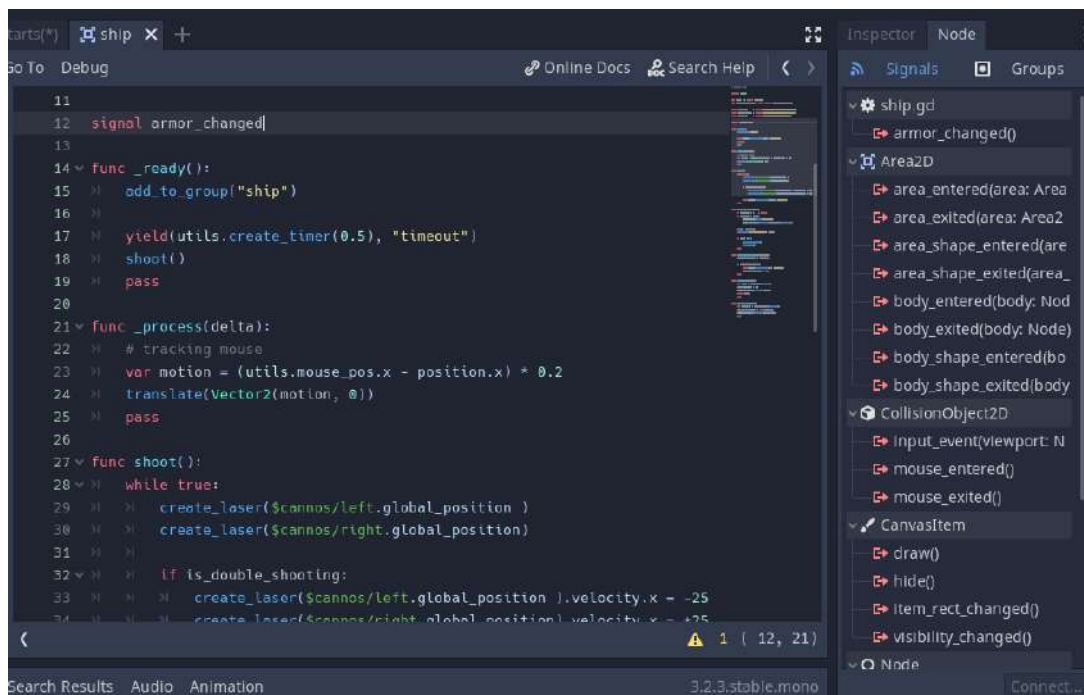


Рисунок 3.16 – Ship

Обробка сигналу

Сигнали "випромінюються", коли відбувається певний вид дій, і вони можуть бути підключені до будь-якої функції будь-якого примірника сценарію. Сигнали використовуються в основному в GUI-вузлах, хоча вони також є і в інших вузлах, і ви навіть можете визначити для користувача сигнали в своїх власних сценаріях.

На цьому етапі ми підключимо сигнал "pressed" ("натиснуто") до користувацької функції. Формування з'єднань - це перша частина, а визначення користувацької функції - друга. Для першої частини Godot надає два способи створення з'єднань: через візуальний інтерфейс, який редактор надає, або через код.

Хоч ми і будемо використовувати метод написання коду для решти цієї серії уроків, давайте розглянемо як працює інтерфейс редактора, для подальшого використання.

Виберіть вузол Button в дереві сцен і потім виберіть вкладку "Вузол" (Node). Далі, переконайтеся, що ви вибрали "Сигнали" (Signals).

Написання скрипту для Main. (рисунок 3.12.)

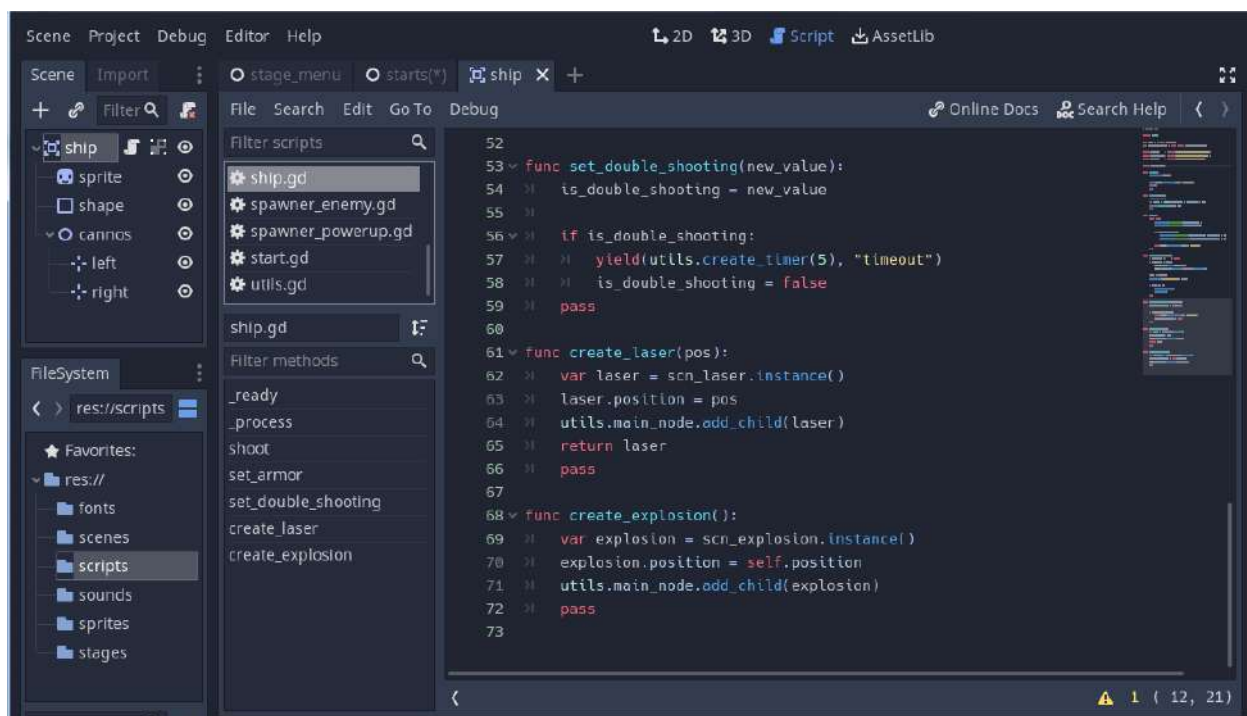


Рисунок 3.17 – Ship сигнал

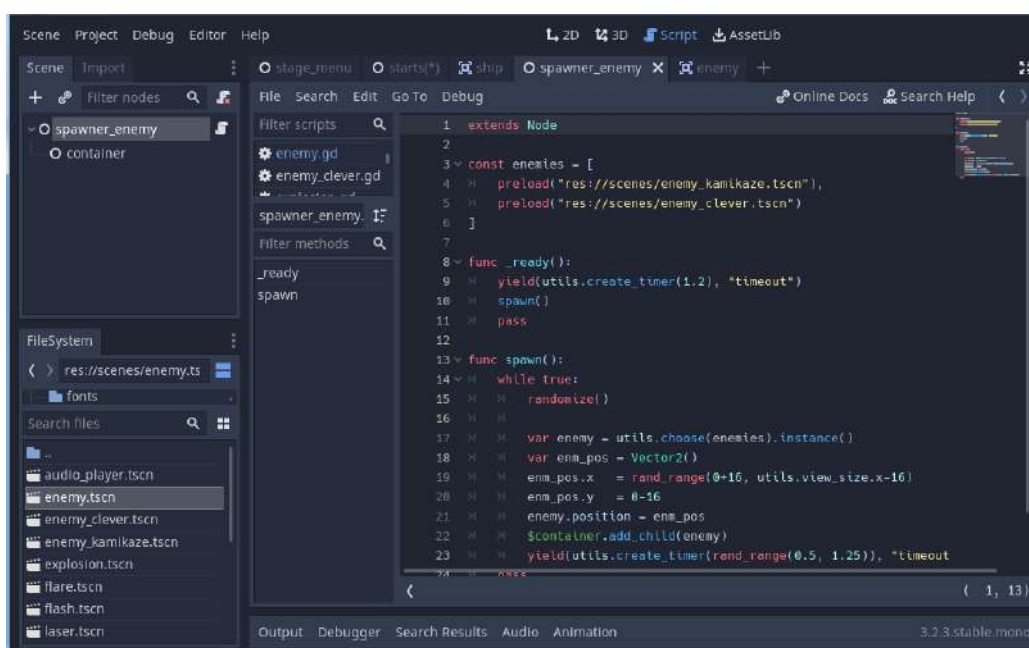


Рисунок 3.18 – Enemy

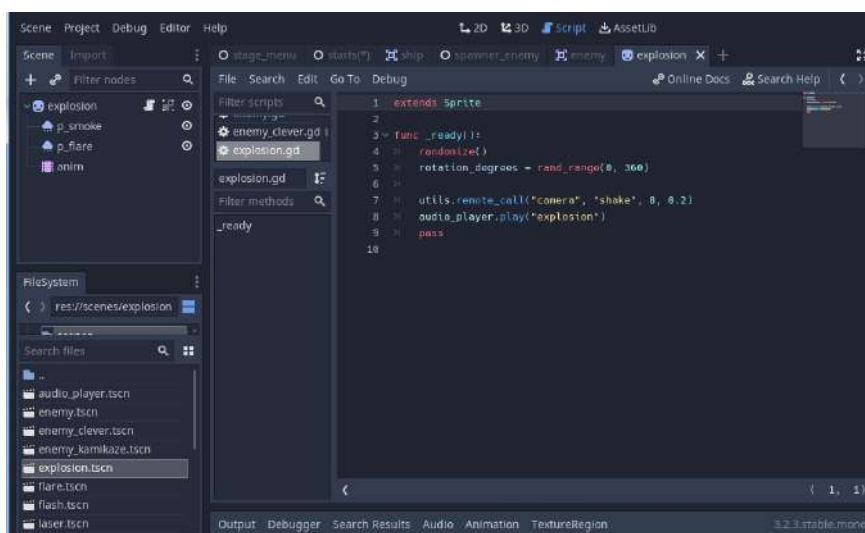


Рисунок 3.19 – Enemy explosion

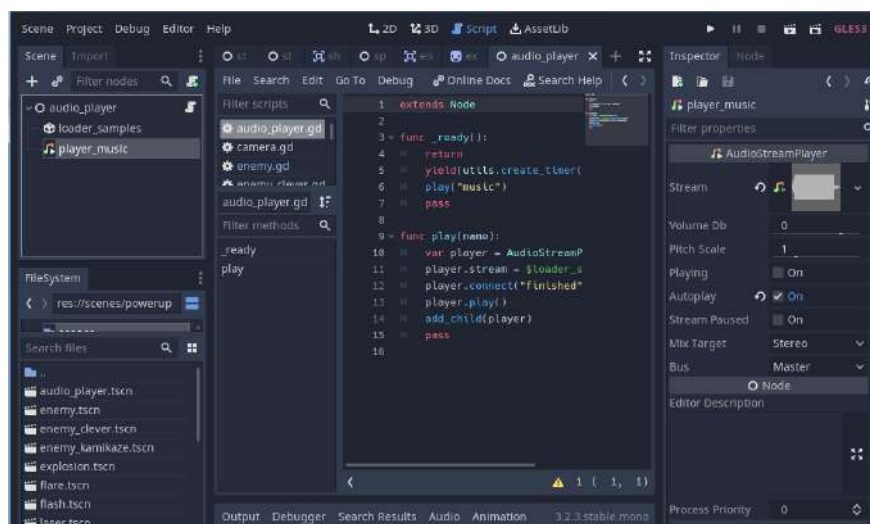


Рисунок 3.20 – Audio.player

3.4 Тестування гри та позбавлення багів

Варто зазначити, що на даному етапі реалізації проекту проводити будь-яке велике тестування сенсу немає, так як в грі реалізована лише мала дециця можливостей, що передбачалося реалізувати. Саме з цієї причини дане тестування, є лише невеликий перевіркою на відповідність основним функціональним вимогам.

Дане тестування показує, що на даному етапі реалізації проекту, гра відповідає більшості функціональних вимог виділених на етапі постановки завдання. Але все ж варто приділити особливу увагу програмного коду і виправити помилку, через яка ситуація програшу гравця відпрацьовується невірно.

Таблиця. 3.1. Тестування гри

Призначення	Дії	Очікуваний результат	Отриманий результат	підсумок
Перевірка на коректне реагування персонажа на що надходять команди про пересуванні	пробігти	Персонаж рухається в призначеному напрямку	Персонаж пересувався в потрібному напрямку, коректне відображення анімації	Пройдено
Перевірка поведінки ворогів	вороги по сьому екрані	Поява ворогів заданий ран домом при скрипті	Вороги пересувалися по поверхні при появі	пройдено
Перевірка натискання кнопок Старт	Курсор на кнопку Старт	Кнопка запускає гру	Кнопка запускає гру	пройдено
Перевірка мелодії	Запустивши гру мелодія грає далі	Буде запущена музикальна доріжка	При старті гри мелодія програється	пройдено

Продовження таблиці. 3.1. Тестування гри

Призначення	Дії	Очікуваний результат	Отриманий результат	підсумок
Перевірка на коректну реалізацію програшу гравця	Моб повинен завдати урон ігроку	Після зіткнення моба з грацем, повина гра запинитися	При зіткненні моба, гра завершує процес	пройдено
Перевірка звуку при проігрошу	Зікнувшись с мобом повина відігровати мелодія програшу	Повина прозвучати мелодія	Мелодія було проігрована	пройдена
Перевірка на коректну реалізацію програшу гравця	Вороги повинен здати удар гравцю	Зіткнення ворог з гравцем, гра запинитися	При зіткнення ворога, завершує процес	

Дане тестування показує, що на даному етапі реалізації проєкту, гра відповідає більшості функціональних вимог виділених на етапі постановки завдання. Але все ж варто приділити особливу увагу програмного коду і виправити помилку, через яку ситуація програшу гравця відпрацьовується невірно.

ВИСНОВКИ

У роботі проведено всебічний аналіз завдання та сучасних освітніх ігрових середовищ. Вивчено різні технології для створення ігор, що дозволило сформувати цілісне уявлення про наявні методи та засоби розробки.

Досліджено різновиди ігрових движків для різних платформ, що дало змогу визначити їх переваги та недоліки. Особливу увагу приділено інструментарію Godot, який показав високий потенціал для створення освітніх ігор.

Детально розглянуто можливості та особливості Godot Engine. Аналіз функціоналу цього движка підтвердив його ефективність та відповідність вимогам для реалізації освітніх ігор.

Godot Engine виявився потужним інструментом, що надає широкий спектр можливостей для розробки інтерактивних та навчальних ігор.

Виконано процес збору асетів для гри та їх розпакування. Це стало важливим етапом для створення якісного контенту гри.

Показано детальне створення гри та написання скриптів на Godot Engine, що дозволило створити працездатний прототип освітньої гри. Це підтвердило можливість використання цього інструменту для досягнення поставленої мети.

Створений працездатний прототип освітньої гри демонструє практичне застосування теоретичних знань та обґрунтування вибору програмного забезпечення.

Робота може бути корисною для подальших досліджень та розробок у сфері освітніх ігор, а також для впровадження подібних проектів в інших навчальних закладах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1 Документція до Godot Engine. URL: <https://docs.godotengine.org/uk/4.x/about/introduction.html>(дата звернення 23.04.2024).
- 2 Застосування об'єктно-орієнтованого підходу у Godot. URL: https://docs.godotengine.org/uk/4.x/tutorials/best_practices/index.html(дата звернення 29.04.2024).
- 3 Редактор Godot для 2D гри. URL: https://docs.godotengine.org/uk/4.x/getting_started/first_2d_game/index.html(дата звернення 05.05.2024).
- 4 Офіційний сайт програми Godot Engine. URL: <https://godotengine.org/> (дата звернення 21.04.2024).
- 5 Перший офіційний сайт спрайтів. URL: <https://craftpix.net/freebies/free-elven-land-game-battle-backgrounds/>(дата звернення 12.05.2024).
- 6 Другий офіційний сайт спрайтів. URL: <https://opengameart.org/content/background-clouds-and-mountains-parallax>(дата звернення 13.05.2024).
- 7 Сайт форум про обговорення створення 2D гри. URL: <https://gamedev.dou.ua/forums/tags/2D/?from=toptags>(дата звернення 17.05.2024).
- 8 Офіційний сайт Godot Engine блог. URL <https://godotengine.org/news> (дата звернення 22.05.2024).
- 9 Вікіпедія Godot Engine. URL: <https://uk.wikipedia.org/wiki/Godot> (дата звернення 21.04.2024).
- 10 Перший допоміжний сайт с поетапним пояснюванням роботи на базі Godot Engine. URL: <https://www.dvt.co.za/news-insights/insights/item/333-creating-a-2d-game-with-godot-engine-v3> (дата звернення 24.05.2024).

11 Другий допоміжний сайт с поетапним пояснюванням роботи на базі Godot Engine. URL: <https://godottutorials.pro/how-to-make-games-godot/> (дата звернення 25.05.2024).

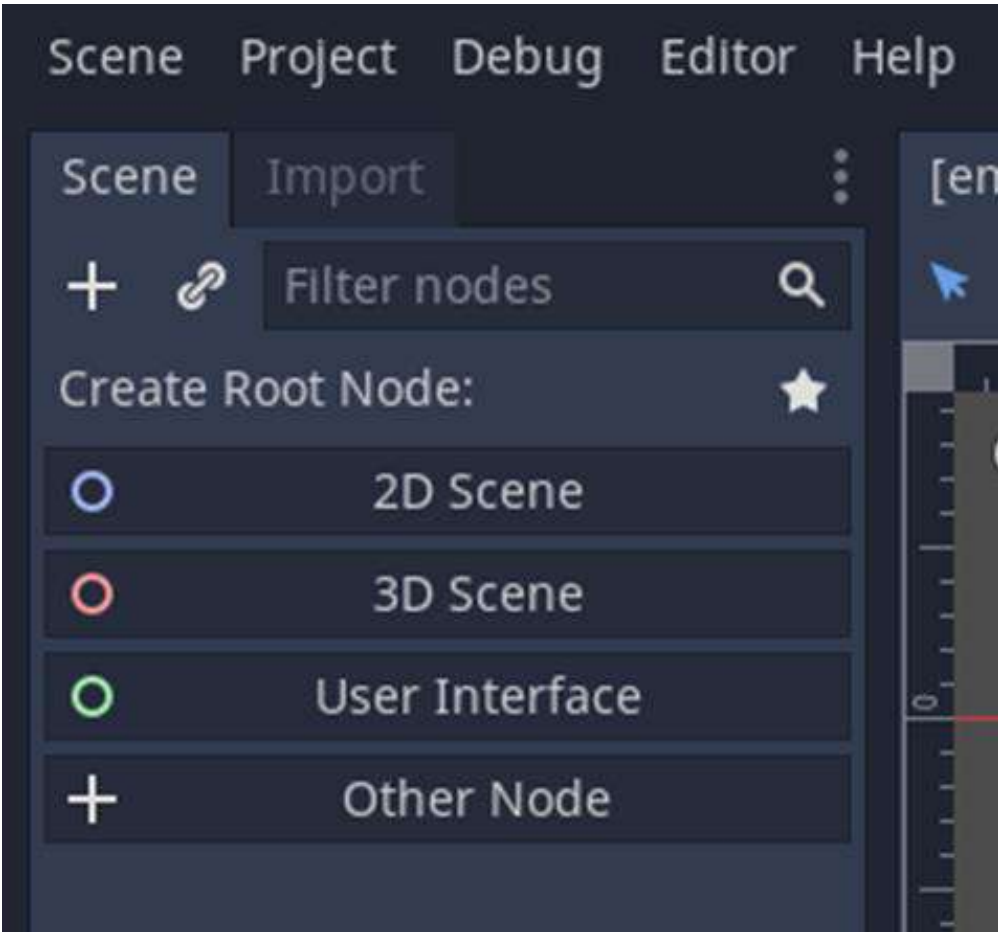
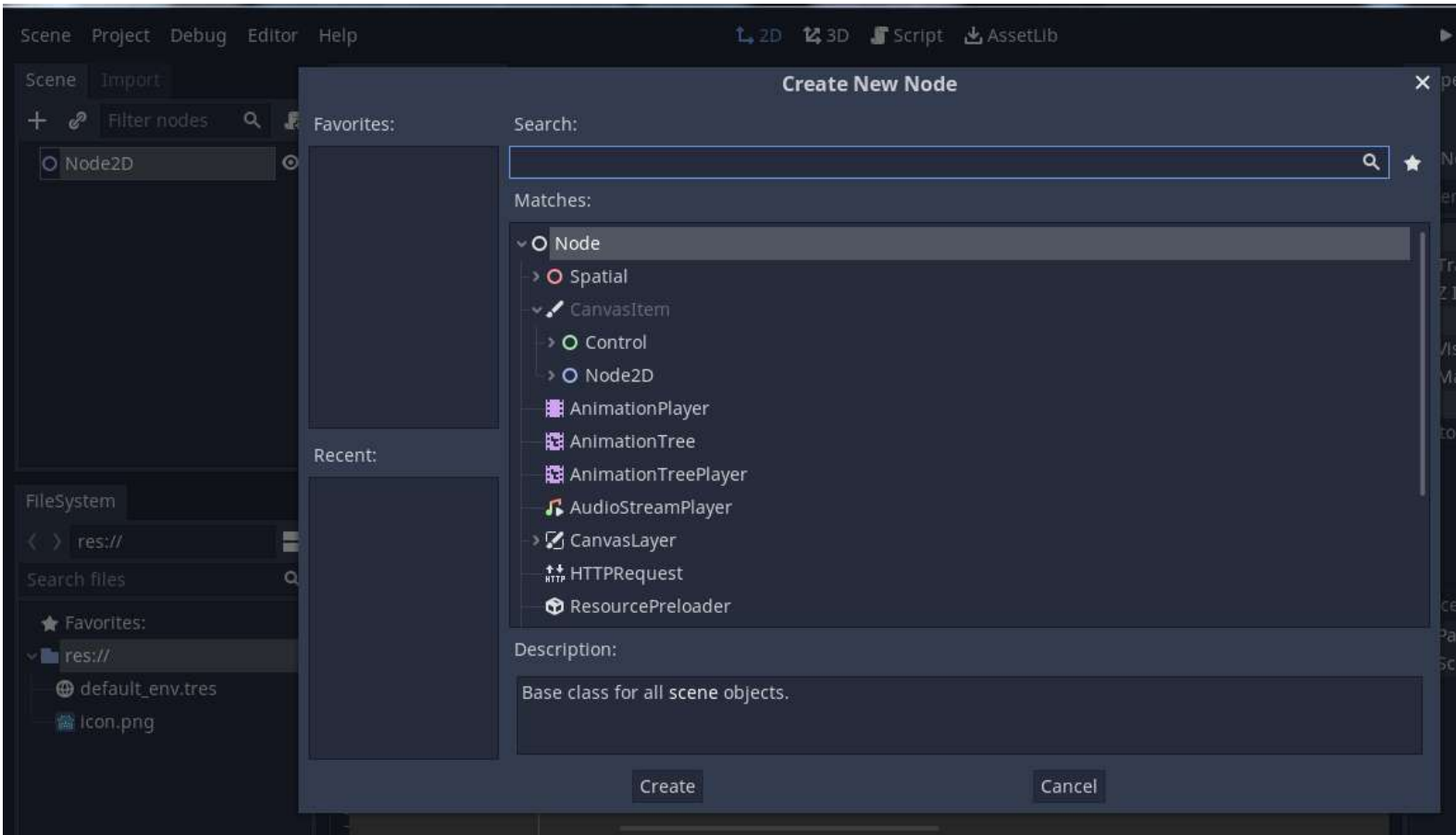
12 Що таке Godot Engine. Посібник із розробки 2D та 3D ігор. URL: <https://gamedevacademy.org/what-is-godot/> (дата звернення 27.05.2024).

13 Репозиторії URL:<https://github.com/search?q=godot&type=repositories> (дата звернення 08.05.2024).

14 Початок роботи з Godot URL: <https://developer.oculus.com/blog/getting-started-with-the-open-source-game-engine-godot/> (дата звернення 27.04.2024).

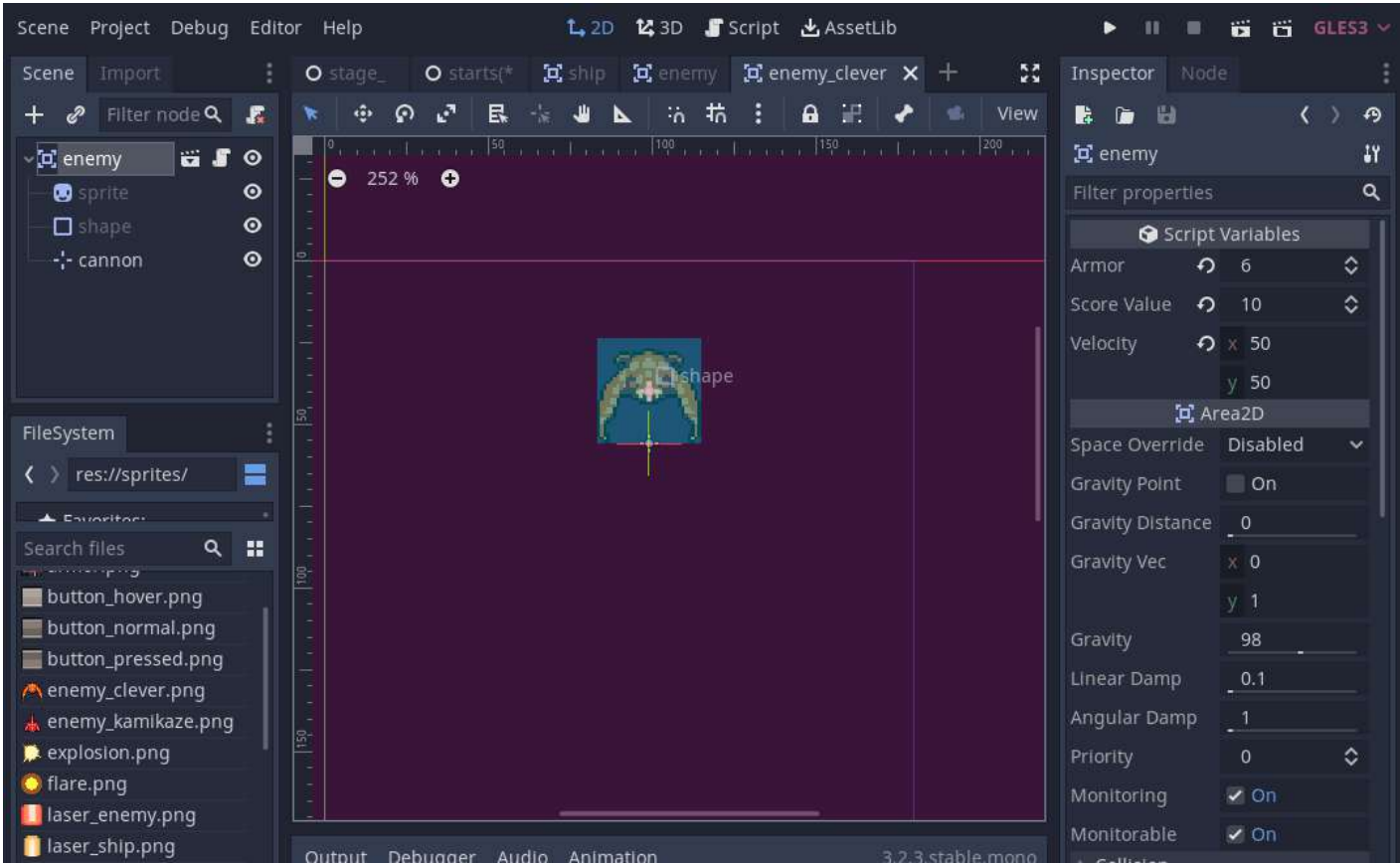
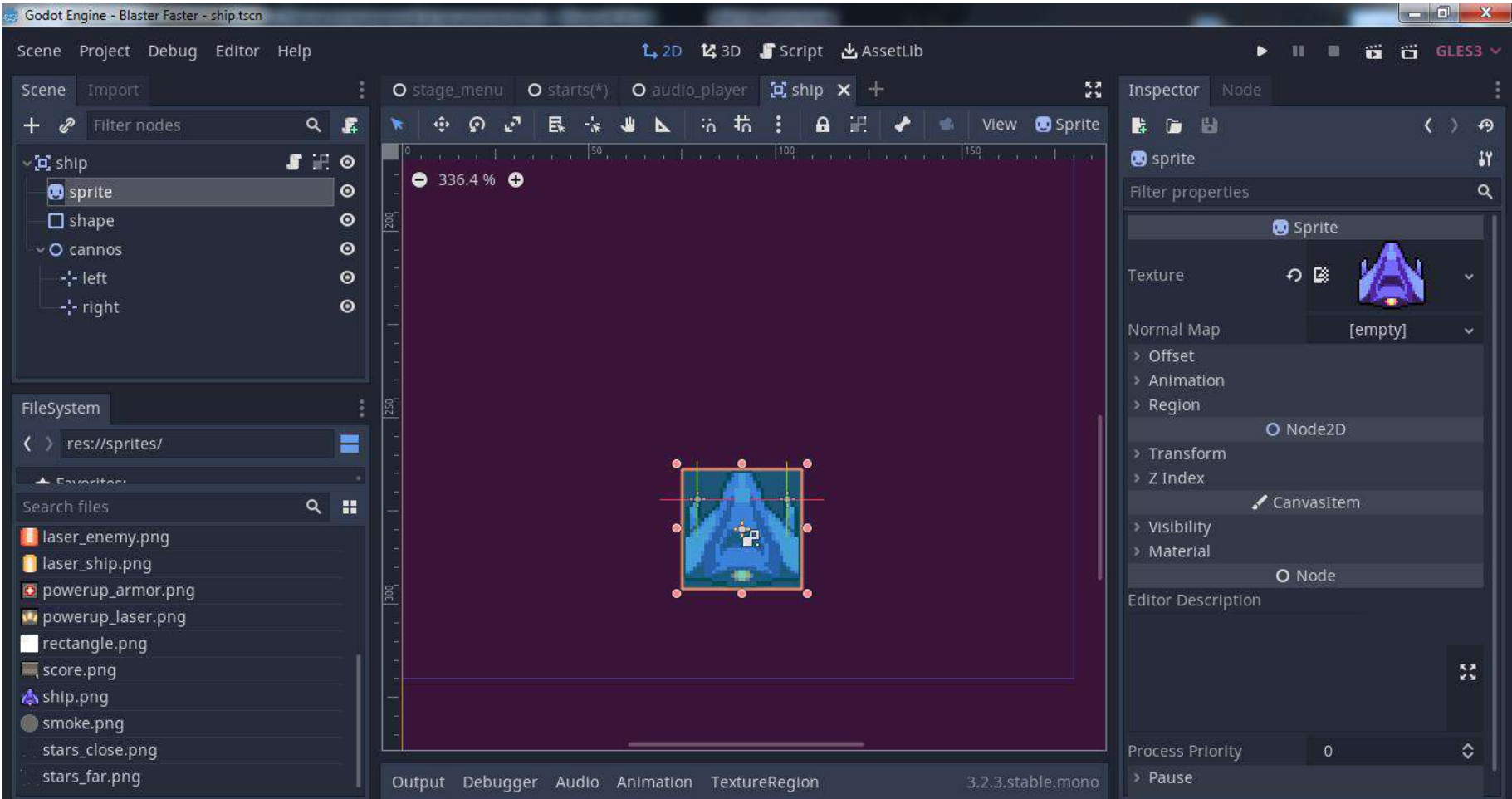
15 Вибір мови програмування для Godot Engine URL:<https://itproger.com/ua/courses/languages/page-3> (дата звернення 24.04.2024).

Вузли для функціювання бази гри



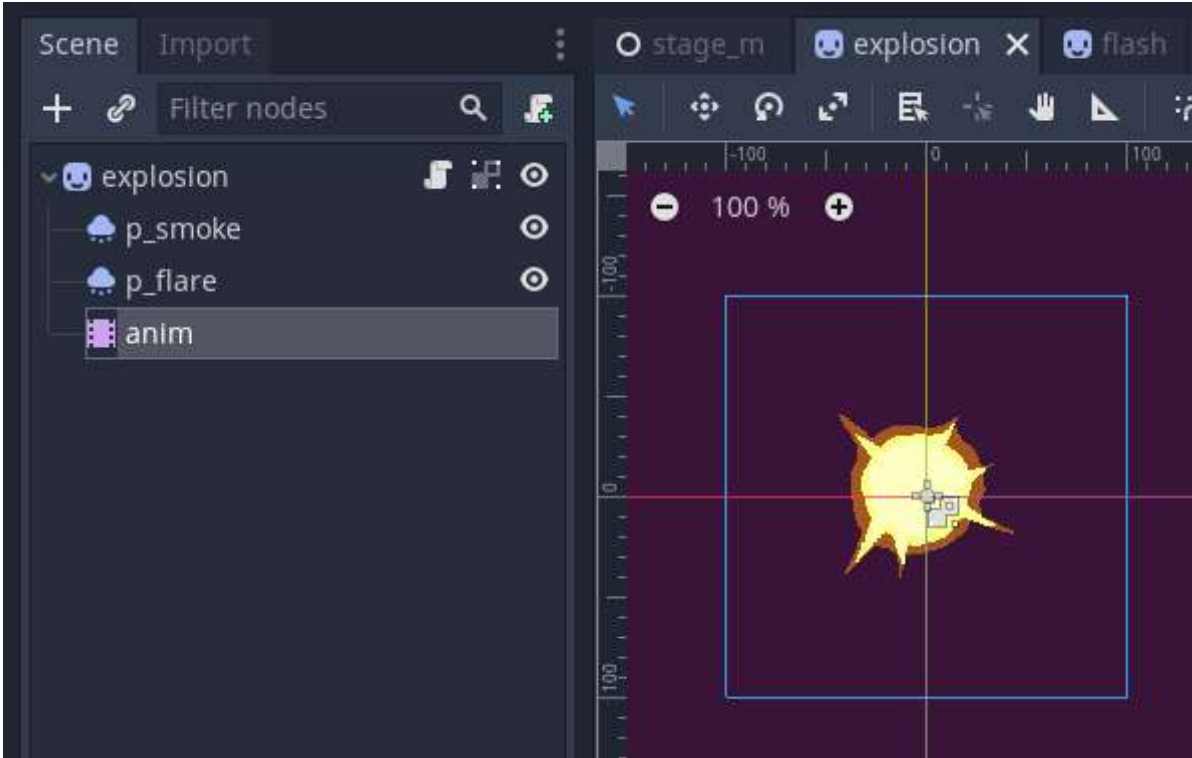
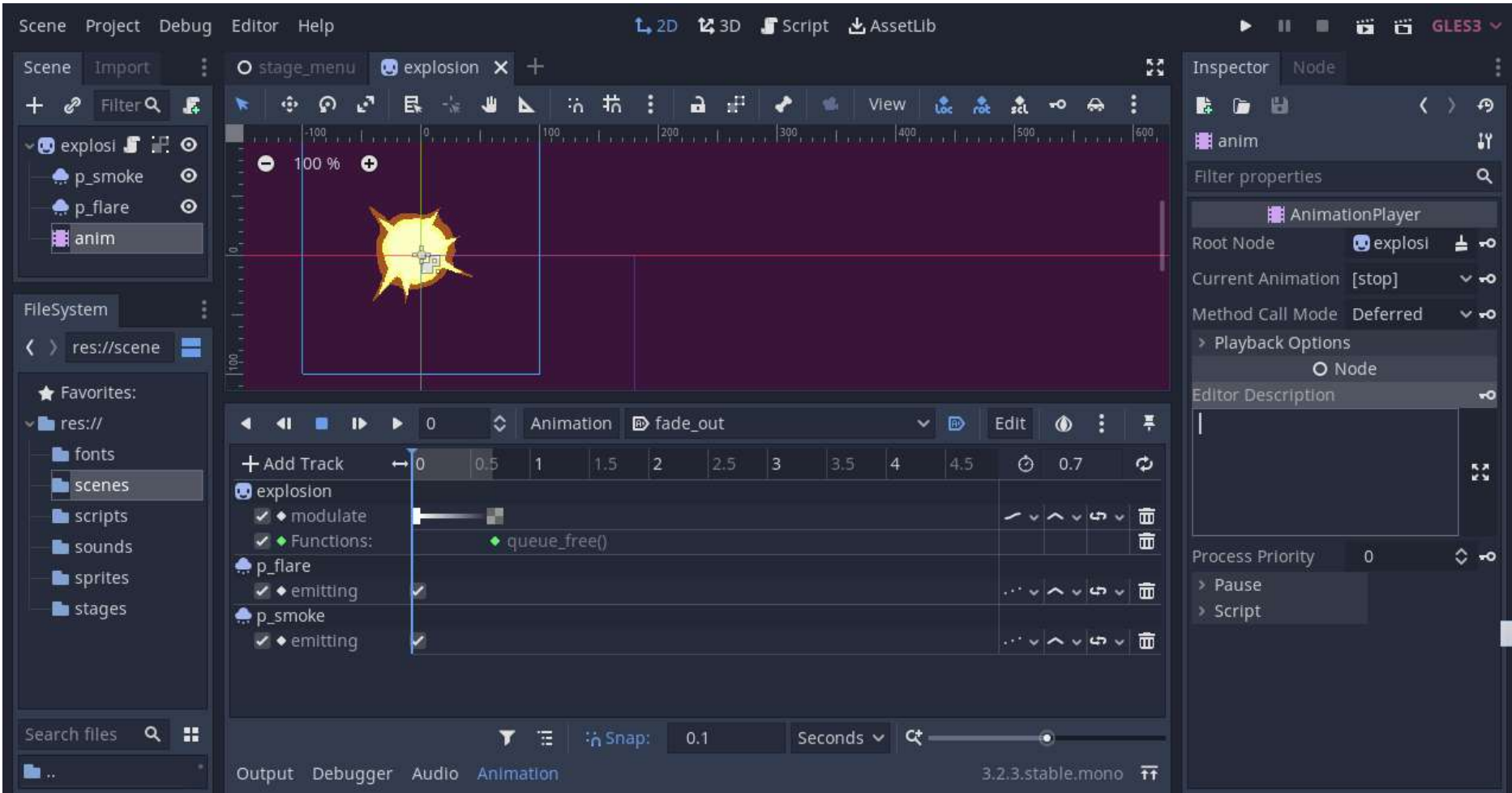
					13.02070849.51131 ПЛ1						
					Розробка гри на базі Godot Engine Вузли для функціювання бази гри	Літ.		Маса	Масштаб		
Зм	Лист	№ докум.	Підп.	Дата							
	Розроб.	Ещєкгулієва Г.П.									
	Перевір.	Тягунова М.Ю									
	Т. контр.					Лист. 1		Листів 1			
						НУ «Запорізька політехніка» КНТ-511сп					
	Н. контр.	Щербак Н.В.									
	Затв.	Кудєрметов Р.К.									

Вузли для функціонування ігрока та ворога ігрока



					13.02070849.51131 ПЛ2			
					Розробка гри на базі Godot Engine Вузли для функціонування ігрока та ворога ігрока	Літ.	Маса	Масштаб
Зм.	Лист	№ докум.	Підп.	Дата				
Розроб.	Ещекгулієва Г.П.		<i>Ещекгулієва</i>					
Перевір.	Тягунова М.Ю.		<i>Тягунова</i>					
Т. контр.								
					НУ «Запорізька політехніка» КНТ-511сп	Лист. 1		Листів 1
Н. контр.	Щербак Н.В.		<i>Щербак</i>					
Затв.	Кудерметов Р.К.		<i>Кудерметов</i>					

Сцена вузлів анімації та сцена с доріжкою анімації



					13.02070849.51131 ПЛЗ			
					Розробка гри на базі Godot Engine Сцена вузлів анімації та сцена с доріжкою анімації	Літ.	Маса	Масштаб
Зм.	Лист	№ докум.	Підп.	Дата				
Розроб.		Ещекгулієва Г.П.	<i>Ещекгулієва</i>					
Перевір.		Тягунова М.Ю.	<i>Тягунова</i>					
Т. контр.						Лист. 1		Листів 1
Н. контр.		Щербак Н.В.	<i>Щербак</i>			НУ «Запорізька політехніка» КНТ-511сп		
Затв.		Кудерметов Р.К.	<i>Кудерметов</i>					

Сигнал ігрока та ворога



					13.02070849.51131 ПЛ4						
					Розробка гри на базі Godot Engine Сигнал ігрока та ворога	Літ.		Маса	Масштаб		
Зм.	Лист	№ докум.	Підп.	Дата							
Розроб.		Ещекгулієва Г.П.									
Перевір.		Тягунова М.Ю.									
Т. контр.						Лист. 1		Листів 1			
						НУ «Запорізька політехніка» КНТ-511сп					
Н. контр.		Щербак Н.В.									
Затв.		Кудерметов Р.К.									