МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ГРИ РАНЕР З ВИКОРИСТАННЯМ РУШІЯ UNITY»**

Виконав: студент 4 курсу, групи 6.1229
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
Я. О. Падалко
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Чопоров С.В.
(підпис)

" ____ " _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Падалко Ярославу Олександровичу

1. Тема роботи Розробка гри Ранер з використанням рушія Unity

керівник роботи Матвіїшина Надія Вікторівна, к.т.н., доцент
(прізвище, ім'я та по–батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

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

1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка гри.
4. Налаштування та тестування гри.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 10.01.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етпів кваліфікаційної роботи	Строк виконання етапів роботи	Примітки
1.	Розробка плану роботи	10.01.2023	
2.	Збір вихідних даних	28.01.2023	
3.	Обробка методичних та	05.02.2023	
	теоретичних джерел		
4.	Розробка першого та другого розділів	10.02.2023	
5.	Розробка третього розділу	20.03.2023	
6.	Оформлення та нормокнтроль	20.05.2023	
	кваліфікаційної роботи бакалавра		
7.	Захист кваліфікаційної роботи	24.06.2023	

Студент _____ Я.О. Падалко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.В. Матвіїшина
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка гри Ранер з використанням рушія Unity»: 53 с., 34 рис., 10 джерел.

ГРА, КРОСПЛАТФОРМНЕ СЕРЕДОВИЩЕ, РАНЕР, СЦЕНАРІЙ ГРИ, UNITY, C#

Об’єкт дослідження – процес розробки гри, включаючи керування проєктом, створення дизайну ігрового світу, звукового оформлення, взаємодії з гравцями.

Мета роботи: створення гри Ранер у кросплатформному середовищі розробки програмного забезпечення Unity.

Метод дослідження – аналіз, описовий, порівняльний.

У кваліфікаційній роботі розроблено сценарій гри в стилі Ранер та реалізовано гру у кросплатформному середовищі Unity: створено та налаштовано ігровий персонаж, перешкоди, бонуси та інші ігрові елементи; розроблено ігрову механіку та ігрові рівні. Для керування персонажем, перешкодами, бонусами тощо, розроблено скрипти із застосування мови програмування C#. Для створення атмосфери гри налаштовано роботу з графікою та аудіоефектами.

SUMMARY

Bachelor's qualification theses "Development of the Runner game using the Unity engine": 53 pages, 34 pictures, 7 sources.

GAME, CROSS-PLATFORM ENVIRONMENT, RUNNER, GAME SCENARIO, UNITY, C#

Object of the study – exploring the game development process, including project management, game world design, sound design, and player interaction.

Aim of the study: Create a game in the Unity development environment for all existing game platforms.

Method of research – analysis, descriptive, comparative.

In the qualifying work, a game script in the style of Runner was developed and the game was implemented in the cross-platform Unity environment: the game character, obstacles, bonuses and other game elements were created and configured; developed game mechanics and game levels. To control the character, obstacles, bonuses, etc., scripts have been developed using the C# programming language. Work with graphics and audio effects has been adjusted to create the atmosphere of the game.

ЗМІСТ

ВСТУП

Розробка ігрових додатків відкриває безмежні можливості для творчості та розвитку навичок у сфері програмування та дизайну ігрових світів. Unity – це одна з найпопулярніших і найпотужніших платформ для розробки ігор.

Метою кваліфікаційної роботи є розробка гри у стилі Ранер на платформі Unity. Unity – це кросплатформне середовище розробки програмного забезпечення, спеціалізована на створенні ігор, візуальних ефектів та інтерактивних додатків. Його використовують для розробки ігор на різних платформах, включаючи ПК, мобільні пристрої, консолі та віртуальну реальність. Unity є дуже популярною середою для розробки ігор, завдяки своїй простоті використання та багатому набору функцій. Він має велику спільноту розробників, яка активно ділиться знаннями та допомагає один одному вирішувати проблеми під час розробки. Крім розробки ігор, Unity також використовується для створення візуальних ефектів у фільмах та інтерактивних додатків для освіти та виробництва.

Ранер – це один із найпопулярніших жанрів ігор, який передбачає безперервний рух головного героя вперед, де йому потрібно уникати перешкод і збирати різні бонуси. Цей жанр дозволяє розробникам ігор створювати динамічні та захоплюючі ігри з високим рівнем складності. У цій кваліфікаційній роботі будуть розглянуті основні кроки розробки гри Ранер на платформі Unity, включаючи створення ігрового світу, розробку механіки гри, обробку взаємодії з гравцем, додавання звукового оформлення та багато іншого. Також у роботі буде досліджено питання оптимізації гри для різних платформ, таких як ПК та мобільні пристрої, Отже, цей проєкт надасть можливість детально ознайомитись з процесом розробки гри в стилі Ранер на платформі Unity, а також здобути практичні навички у програмуванні та дизайні ігор.

1 РЕАЛІЗАЦИЯ КОМП'ЮТЕРНИХ ІГОР

1.1 Ігрова індустрія в сучасному світі

Реалізація комп'ютерних ігор – це процес створення відеоігор, що включає в себе розробку геймплею, створення графіки та звуків, програмування та тестування. Основними етапами розробки ігор є:

– концептуалізація: створення концепції гри, визначення механіки геймплею, характеристик персонажів та умів для перемоги у грі;

– проєктування: розробка сценарію гри, карти рівнів, дизайн персонажів та інших об'єктів, що з'являтимуться у грі;

– розробка: програмування геймплею, графіки, звуків, інтерфейсу та інших аспектів гри;

– тестування: перевірка гри на наявність помилок та недоліків, виправлення помилок;

– випуск: випуск гри на ринок.

У розробці ігор зазвичай задіяні різноманітні спеціалісти, такі як гейм-дизайнери, художники, програмісти, тестувальники та інші. Зазвичай використовуються різноманітні інструменти для розробки ігор, такі як графічні редактори, інтегровані середовища розробки (IDE), фреймворки та бібліотеки для розробки ігор. Комп'ютерні ігри сьогодні відіграють велику роль у суспільстві. Ігри стали невід'ємною частиною культури та розваг, вони впливають на багато аспектів життя людей. Ігри надають можливість відпочинку, спілкування, розвитку креативності, вирішення проблем, розвитку навичок та умінь, а також розваги. Важливо відзначити, що ігри можуть використовуватись у різних галузях, таких як освіта, наука, медицина та бізнес. Однак, як і в будь–якій іншій галузі, існують проблеми, пов'язані з комп'ютерними іграми, такі як залежність, насильство, агресія та інші. Тому важливо ставитися до ігор відповідально та використовувати їх з розумом.

Існує безліч жанрів комп'ютерних ігор. Деякі з найпоширеніших:

– екшен (Action): ігри, в яких гравець керує персонажем, що виконує різні дії, наприклад, біг, стрибки, стрілянину і т.д.;

– стратегії (Strategy): ігри, в яких гравець керує групою персонажів, розвиває свою базу, планує атаки та оборону;

– рольові ігри (RPG): ігри, в яких гравець керує персонажем, розвиває його, виконує квести, бореться з монстрами та іншими гравцями;

– головоломки (Puzzle): ігри, в яких гравець вирішує різні головоломки, головоломки чи завдання;

– гонки (Racing): ігри, в яких гравець керує автомобілем, мотоциклом або іншим транспортним засобом, змагаючись з іншими гравцями або комп'ютером;

– симулятори (Simulation): ігри, в яких гравець керує імітацією реального об'єкта, наприклад авіа–симулятором, симулятором життя і т.д.;

– шутери (Shooters): ігри, в яких гравець керує персонажем, що стріляє зі зброї, що бореться з ворогами;

– аркади: ігри, в яких гравець змагається з іншими гравцями або комп'ютером у виконанні різних завдань;

– спортивні ігри (Sports): ігри, в яких гравець керує командою або окремим спортсменом у симуляції різних видів спорту.

Також існують ігри, які створені для ігрових консолей, мобільні ігри, браузерні ігри. Мобільні ігри стають все більш популярними та доступними, і багато людей грають у них у свій вільний час.

Наразі ігри розробляються в різних країнах світу, включаючи США, Канаду, Велику Британію, Німеччину, Францію, Китай, Японію, Корею та інші. Основними рушіями для створення ігор є Unity, Unreal Engine, CryEngine та інші. Крім того, багато компаній використовують власні рушії розробки ігор. Ігровий рушій – це програмне забезпечення, яке використовується для створення та розробки комп'ютерних ігор. Він надає розробникам все необхідне для створення гри: від графіки та звуку до

фізичного двигуна та системи штучного інтелекту. Існує безліч ігрових рушіїв, які розробники можуть використовуватиме створення ігор. Деякі з найбільш популярних ігрових рушіїв:

– Unity – це один із найпопулярніших ігрових середовищ розробки, який дозволяє створювати ігри для різних платформ, включаючи мобільні пристрої, комп'ютери та ігрові консолі;

– Unreal Engine – це ігрове середовище, створене компанією Epic Games. Він використовується для створення ігор різних жанрів, від шутерів до платформерів;

– CryEngine – це середовище, створений компанією Crytek. Він широко використовується для створення ігор з відкритим світом, включаючи такі ігри як Far Cry і Crysis;

– GameMaker Studio – це IDE, яке дозволяє створювати ігри без знання програмування. Він часто використовується для створення 2D-ігор.

Ці ігрові середовища надають різні інструменти та функції, які полегшують процес розробки ігор, дозволяють розробникам створювати більш реалістичні та інтерактивні ігрові світи, покращують продуктивність гри та забезпечують високу якість графіки та звуку.

1.2 Поняття сценарію гри

Сценарій гри в стилі Ранер (або Endless Runner) визначає основні характеристики гри, такі як механіка гри, ігрові об'єкти, їхня поведінка, а також різні умови, що впливають на ігровий процес. Основна механіка гри в стилі Ранер полягає в тому, що гравець керує персонажем, який автоматично біжить уперед, і повинен уникати перешкод на своєму шляху. У сценарії гри необхідно визначити, які об'єкти є перешкодами, які об'єкти можна збирати, а також які бонуси та нагороди можна отримати за досягнення певних цілей.

Сценарій також визначає, як гравець керуватиме персонажем. Зазвичай це робиться за допомогою сенсорного екрана мобільних пристроїв або клавіатури на комп'ютері. Сценарій може також включати різні рівні складності та різні режими гри, такі як виживання або час виконання рівня.

Графіка та звукові ефекти також є важливими елементами сценарію гри у стилі Ранер. Сценарій повинен визначати, які графічні елементи будуть використовуватися для персонажа, перешкод та бонусів, а також які звукові ефекти використовуватимуться для створення атмосфери гри.

Також у сценарії гри потрібно визначити різні умови, які впливають на ігровий процес, наприклад швидкість руху персонажа, частоту появи перешкод, кількість життів, час, за який потрібно пройти рівень, та інші параметри, які можуть змінюватись в залежності від рівня складності гри. Всі ці параметри та умови повинні бути налаштовані таким чином, щоб створити цікавий та захоплюючий ігровий процес, який викликатиме бажання гравця продовжувати грати та покращувати свої результати.

Крім того, сценарій гри в стилі Ранер має враховувати можливість прогресування у грі, щоб зберігати зацікавленість гравця. Зазвичай це досягається за рахунок поступового збільшення складності гри, додавання нових перешкод та бонусів, а також даючи можливість гравцеві покращувати свої результати та досягнення.

Загалом сценарій гри в стилі Ранер є важливим елементом розробки гри.

Нижче наведено приклад сценарію гри "Subway Surfers". Subway Surfers – це популярна мобільна гра в жанрі "нескінченний Ранер", розроблена компанією Kiloo та випущена у 2012 році. Гравці грають за персонажа на ім'я Джейк, який бігає по залізничних коліях, ухиляючись від поїздів і перешкод, що наближаються, збираючи монети та інші бонуси на своєму шляху. Гра доступна на багатьох платформах, включаючи iOS, Android та Windows Phone.

Приклад сценарію гри "Subway Surfers":

– гравець починає гру на залізничній станції, де він керує головним героєм на ім'я Джейк, який біжить рейками;

– гравець повинен ухилятися від поїздів, що наближаються, перестрибувати через перешкоди і збирати монети;

– гра триває доти, доки головний герой не зіткнеться з поїздом або не настане на перешкоду;

– щоразу, коли гравець збирає монету, його рахунок збільшується;

– гравець може використовувати монети для покупки нових персонажів, різних посилень та оновлень, які допоможуть йому продовжувати біг;

– гра має різні місцевості, такі як міські вулиці, метрополітен і т. д., і кожна місцевість має свої власні перешкоди та унікальний дизайн;

– гравець може конкурувати з іншими гравцями, збирати досягнення та лідирувати у рейтингах.

2 ОСНОВНІ НАЛАШТУВАННЯ ПРОЄКТУ

2.1 Особливості рушія Unity

Unity – це кросплатформний ігровий рушій, розроблений компанією Unity Technologies. Він дозволяє створювати ігри для широкого спектру платформ, включаючи комп'ютери, мобільні пристрої, консолі та віртуальну реальність. Серед особливостей Unity можна виділити:

– широкі можливості: Unity підтримує різні типи ігор та проєктів, включаючи 2D та 3D ігри, ігри з різними жанрами, а також AR та VR проєкти;

– легкість використання: Unity має інтуїтивно зрозумілий інтерфейс і безліч інструментів, які спрощують розробку ігор;

– кросплатформність: Unity дозволяє створювати ігри для різних платформ, таких як Windows, Mac, Linux, iOS, Android, Xbox, PlayStation та інші;

– підтримка різних мов програмування: Unity підтримує кілька мов програмування, таких як C#, C++, JavaScript та інші;

– велика спільнота: Unity має величезну спільноту розробників, де можна знайти відповіді на багато питань та підказки щодо розробки ігор;

– asset Store: Unity Asset Store пропонує широкий вибір готових ресурсів, таких як моделі персонажів, анімації, текстури та інші, які можуть бути використані в проєкті;

– можливості розширення: Unity дозволяє створювати власні інструменти та плагіни для оптимізації процесу розробки ігор.

Через свої широкі можливості та легкість у використанні, Unity став одним з найбільш популярних ігрових движків у світі. Він має простий і зрозумілий інтерфейс, який робить його досить легким для вивчення творцями ігор, що починають. Одна з основних причин, чому Unity є гарним

для початківців, полягає в тому, що він використовує мову програмування C#[3]. C# вважається однією з найдоступніших і найпростіших розуміння мов програмування. Завдяки цьому новачки можуть швидко навчитися створювати ігри з використанням Unity. Крім того, Unity має велику спільноту користувачів, які надають навчальні матеріали та ресурси для вивчення різних тонкощів середовища розробки, серед яких: відеоуроки, форуми та блоги. Це допомагає новачкам швидко навчатися та вирішувати проблеми, що виникають у процесі створення ігор.

Нарешті, Unity має безкоштовну версію, яку можна використовувати для навчання та створення ігор. Безкоштовна версія має всі необхідні функції для створення базових ігор і є відмінним інструментом для творців ігор, що починають. Серед найпопулярніших ігор, створених на Unity, можна виділити:

– Among Us – розрахована на багато користувачів гра, в якій гравці грають за членів команди на космічному кораблі, один з яких є зрадником;

– Ori and the Blind Forest – ще один платформер з елементами RPG, в якому гравець грає за лісового духу Ori, який бореться з темною силою, що охопила ліс;

– Cuphead – шутер з боковою прокруткою, виконаний у стилі 1930–х років, де гравець грає за чашку та бореться з босами у нелегкому світі;

– Cities: Skylines – стратегія симулятор містобудування, в якій гравець будує та керує своїм містом, піклуючись про добробут його мешканців.

Щоб почати працювати на Unity, потрібно зареєструватися на офіційному сайті розробника, завантажити Unity Hub, і в ньому вибрати версію IDE, на якій приходитиме створення гри.

У цій кваліфікаційній роботі створено гру в стилі Ранер. Ігри в стилі Ранер, також відомі як «нескінченні» або «нескінченні Ранери, це вид ігор, в яких гравець керує персонажем, який біжить нескінченним шляхом, уникаючи перешкод, збираючи бонуси і намагаючись протриматися

якнайдовше. Такі ігри зазвичай мають простий геймплей та управління, що робить їх доступними для широкої аудиторії, включаючи новачків в іграх. Більшість ранерів мають яскраву та привабливу графіку, а також захоплюючу музику, яка допомагає створити атмосферу та напругу у грі. Серед популярних Ранерів, розроблених на рушію Unity, можна виділити:

– Subway Surfers – гравець грає за персонажа, що біжить залізницею, уникаючи перешкод і збираючи монети;

– Temple Run – гравець грає за археолога, що біжить лабіринтом джунглів, уникаючи перешкод і збираючи монети;

– Vector – гравець грає за паркурщика, що біжить по міському даху, уникаючи перешкод та ухиляючись від переслідування;

– Jetpack Joyride – гравець грає за наукового працівника, який керує реактивним ранцем і літає лабораторією, уникаючи перешкод і збираючи монети.

Щоб почати працювати на Unity, потрібно зареєструватися на офіційному сайті розробника, завантажити Unity Hub, і в ньому вибрати версію IDE, на якій приходитиме створення гри. Після повної інсталяції і починається повна робота над створенням гри. У програмі Unity Hub створюємо новий проєкт, обираємо 2D вигляд (Рис. 2.1).

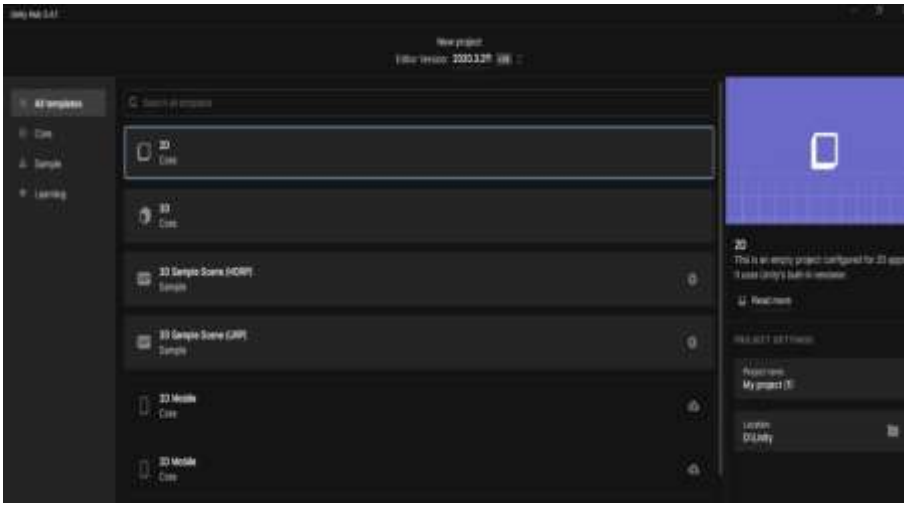


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.1 – Створення проєкту

Отримаємо вікно у вигляді, представленому на Рис. 2.2, та починаємо підготовлювати базу для створення проєкту.

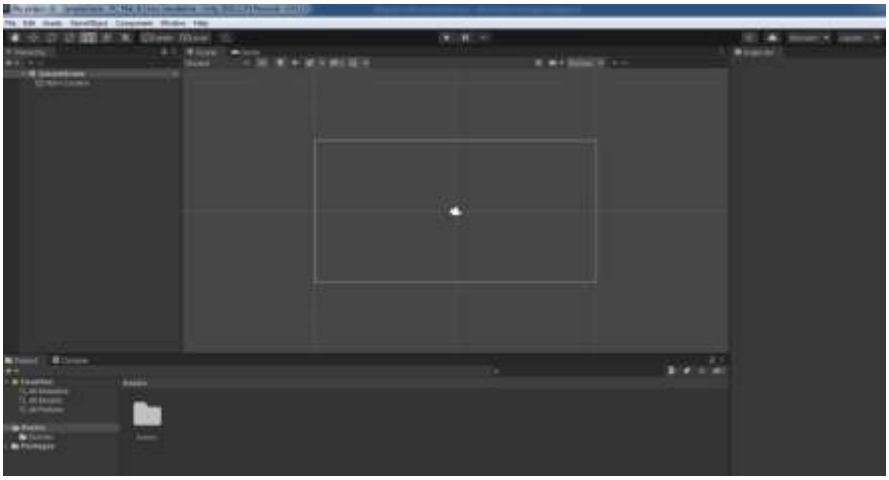


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.2 – Початкове вікно проєкту

2.2 Налаштування конфігурацій проєкту

Найголовніше в аспекті налаштування проєкту, це зрозуміти, на який охоплення людей буде ця гра. У Unity кожна платформа має свої власні налаштування складня, які визначають, як додаток буде збиратися та працювати на певній платформі. Ось основні особливості кожної платформи Build Setting у Unity:

a) PC, Mac & Linux Standalone:
   - вибір платформи: Windows, Mac чи Linux;
   - розрядність: 32–bit чи 64–bit;
   - тип білда: Debug або Release;
   - параметри складня: опції складня, які залежать від платформи, наприклад, налаштування екрана та увімкнення/вимкнення функцій;

б) Android:

- архітектура: вибір архітектури процесора, яка визначає, як програма буде працювати на пристроях Android;
- тип білда: Debug або Release;
- маніфест: конфігураційний файл, який визначає інформацію про пристрій, який використовуватиме програму;
- підпис: сертифікат, який використовується для підписання програми перед завантаженням на Google Play Store;

в) iOS:

- тип білда: Debug або Release;
- схема складня: налаштування складня, такі як вибір цільового пристрою та увімкнення/вимкнення функцій;
- provisioning Profile: файл, який використовується для ідентифікації та перевірки розробника та програми перед його завантаженням на пристрої iOS;
- ключі та сертифікати: використовуються для підписання та складня програми;

г) WebGL:

- тип білда: Debug або Release;
- розмір буфера: розмір буфера, який використовується для завантаження та відтворення веб–сторінки з WebGL–програмою;
- стиснення текстур: увімкнення/вимкнення стиснення текстур для зменшення розміру файлів, що завантажуються;

д) Universal Windows Platform:

- тип білда: Debug або Release;
- тип пакета: вибір типу пакета програми, наприклад APPX або MSIX;
- розмір буфера: розмір буфера, який використовується для завантаження та відтворення програми;

– маніфест: конфігураційний файл, який визначає інформацію про пристрій, який використовуватиме програму.

Кожна з перерахованих вище вузько спрямована на певну сферу користування пристроями. Найпопулярніша звичайно PC, Mac & Linux Standalone, так як більшість грають в ігри на персональних комп'ютерах через зручність і простоту встановлення гри на платформі Unity, коли лише потрібна оболонка Unity [4].

У даній кваліфікаційній роботі буде розглянуто платформу WebGL, тому що на цій платформі легше реалізувати гри в інтернеті. Ця платформа створена для публікації гри у браузері, а браузерні ігри завжди мають попит в інтернеті. Для них не потрібно нічого завантажувати, лише зайти на сайт та почати грати.

2.3 Створення ігрових активів

Один із найголовніших аспектів гри це її візуальні об'єкти: головний персонаж, перешкоди, задній фон та оформлення ігрових написів. При створенні текстур для гри спочатку потрібно вигадати ідею для реалізації візуальних об'єктів. Текстури в іграх використовуються для надання об'єктам на сцені візуального вигляду та створення ілюзії тривимірного простору [7].

Текстури можуть бути зображеннями, фотографіями, анімаціями або іншими графічними елементами, які накладені на поверхні 3D-моделей або використовуються в 2D-іграх для створення більш реалістичної картинки. Крім того, текстури можуть служити для створення ефектів, таких як світіння, відображення та тіні, які покращують візуальний досвід гри та допомагають створити більш реалістичне оточення.

Найпростішою ідеєю була звичайна людина, що біжить по дорозі, і машини йому на зустріч, але це занадто просто і не цікаво. У зв'язку з останніми подіями у світі, був момент життя у всіх, коли був оголошений

тотальний локдаун через коронавірус. Мені стало цікаво, як це можна інтерпретувати у гру. Найцікавішим на мою думку став "літаючий чоловік у літаку у вигляді шприца" (Рис. 2.3), а на зустріч йому летять бактерії вірусу.

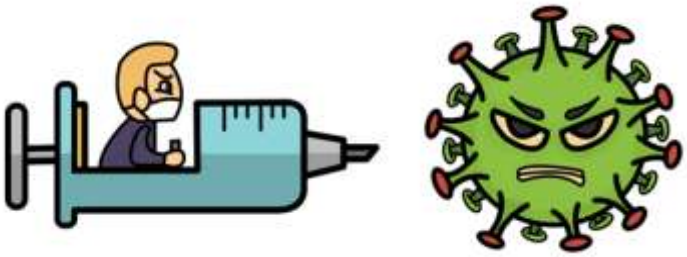


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.3 – Головні дійові особи

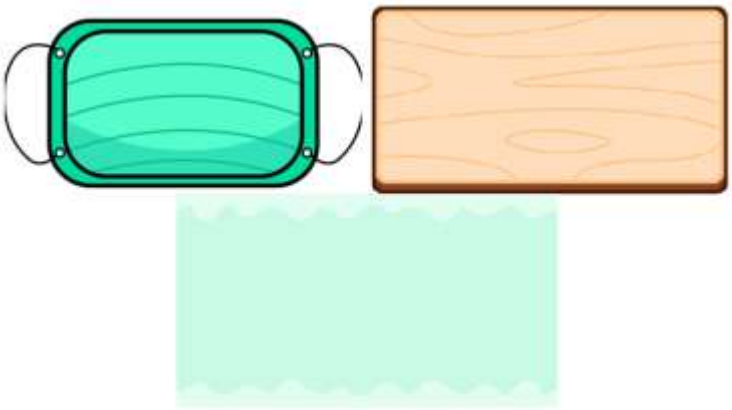


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.4 – Допоміжні текстури гри

Для створення цих об'єктів було залучено художника, який використовував графічний планшет для реалізації цієї ідеї (Рис. 2.4). Всі картинки були одразу поміщені в папку "Materials" для зручності їх використання.

3 СТВОРЕННЯ ГРИ РАНЕР

Створення ігор – це захоплюючий процес, який поєднує творчість, програмування та дизайн. Сьогодні, завдяки розвитку сучасних ігрових двигунів, створення ігор доступне кожному охочому. Unity – один з найпопулярніших ігрових движків на ринку, який дозволяє створювати ігри для багатьох платформ, включаючи комп'ютери, мобільні пристрої, ігрові консолі і навіть віртуальну реальність. У цьому розділі розглянуто процес створення гри на Unity з нуля.

Використано C# як мову програмування, яка є однією з основних мов розробки в Unity.

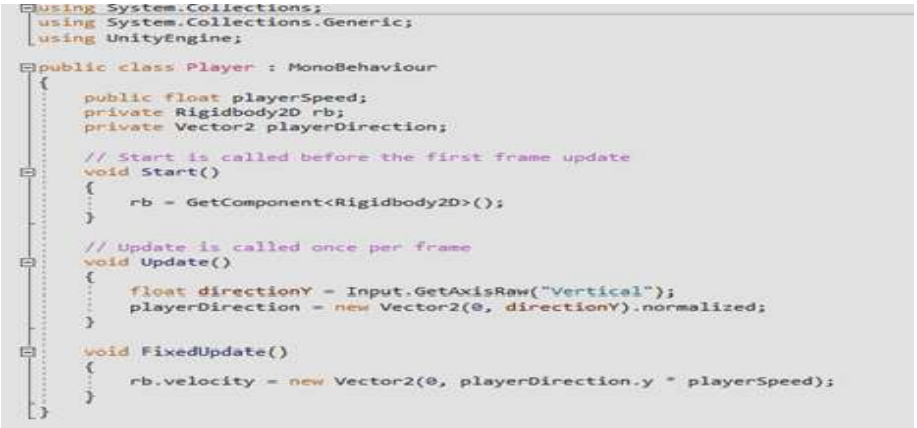
У процесі створення гри розглянуто:

– створення ігрового світу та об'єктів;
– призначення властивостей та компонентів ігрових об'єктів;
– написання скриптів на C#;
– створення інтерфейсу користувача;
– оптимізація продуктивності гри;
– експорт гри на різні платформи.

Результатом кваліфікаційної роботи є гра на основі Unity, яку можна публікувати у Інтернеті для загального користування. Однією з головних переваг загального доступу – можливість грати на будь-якому пристрої, що має підключення до Інтернету, що дозволяє грати в будь-який час і в будь-якому місці. Це також означає, що ігри можуть бути швидко та легко оновлені, дозволяючи гравцям отримувати доступ до нових контентів та покращеної геймплейної механіки.

3.1 Створення головного об'єкту

В перш чергу була створена папка “Scripts”, в якій будуть розміщені всі скрипи цієї гри. Найперший скрипт створений для головного персонажа, він називається: ”Player” (Рис. 3.1).



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

public class Player : MonoBehaviour
{
    public float playerSpeed;
    private Rigidbody2D rb;
    private Vector2 playerDirection;

    // Start is called before the first frame update
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    void Update()
    {
        float directionY = Input.GetAxisRaw("Vertical");
        playerDirection = new Vector2(0, directionY).normalized;
    }

    void FixedUpdate()
    {
        rb.velocity = new Vector2(0, playerDirection.y * playerSpeed);
    }
}
```

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.5 – Скрипт Player

Public float playerSpeed – змінна "playerSpeed" зберігатиме значення швидкості гравця у грі.

Private Rigidbody2D rb – компонент Rigidbody2D використовується для імітації фізичної поведінки об'єктів у 2D-просторі. Він містить інформацію про масу об'єкта, його швидкість та інші фізичні параметри, і дозволяє об'єктам реагувати на фізичні сили, такі як гравітація або зіткнення з іншими об'єктами.

Private Vector2 playerDirection – змінна "playerDirection" зберігатиме вектор напряму, в якому рухається гравець у грі. Тип "Vector2" є двомірним вектором у двомірному просторі. Вектор може бути використаний для представлення напрямку руху об'єкта, його швидкості, позиції або інших величин, пов'язаних із рухом об'єкта у просторі.

Метод Update ()

Цей метод викликається один раз на кожен кадр і використовується для оновлення логіки гри. В даному випадку метод Update() визначає напрямок

руху гравця в залежності від введення гравця з клавіатури. Змінна directionY визначається як значення, отримане з виклику методу GetAxisRaw("Vertical"), який повертає значення в діапазоні від -1 до 1 залежно від того, чи натиснута клавіша "вгору" або "вниз". Потім значення directionY використовується для створення нового вектора playerdirection за допомогою методу new Vector2(0, directionY). Метод normalized використовується, щоб нормалізувати вектор і встановити його довжину 1, таким чином, гравець буде рухатися з однаковою швидкістю в будь-якому напрямку.

Метод *void FixedUpdate()*

Цей метод викликається певну кількість разів на секунду і використовується для виконання фізичних розрахунків та оновлення фізики у грі. В даному випадку метод FixedUpdate() встановлює швидкість гравця, використовуючи посилання на компонент Rigidbody2. Весь цей скрипт прикріплюємо до створеного 2D об'єкту, наякий так само накладена текстура (Рис. 2.1).

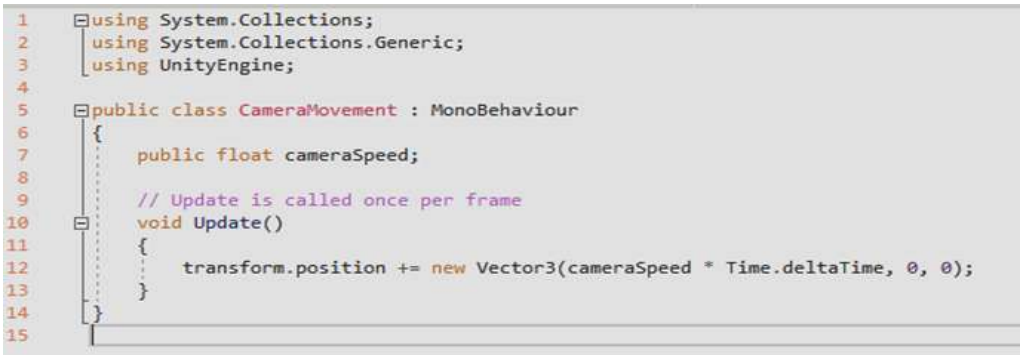
3.1.1 Створення ігрової камери

Головне в іграх у жанрі Ранер це ігрова камера, яка слідує за гравцем. Це робиться для того, щоб все було схоже на те, що гравець рухається вперед. Але насправді гравець просто стоїть на місці, а єдиним об'єктом, що рухається, є камера. Створюється наступний скрипт під назвою “Camera Movement” (Рис. 3.2)

Public float cameraSpeed ця змінна визначає швидкість руху камери горизонтальному напрямку.

Transform.position += new Vector3(cameraSpeed * Time.deltaTime, 0, 0) – цей рядок коду використовується для зміни позиції камери. Вона використовує властивість transform.position, щоб отримати поточну позицію камери, а потім додає до неї новий вектор, створений за допомогою методу

new Vector3. Цей вектор визначає, як сильно повинна рухатися камера в кожному напрямку. У цьому випадку камера рухається лише праворуч на значення, задане в змінній cameraSpeed, помножене на Time.deltaTime. Time.deltaTime використовується для врахування змін часу між кадрами, щоб швидкість руху камери залишалася постійною незалежно від частоти кадрів [6].



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

public class CameraMovement : MonoBehaviour
{
    public float cameraSpeed;

    // Update is called once per frame
    void Update()
    {
        transform.position += new Vector3(cameraSpeed * Time.deltaTime, 0, 0);
    }
}
```

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.6 – Складові скрипта для камери

Створюється об'єкт “Main Camera”, який поміщений в головний об'єкт гри “Game Manager”. Скрипт який повязаний з рухом камери накладаємо на створений раніше об'єкт “Main Camera”.

3.1.2 Створення циклічного заднього фону

Наступним кроком у налаштуванні камери буде підстановка заднього фону та його нескінченна генерація. Також створюється порожній об'єкт, який називає “Background”. У папці де знаходяться всі текстури, третя текстура є тлом гри (Рис.3.2). У його налаштуваннях потрібно змінити Wrap Mode. Змінити на такий як вказано на Рис. 3.3.

Після накладання текстури на об'єкт її теж потрібно налаштувати. Змінюється значення в клітинці Shader з Standart на Unity/Texture. Шейдери в Unity – це програми, які визначають, як відображаються графічні об'єкти в

грі. Вони дозволяють створювати різноманітні ефекти та покращувати візуальну якість гри. Шейдери працюють на рівні графічної карти, що дозволяє їм бути швидше та ефективніше, ніж використання більш традиційних методів відображення графіки. Вони можуть використовуватися як для створення простих ефектів, так і для створення складних ефектів, які потребують високого ступеня деталізації та реалістичності. Шейдери можуть бути створені як мовою програмування, і з використанням графічних інтерфейсів.

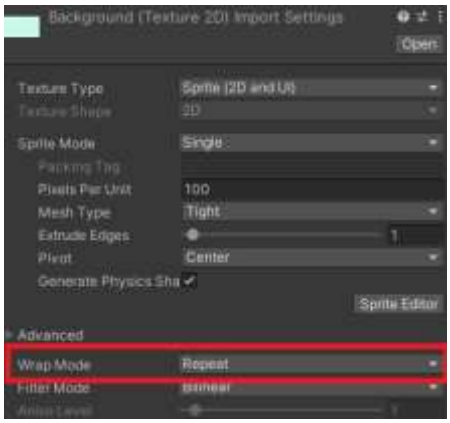


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.7 – Налаштування заднього фону

В цілому, шейдери дозволяють створювати різноманітні візуальні ефекти, які роблять гру більш красивою та цікавою [1]. Так само підганяємо розмір фону під розмір камери та гравця, щоб все виглядало правильно. Поміщаємо об'єкт “Background” у теку “Game Manager” (Рис. 3.4).

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.8 – Остаточний вигляд фону

Далі створюємо скрипт “Looping Backround” (Рис. 3.5).

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

public class LoopingBackground : MonoBehaviour
{
    public float backgroundSpeed;
    public Renderer backgroundRenderer;

    // Update is called once per frame
    void Update()
    {
        backgroundRenderer.material.mainTextureOffset += new Vector2(backgroundSpeed * Time.deltaTime, 0f);
    }
}
```

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.9 – Новий скрипт для гри

Цей скрипт "Looping Background" відповідає за анімацію заднього фону, що нескінченно повторюється в грі. Змінна "backgroundSpeed" відповідає за швидкість руху заднього фону. Змінна "backgroundRenderer" є посиланням на компонент Renderer, який відображає задій фон на екрані [2].

У методі "Update" кожен кадр відбувається оновлення та усунення текстури заднього фону. Це відбувається шляхом додавання до поточного усунення нового вектора "Vector2", де "backgroundSpeed" множиться на

"Time.deltaTime", а це дозволяє робити рух заднього фону плавним і залежним від часу. Таким чином, скрипт дозволяє створити ефект нескінченної анімації заднього фону у грі, який рухатиметься зі швидкістю, заданою у змінній "backgroundSpeed".

3.1.3 Створення небезпечних об'єктів для гравця

Програш у Ранерах відбувається, коли гравець не може подолати певний рівень, перебуває у зіткненні з перешкодою або не встигає зробити необхідну дію вчасно. У ранерах зазвичай дуже швидкий темп гри, і гравець повинен швидко реагувати на те, що відбувається на екрані. Невдача може статися, якщо гравець не встиг стрибнути, не помітив перешкоди або зробив неправильний вибір дії у критичний момент. Крім цього, програш може відбуватися через технічні проблеми, такі як зависання гри або збій в управлінні. Це може бути пов'язано з неправильною роботою апаратного або програмного забезпечення на пристрої, на якому гра працює. В цілому, програш у ранерах може бути викликаний різними причинами, але, як правило, він відбувається через те, що гравець не зміг швидко і точно реагувати на те, що відбувається на екрані[8]. У цьому додатку користувач проходитиме об'єкти за допомогою стрілок на клавіатурі.

Для початка потрібно створити порожній ігровий об'єкт “Obstacle”, в який буде поміщена текстура у відповідному вигляді (Рис. 2.1), яка знаходиться праворуч, під назвою “Obstacle Sprite”. Весь цей об'єкт переносимо у Prefab.

Unity Prefab – це попередньо зібраний об'єкт, який можна використовувати у грі кілька разів. Це означає, що можна створити об'єкт з певними властивостями та компонентами, зберегти його як Prefab та використовувати його у різних сценах вашої гри. Префаби дозволяють скоротити час, необхідний для створення та налаштування об'єктів у грі.

Завдяки цьому елементу можна створювати та настроювати об'єкти в одній сцені, а потім використовувати їх в інших сценах без необхідності повторно створювати та налаштовувати їх. Крім того, можно використовувати префаби для створення екземплярів об'єктів у реальному часі, наприклад, створювати нових ворогів або декорації на льоту, що може бути корисним для динамічної зміни рівня та управління його складністю. Як в нашому випадку, коли буде відбуватися генерація нових об'єктів, які будуть елементом програшу в грі. Префаби також дозволяють вам легко вносити зміни до об'єктів, змінюючи лише один префаб, а всі екземпляри, створені на його основі, автоматично успадковують ці зміни. Це зручно для роботи з великою кількістю об'єктів у грі та полегшує процес розробки та тестування ігрових об'єктів.

Для отримання генерації об'єктів потрібно створити наступний скрипт під назвою “Spawn Obstacle” (Рис. 3.6). У ньому буде механізм випадкової появи противників гравця, а так само в налаштуваннях цього скрипту можна змінювати межі екрана, де з'являтимуться нові об'єкти.

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

public class SpawnObstacles : MonoBehaviour
{
    public GameObject obstacle;
    public float maxX;
    public float minX;
    public float maxY;
    public float minY;
    public float timeBetweenSpawn;
    private float spawnTime;

    // Update is called once per frame
    void Update()
    {
        if(Time.time > spawnTime)
        {
            Spawn();
            spawnTime = Time.time + timeBetweenSpawn;
        }
    }

    void Spawn()
    {
        float randomX = Random.Range(minX, maxX);
        float randomY = Random.Range(minY, maxY);

        Instantiate(obstacle, transform.position + new Vector3(randomX, randomY, 0), transform.rotation);
    }
}
```

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.10 – Скрипт для генерації об’єктів

Всередині скрипта задаються параметри розташування та часу між спаунами перешкод, а також сама перешкода, яка буде створюватись. Цей скрипт відповідає за створення ігрових об'єктів–перешкод у випадкових місцях на сцені. У змінних maxX, minX, maxY та minY задаються максимальні та мінімальні значення координат X та Y відповідно, у яких

можуть бути створені перешкоди. Скрипт відповідає за генерацію перешкод у певних місцях на сцені. У змінних задаються параметри місця появи (мінімальні та максимальні значення за X та Y координатами), префаб перешкоди та час між появами нових перешкод. У функції Update() перевіряється, чи пройшло достатньо часу для генерації нової перешкоди. Якщо минуло, то викликається функція Spawn(), яка генерує випадкові координати X і Y у заданих діапазонах, використовуючи функцію Random.Range(). Потім створюється екземпляр перешкоди за допомогою функції Instantiate(), яка використовує заданий префаб перешкоди і додає випадкові координати до координат позиції нашого об'єкта, щоб розмістити його у випадковому місці на сцені. У змінній spawnTime зберігається час останньої генерації перешкоди, щоб знати, коли наступна перешкода має бути згенерована. У змінній obstacle вказується префаб ігрового об'єкта–перешкоди, який має бути налаштований заздалегідь. Це дозволяє створювати різні типи перешкод, залежно від вимог гри [10].

У змінній timeBetweenSpawn задається інтервал часу, через який створюється нова перешкода. Коли час Time.time перевищує spawnTime, викликається функція Spawn(), яка створює нову перешкоду за допомогою функції Instantiate(). У функції Spawn() використовується функція Random.Range() для отримання випадкових значень координат X та Y у заданих діапазонах. Потім координати передаються функції Instantiate() як параметр position [6]. Створений об'єкт–перешкода матиме ту саму орієнтацію і масштаб, що його префаб. Таким чином, при кожному спауні створюється нова перешкода із випадковим розташуванням на сцені. Усі параметри перешкоди та частота його генерації задаються в інспекторі об'єкта, до якого прикріплено цей скрипт.

Наступним кроком у реалізації цього скрипту буде створення порожнього об'єкта під назвою “Spawn Point”. Цей об'єкт створено для реалізації дій скрипта, у виконанні завдання генерації об'єктів максимальної висоти X і Y.

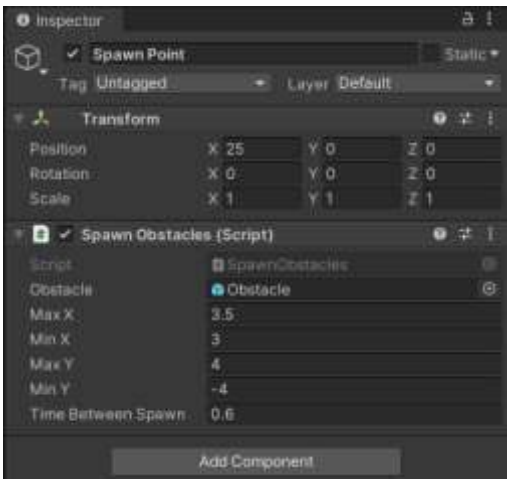


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.11 – Налаштування скрипта Spawn Obstacles

Сам вид налаштування виглядає дуже просто, потрібно додати з папки префаб текстуру “вірусу” і додати її в основну папку де знаходяться всі ігри, що складають. Після його додавання потрібно у вкладці сцена за допомогою ручного підбору висоти записати їх у відповідні осередки (Рис. 3.7). Також виставити частоту появи об'єктів, чим менше буде значення, тим простіше буде гра і відповідно чим більше значення тим складніше буде гра.

Проміжний вид гри з генерацією об'єктів виглядає так (Рис. 3.8):

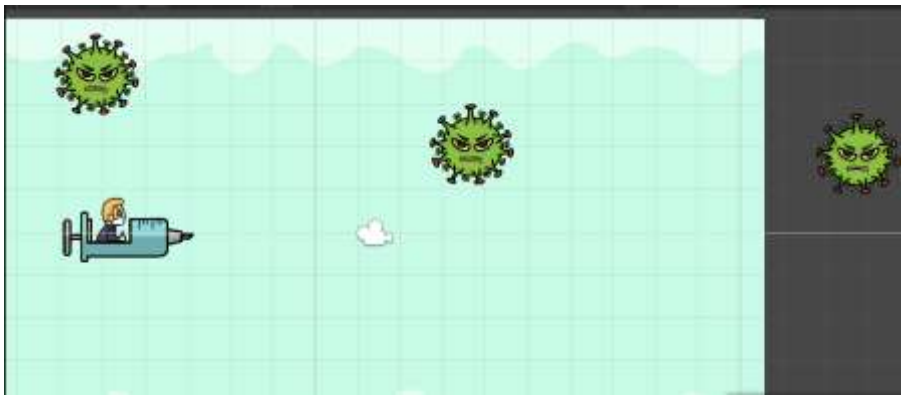


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.12 – Проміжний вигляд гри

3.1.4 Створення колізії та руйнування об'єктів

Колізія (від англ. collision) – це зіткнення двох або більше об'єктів у просторі. В іграх та комп'ютерній графіці колізії використовуються для визначення, коли об'єкти стикаються один з одним, і для симуляції реалістичної фізики та поведінки об'єктів. У Unity колізія – це процес визначення перетину геометричних форм (колайдерів) двох або більше об'єктів у просторі. Колайдери – це компоненти, які можуть бути додані до ігрових об'єктів у Unity, щоб визначити їхню геометричну форму та виявляти зіткнення з іншими об'єктами. Колайдери в Unity можуть мати різні форми: це можуть бути прості геометричні форми, такі як куби, сфери, капсули або складніші форми, створені з безлічі примітивних форм. Колайдери можуть бути прикріплені до об'єктів, які повинні виявляти зіткнення, такі як ігрові персонажі, об'єкти оточення, перешкоди і т.д. Коли два або більше колайдерів перетинаються у просторі, Unity генерує подію колізії. Ця подія може бути використана для виконання різних дій, наприклад, для обробки шкоди, переміщення персонажа, відскоку об'єктів, зміни поведінки гри і т.д. Колізія в Unity також може використовуватися для реалізації фізичної взаємодії між об'єктами, такими як гравітація, тертя, гнучкість і т.д. Для цього в Unity використовується фізичний двигун, який визначає, як об'єкти повинні поводитися при зіткненні та як вони повинні взаємодіяти один з одним. Колізії у Unity відіграють важливу роль у створенні реалістичних та інтерактивних ігрових світів. Завдяки колізіям можна створювати складні сцени з безліччю об'єктів та персонажів, які взаємодіють один з одним та взаємодіють із навколишнім середовищем. Колізії у Unity також дозволяють створювати різні ігрові механіки, такі як пастки, перепони, битви тощо.

Для створення колізії потрібно додати порожній об'єкт, який зберігатиме рамки ігрової зони. Він називатиметься “Borders”, а його складові “Top Border” і Border”, які відповідно до своїх назв створюють колізію ігрового поля. У цих двох комплектуючих потрібно додати

компонент, завдяки якому буде створюватися колізія поля (Рис. 3.9). Він називається “Box Collider 2D”. Після його додавання потрібно визначиться з його розміром і поставити його на кордоні з заднім фоном. Щоб спростити додавання колізії меж гри, робиться копія Top Border`a, тільки змінюється назва на Bottom.

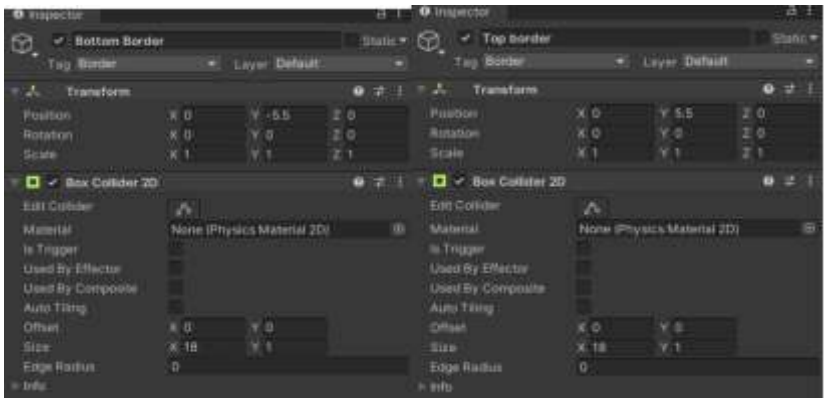


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.13 – Панель налаштування Нижнього та Верхнього кордону гри

Ось такий вид гри отримуємо, після додавання верхньої та нижньої межі гри (Рис. 3.10).

Такий самий процес додавання потрібно застосувати і до моделі Player. Також потрібно додати компонент “Box Collider 2D”, і найголовніше виділити межі гравця вручну, тобто обвести його текстуру.

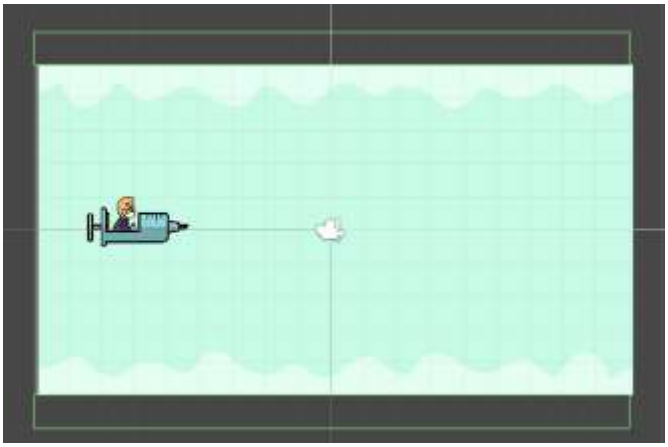


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.14 – Вигляд меж гри

На Рис. 3.11 колізія гравця зроблена червоною лінією, але в Unity вона має інший колір. Через його кольори було прийнято рішення, що краще виділить колізію гравця червоним кольором.

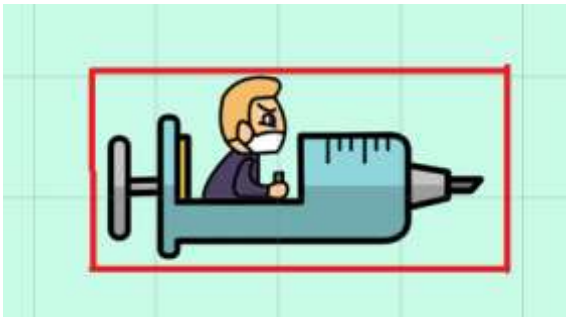


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.15 – Колізія моделі Player

Важливим у таких іграх є оптимізація ігрового процесу. Оптимізація у грі на Unity – це процес покращення продуктивності гри шляхом зменшення навантаження на комп'ютер або мобільний пристрій, на якому запускається гра. При розробці ігор на Unity розробники стикаються з низкою проблем, які можуть знижувати продуктивність та погіршувати досвід гри для користувача. Деякі з цих проблем можуть бути пов'язані із загальними проблемами продуктивності, такими як низька частота кадрів або тривалий час завантаження. Інші проблеми можуть виникати через специфічні проблеми, пов'язані з певними типами контенту, такими як великі текстури, складні моделі або ефекти частинок.

Оптимізація в грі на Unity може включати такі методи:

– зменшення кількості полігонів об'єктів. Чим менше полігонів у об'єкта, тим менші витрати на його малювання і тим більша продуктивність гри;

– використання оптимізованих текстур. Оптимізовані текстури займають менше місця на диску і споживають менше ресурсів під час завантаження та малювання;

– оптимізація освітлення. Висвітлення може займати значну кількість ресурсів, тому його можна налаштовувати таким чином, щоб воно споживало менше ресурсів;

– оптимізація скриптів. Скрипти можуть суттєво впливати на продуктивність гри, тому їх потрібно оптимізувати, видаляючи з них непотрібні операції та поєднуючи їх у більш компактні та ефективні;

– використання LOD (Level of Detail). LOD дозволяє замінювати деталізовані об'єкти менш деталізовані при видаленні від них гравця, що дозволяє знизити навантаження на процесор;

– оптимізація колізій. Колізії – це процес визначення зіткнень між об'єктами. Оптимізація колізій допоможе прискорити процес обробки зіткнень і зменшити навантаження на процесор;

– використання об'єктів Unity Pooling. Unity Pooling – це техніка, яка дозволяє використовувати повторно об'єкти замість створення нових щоразу під час їх використання. Це дозволяє зменшити витрати на створення та знищення об'єктів та прискорити роботу гри.

Оптимізація гри на Unity важлива, тому що вона дозволяє створювати якісніші ігри з кращими візуальними ефектами та вищою частотою кадрів при використанні меншої кількості ресурсів комп'ютера.

Оптимізація гри – це процес покращення продуктивності гри, щоб вона працювала більш ефективно на цільовій платформі (наприклад, на комп'ютері або мобільному пристрої). Оптимізація гри є важливим процесом у розробці ігор, оскільки дозволяє зробити гру доступною для більш широкої аудиторії та забезпечити найкращий ігровий досвід для користувачів. Вона включає усунення затримок, зниження використання ресурсів, таких як процесор, пам'ять та графіка, та забезпечення більш плавної та стабільної роботи гри в цілому. І для оптимізації цієї гри, створено скрипт який за функціоналом знищуватиме пройдені об'єкти гравцем, після зіткнення із заднім кордоном гри, яка створена за принципом верхньої та нижньої меж.

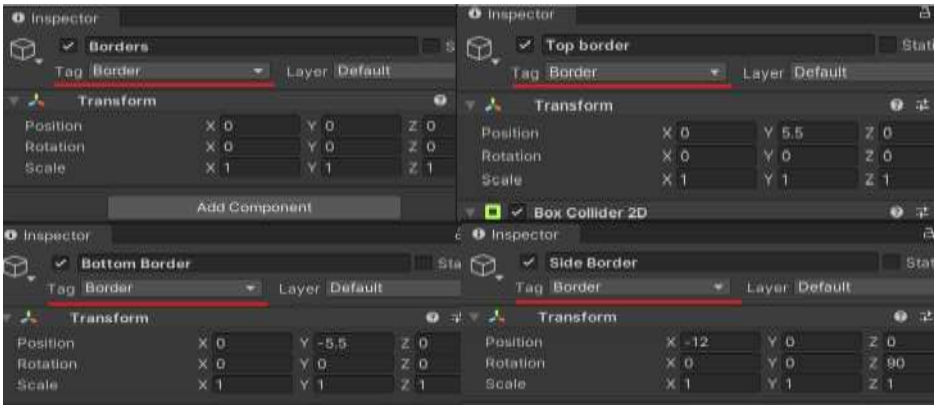


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.16 – Налаштування тега для меж гри

У розділі “Borders”, де знаходяться раніше створенні межі, створюється ще одне така сама, тільки розгорнута на 90 градусів. Нова межа матиме назву “Side Border”. Всі матеріали, що знаходяться в цій папці, для зручності їх використання, потрібно додати до їх опису тег Border. Також щоб це працювало потрібно додати колізію об'єкту (Рис 3.12). Це буде так само важливо, додавання в гру цього елементу, щоб отримати функції програшу в грі.

Раніше вже було додавання колізії для об'єктів, але тут різниця в тому, що минулого разу об'єкт був прямокутний, а зараз це коло. Завдяки широкому функціоналу Unity, в його середовищі є функція додавання колізії у вигляді кола, вона називається “Circle Collider 2D” (Рис. 3.13). Налаштування колізії також виставляється в ручну, тобто коло заїдається самостійно, підбираючи діаметр під намальовану текстуру. Circle Collider 2D – це компонент фізичного двигуна Unity, який додає до ігрового об'єкта 2D колайдер у формі кола. Він використовується для визначення зіткнень з іншими об'єктами у сцені на основі форми кола (Рис. 3.14).

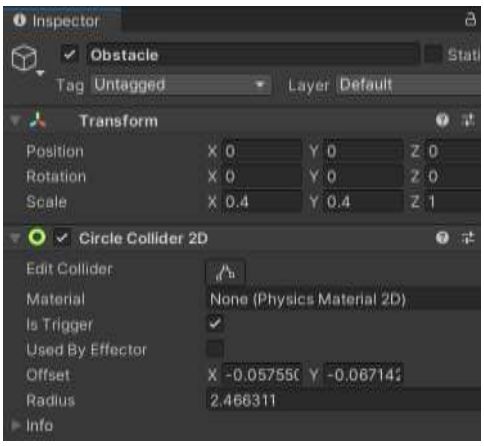


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.17 – Функція Circle Collider 2D

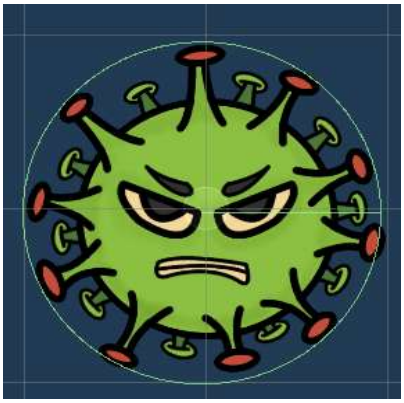


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.18 – Колізія у вигляді кола

У доданому компоненті ще потрібно включити функцію “Is Trigger”. Is Trigger – це опція в Unity, яка може бути включена для колайдерів об'єктів. Коли ця опція увімкнена, колайдер не використовуватиметься для обробки зіткнень у фізичному движку, а використовуватиметься лише для виявлення зіткнень. Це може бути корисним у ряді сценаріїв, наприклад, коли вам потрібно виявити, що гравець перетнув певну область у грі, або щоб взаємодіяти з об'єктами без їхньої фізичної взаємодії. Коли Is Trigger включений для колайдера об'єкта, він генеруватиме події OnCollisionEnter, OnCollisionStay та OnCollisionExit, які повідомляють, коли об'єкт входить у контакт з іншим об'єктом, перебуває в контакті з ним і виходить із контакту

відповідно. Замість OnCollision подій також можна використовувати події OnTriggerEnter, OnTriggerStay та OnTriggerExit для обробки перетинів об'єктів, коли Is Trigger увімкнена. Крім того, використання Is Trigger може допомогти оптимізувати продуктивність вашої гри, оскільки він не витрачатиме обчислювальні ресурси на розрахунок фізичного зіткнення, а лише на виявлення зіткнення [4].

Далі нам знадобиться скрипт під назвою “Obstacle” (Рис. 3.15). У ньому і буде прописано систему знищення об'єктів які залишились позаду гравця.

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.19 – Скрипт Obstacle

Цей скрипт відповідає за поведінку перешкод у грі. Коли об'єкт з цим скриптом стикається з іншим об'єктом, у якого встановлено компонент Collider2D і тег "Border", об'єкт з цим скриптом знищується за допомогою функції Destroy (). У цьому скрипті також оголошено змінну player типу GameObject, але ця змінна не використовується.

private GameObject player; – оголошує приватне поле player, яке використовуватиметься для зберігання посилання на об'єкт гравця.

private void OnTriggerEnter2D (Collider2D collision) – метод, який викликається при зіткненні об'єкта з колайдером. У цьому випадку використовується 2D колайдер. При зіткненні відбувається перевірка на те,

чи є тег об'єкта "Border", що зіткнувся. Якщо так, об'єкт знищується за допомогою методу Destroy(this.gameObject). Це потрібно для того, щоб об'єкт не виходив за межі екрана та не навантажував пам'ять, якщо він більше не потрібний. Загалом цей скрипт використовується для видалення об'єкта, коли він стикається з межею ігрового поля.

Цей скрипт потрібно додати на об'єкт Obstacle. А також треба додати функцію “Rigidbody 2D” (Рис. 3.16).

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.20 – Функція Rigidbody 2D

Rigidbody 2D – це компонент Unity, який додає фізичну поведінку до ігрового об'єкта у двовимірному просторі. Він дозволяє об'єктам рухатися, зіштовхуватися, обертатися та взаємодіяти з іншими об'єктами у сцені. Коли об'єкту додається Rigidbody 2D, він стає фізичним об'єктом і починає взаємодіяти з іншими об'єктами у сцені. Rigidbody 2D використовує фізичні закони, такі як закони Ньютона для визначення руху об'єкта та його зіткнень з іншими об'єктами.

Деякі з властивостей Rigidbody 2D включають:

– Mass: визначає, наскільки важко або легко рухати об'єкт;

– Gravity Scale (Маштаб гравітації): визначає силу тяжіння, що діє на об'єкт;

– Linear Drag (лінійний опір): визначає, як швидко об'єкт втрачає свою швидкість;

– Angular Drag (кутовий опір): визначає, як швидко об'єкт перестає обертатися;

– Is Kinematic (кінематичний): якщо цей параметр встановлений у true, об'єкт не буде взаємодіяти з іншими об'єктами, але його можна буде переміщати програмно;

– Constraints (обмеження): визначає, які рухи об'єкта будуть обмежені.

Rigidbody 2D може використовуватися для створення реалістичної фізики у грі, такої як падіння об'єктів під впливом гравітації, взаємодія об'єктів при зіткненні тощо.

3.1.5 Система програшу

Щоб створити цю функцію, потрібно скористатися компонентом “Canvas”. Canvas в Unity – це основний компонент інтерфейсу користувача (UI) в Unity, який використовується для створення і відображення 2D–елементів інтерфейсу, таких як кнопки, текстові поля, смуги прокручування, зображення і т.д. Canvas можна порівняти з полотном, на якому можна створювати та розміщувати об'єкти інтерфейсу користувача. Він може бути налаштований на роботу в режимах Screen Space – Overlay, Screen Space – Camera та World Space залежно від того, як потрібно розмістити елементи на екрані.

Основні властивості Canvas в Unity:

– Render Mode (режим відображення) – визначає, як відображатиметься Canvas. Режими відображення включають Overlay, Camera та World Space;

– Pixel Perfect – дозволяє налаштувати розміри Canvas і елементів інтерфейсу користувача для отримання ідеальних, піксельних співвідношень;

– Sorting Order – визначає порядок відображення Canvas у сцені, якщо є кілька елементів інтерфейсу, що знаходяться на одному рівні;

– Reference Resolution – визначає дозвіл, який використовуватиметься розробки інтерфейсу. При цьому елементи UI автоматично масштабуються для відповідності різним екранам та пристроям;

– Scale With Screen Size – дозволяє масштабувати елементи інтерфейсу користувача відповідно до різних роздільних здатностей екранів;

– Canvas Scaler – компонент, який дозволяє налаштувати різні параметри масштабування елементів інтерфейсу користувача.

В цілому, використання Canvas в Unity дозволяє легко створювати і налаштовувати інтерфейс користувача для вашої гри або програми, без необхідності використовувати сторонні бібліотеки або плагіни[10]. Canvas дозволяє розміщувати зображення, текст, кнопки, поля введення та інші елементи інтерфейсу користувача в ігровому світі і змінювати їх положення, розмір, колір та інші властивості в реальному часі. У цій функції відбувається налаштування вікна після ураження гравця, його розміри визначаються самим розробником. Дані вказуються у розділі “Canvas Scaler” (Рис. 3.17).

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.21 – Функція Canvas

Для додавання цих функцій потрібно створити додатковий ігровий матеріал під назвою “Game Over Panel”. Цей елемент гри потрібно помістити в папку “Canvas”. Найголовнішим елементом цієї функції є вікно, що з'являється, після зіткнення гравця з перешкодою. Це зображення (Рис. 1.6) потрібно помістити у “Game Over Panel”, і змінити назву на “Wooden Panel”. І так само вибрати розмір панелі програшу. У кожній грі повинен бути текст, для спрощення розуміння гри. І в цій грі не виняток, на цю панель буде додано текст “Game Over”.

Цей текст також потрібно відредагувати: вибрати бажаний колір, встановити тип шрифту, розташування на панелі.

Один із важливих моментів гри це почати її заново. Отже, потрібно додати кнопку рестарту. З Допомога функціоналу Unity, додаємо елемент “Button”. Button (кнопка) в Unity – це один з основних елементів інтерфейсу користувача, що дозволяє користувачеві взаємодіяти з грою або програмою, клацнувши по ньому. Button є одним із видів компонентів UI в Unity. Button можна додати на екран за допомогою компонента Button на панелі Inspector.

Після додавання кнопки на екран можна налаштувати її текст, колір, зображення та інші властивості, щоб вона відображалася та працювала відповідно до потреб проєкту. Button має кілька подій, які можна використовувати для взаємодії з ним: OnClick, OnEnter, OnExit і т.д. Подія OnClick виникає при натисканні на кнопку, що дозволяє викликати певні методи та функції, які виконують дії, необхідні для проєкту. Кнопки широко використовуються в ігрових проєктах і програмах для створення інтерактивних елементів інтерфейсу користувача, таких як меню, вікна, діалоги, перемикачі і т.д.

Для цієї кнопки створюється окремий елемент гри “Restart Button” (Рис. 3.18).

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.22 – Налаштування клавіші Restart

Так як раніше було вже заготовлена текстура цієї кнопок, то її всього лише потрібно перемістити на “Source Image”. Також додати елемент “TEXT”, з підписом “Restart”, а також його відредагувати як і текст на “Wooden Panel”. Щоб це все запрацювало потрібно, у раніше створеному скрипті “Obstacle” (Рис. 3.19) додати деякі нові функції.

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

public class Obstacle : MonoBehaviour
{
    private GameObject player;
    // Start is called before the first frame update
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.tag == "Border")
        {
            Destroy(this.gameObject);
        }

        else if(collision.tag == "Player")
        {
            Destroy(player.gameObject);
        }
    }
}
```

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.23 – Оновлений скрипт Obstacle

Даний скрипт реалізує взаємодію перешкоди та гравця у 2D-грі на Unity. Перше, що робить скрипт – знаходить гравця на сцені Unity за допомогою методу GameObject.FindGameObjectWithTag("Player") і зберігає його у змінну player. Потім скрипт містить метод OnTriggerEnter2D(Collider2D collision), який викликається, коли об'єкт із цим скриптом стикається з іншим колайдером у грі. Залежно від тега об'єкта, з яким сталося зіткнення, скрипт або видаляє сам об'єкт, або видаляє гравця. Якщо об'єкт стикається з колайдером з тегом Border, то об'єкт, на якому даний скрипт знаходиться, буде видалений з гри за допомогою методу Destroy(this.gameObject). Якщо об'єкт стикається з колайдером із тегом "Player", то гравець, збережений у змінній player, буде видалено за допомогою методу Destroy(player.gameObject). Ми використовуємо метод Destroy для видалення об'єктів. Передаючи this.gameObject, ми видаляємо перешкоду, яку вплинула колізія, а передаючи player.gameObject, ми видаляємо гравця[5]. Таким чином, цей скрипт реалізує функціонал видалення об'єкта під час зіткнення з межею ігрового поля чи гравцем. Також створюється новий скрипт “Game Over” (Рис. 3.20) , в якому буде реалізовано програш.

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

public class GameOver : MonoBehaviour
{
    public GameObject gameOverPanel;

    // Update is called once per frame
    void Update()
    {
        if(GameObject.FindGameObjectWithTag("Player") == null)
        {
            gameOverPanel.SetActive(true);
        }
    }

    public void Restart()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.24 – Новий скрипт Game Over

Цей скрипт у Unity відповідає за відображення панелі програшу та перезапуск сцени після ураження гравця. Починається він з оголошення змінної "gameOverPanel" типу GameObject, яка використовується для відображення панелі програшу. Далі, у функції "Update()" скрипт перевіряє наявність об'єкта з тегом "Player" на сцені. Якщо такого об'єкта немає, значить гравець програв, і панель програшу має бути активована методом SetActive (true). Функція "Restart()" використовується для перезапуску поточної сцени, натиснувши кнопку "Restart". Метод Restart() викликається, коли гравець натискає кнопку перезапуску на панелі gameOverPanel. Завантажується поточна сцена занову, використовуючи метод LoadScene(), із класу SceneManager[5].

Таким чином рівень перезавантажується і гравець може почати гру заново. Наприклад, якщо ми маємо сцену "Level 1", і ми знаходимося на цій сцені, то метод GetActiveScene().name повертає рядок "Level 1". Скрипт передає цей рядок метод LoadScene(), який завантажує цю ж сцену заново, починаючи з її початку. Таким чином, скрипт GameOver відстежує кінець гри та дозволяє гравцеві перезапустити рівень за допомогою однієї кнопки.

Вона завантажує поточну сцену за допомого методу Scene Manager.LoadScene(SceneManager.GetActiveScene().name.

В цілому, цей скрипт є простим, але ефективним способом обробки подій програшу у грі та дозволяє гравцеві швидко перезапустити гру після ураження. Створений скрипт потрібно помістити на головний елемент гри “Game Manager”, і помістити в створену комірку покласти, створений раніше елемент “Game Over Panel” (Рис. 3.21).

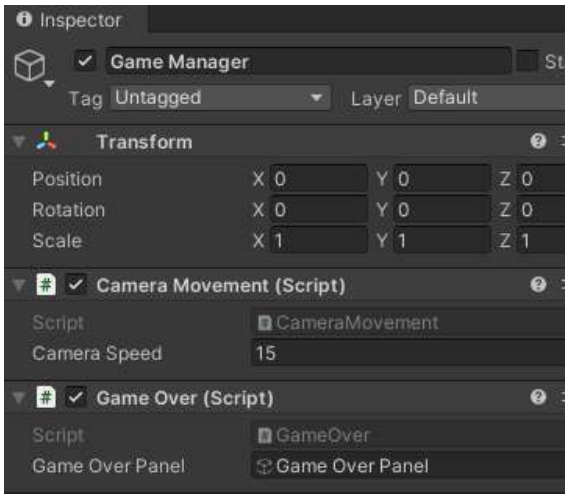


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.25 – Налаштування панелі

У панелі налаштувань “Restart Button”, у розділі “OnClick” помістити в порожню комірку “Game Manager”, та додати функцію “GameOver.Restart”. Функція GameOver.Restart – це функція користувача, яка викликається при завершенні гри (Game Over) і дозволяє перезапустити гру. Зазвичай, ця функція викликається при натисканні на кнопку "Restart" або подібну до неї, яка з'являється на екрані після закінчення гри (Рис. 3.22). При натисканні на цю кнопку, функція GameOver.Restart перезапускатиме гру заново з початку.

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.26 – Фінальний вигляд панелі

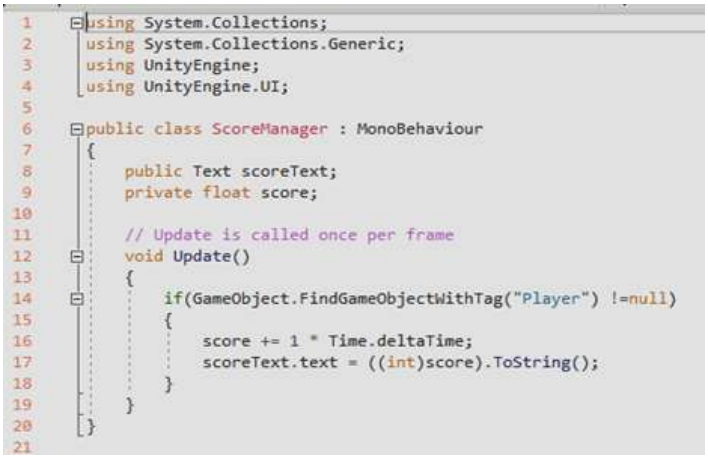
А в елементі “Player”, змінити тег з “Untagged” на “Player”. Для того, щоб скрипт та його функції повністю працювали [7].

3.1.6 Система набору очок і внутрішньоігрові ефекти

Система набору очок в іграх Ранерах – це спосіб відстеження успіхів гравця у грі та надання йому можливості порівняти свої результати з результатами інших гравців. Зазвичай очки нараховуються за певні дії, наприклад, за подолання перешкод, збирання монет та інших бонусів, виконання завдань та досягнень у грі. За кожну виконану дію гравець отримує певну кількість очок, які накопичуються у його загальному рахунку. Крім того, в іграх Ранер часто використовується система множників очок. Це означає, що при виконанні кількох дій поспіль гравець отримує все більше очок за кожну наступну дію, що дозволяє йому швидше набирати очки. Набрані окуляри можуть бути використані для різних цілей, наприклад, для покупки нових персонажів, покращення здібностей персонажа або відкриття нових рівнів. Система набору очок в іграх Ранерах є одним із способів мотивації гравців та спонукання їх до подальшого покращення своїх результатів.

Для того, щоб реалізувати рахунок у грі, потрібно додати текстовий елемент під назвою “Score Text”. Потрібно підібрати висоту знаходження

тексту і вибрати колір, щоб було добре його видно на задньому тлі. Найоптимальнішим знаходженням очок це по центру екрану, а колір був вибраний зелений. Також потрібно створити новий скрипт під назвою “Score Manager”. Даний скрипт "ScoreManager" (Рис. 3.23) відповідає за відображення поточного рахунку гравця у реальному часі у грі на Unity. Він використовується для того, щоб гравець міг бачити свій рахунок, а також відстежувати свій прогрес [9].



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

public class ScoreManager : MonoBehaviour
{
    public Text scoreText;
    private float score;

    // Update is called once per frame
    void Update()
    {
        if(GameObject.FindGameObjectWithTag("Player") !=null)
        {
            score += 1 * Time.deltaTime;
            scoreText.text = ((int)score).ToString();
        }
    }
}
```

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.27 – Новий скрипт для рахунку очок

Спочатку ми оголошуємо дві змінні: "scoreText" та "score". Змінна "scoreText" використовується для збереження посилання на текстовий елемент UI, який відображатиме поточний рахунок гравця на екрані. Змінна "score" використовується для збереження поточного значення рахунку. У методі Update() скрипта відбувається оновлення значення score та його відображення у scoreText. Якщо гравець існує (його об'єкт можна знайти за допомогою тега "Player"), то значення score збільшується на 1 за кожен кадр, помножений на Time.deltaTime – це дозволяє зробити ігровий процес більш плавним та незалежним від швидкості оновлення кадрів. Потім scoreText оновлюється за допомогою методу text, який приймає як аргумент рядок, який відображатиметься на екрані. В даному випадку використовується

((int)score).ToString(), щоб привести значення score до цілого числа і перевести його в рядок [6].

Далі, у методі "Update()" перевіряється, чи є гравець на сцені за допомогою методу "GameObject.FindGameObjectWithTag("Player")". Якщо гравець є на сцені, то до поточного значення рахунку "score" додається 1 помножений на "Time.deltaTime". "Time.deltaTime" – це час, що минув між поточним та попереднім кадрами, і використовується для того, щоб рахунок не збільшувався надто швидко. Потім поточне значення рахунку перетворюється на ціле число і надається змінною "scoreText.text", яка відображає поточний рахунок на екрані. Таким чином, при кожному кадрі, коли гравець знаходиться на сцені, його рахунок збільшуватиметься на одиницю за певний час і відображатиметься на екрані у вигляді тексту. Це дозволяє гравцеві відстежувати свій поточний прогрес у грі.

Цей скрипт потрібно помістити в елемент “Score Text”, і так само створений елемент тексту, який веде рахунок, помістити в комірку у скрипті (Рис. 3.24).

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.28 – Вигляд скрипта на об'єкті

Щоб зробити гарну гру можна додати деякі ефекти. Ефекти в Unity це візуальні та звукові елементи, які додають додаткові емоційні та візуальні аспекти до ігор та інших візуальних програм. Ефекти можуть бути використані для створення реалістичних атмосфер, покращення візуальної привабливості, виділення певних об'єктів та подій, а також для привернення уваги користувача. У Unity існує безліч різних ефектів, які можуть бути додані до ігрових об'єктів [7].

Деякі з найпоширеніших ефектів включають:

– Particle System – дозволяє створювати та анімувати частинки, такі як дим, вогонь, сніг, дощ тощо;

– Lighting – освітлення, включаючи точкові та спрямовані джерела світла, а також плавне освітлення та глобальну ілюмінацію;

– Camera Effects – дозволяють створювати різні ефекти для камери, такі як розмиття, спотворення, корекцію кольорів та інші візуальні ефекти;

– Audio Effects – дозволяють створювати різні звукові ефекти, такі як луна, реверберація, згасання та інші;

– UI Effects – дозволяють створювати різні візуальні ефекти для інтерфейсу користувача, такі як анімації, плавна поява і зникнення і т.д.;

– Physics Effects – дозволяють створювати ефекти фізичного на об'єкти, такі як вибухи, коливання та інші.

У Unity також є можливість створення власних ефектів за допомогою скриптування та використання сторонніх пакетів, які розширюють можливості ефектів. Перше, що варто додати цей музичний супровід в ігри. Музика в іграх – це важливий елемент гри. Музика в іграх відіграє важливу роль, оскільки вона створює атмосферу, посилює емоційний ефект та допомагає підтримувати гравця у зоні комфорту. Музичні ефекти можуть бути використані для підкреслення певних моментів у грі. Наприклад, звук, який програється, коли гравець проходить рівень, може створити почуття задоволення та досягнення мети. Звук, який програється під час поразки або програшу, може викликати почуття розчарування та надихнути гравця зробити ще одну спробу. Музика також може використовуватись для створення певного настрою у грі. Наприклад, похмура та зловісна музика може допомогти створити атмосферу страху та напруги, тоді як легка та мелодійна музика може створити більш розслаблюючу та комфортну атмосферу. Крім того, музика в іграх може допомогти створити сильний зв'язок між гравцем та грою. Відповідна музика може викликати у гравця емоційну реакцію, що допомагає утримати його у грі та змушує гравця грати

довше. В цілому, музика є важливим елементом в іграх, оскільки вона допомагає створювати певну атмосферу та емоційний зв'язок між гравцем та грою, що сприяє більш повному зануренню в ігровий світ[1].

Щоб додати музику до гри, потрібно створити черговий скрипт під назвою “Background Music” (Рис. 3.25). Даний скрипт реалізує музичний фон для гри, яке зберігається протягом усієї гри, навіть якщо сцени перемикаються.

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.29 – Скрипт додавання музики

Для початку визначимося зі змінними та типами даних.

private static BackgroundMusic backgroundMusic – це статична змінна типу BackgroundMusic, яка зберігає посилання на об'єкт музичного фону. Модифікатор static гарантує, що змінна буде доступна всім екземплярів класу і існує лише у єдиному екземплярі.

void Awake() – це метод, який викликається на момент активації об'єкта. У цьому випадку він використовується для перевірки наявності об'єкта музичного фону на сцені та його збереження у разі його відсутності.

if(backgroundMusic == null) – перевірка, чи існує об'єкт музичного фону на сцені.

backgroundMusic = this; – якщо об'єкт музичного фону відсутній, зберігаємо посилання на поточний об'єкт музичного фону.

DontDestroyOnLoad(backgroundMusic); – Зберігаємо об'єкт музичного фону протягом усієї гри, не знищуючи його при перемиканні сцен.

else – якщо об'єкт музичного фону вже існує на сцені, поточний об'єкт музичного фону знищується, щоб не створювати дублікати.

Destroy(gameObject); – Знищення поточного об'єкта музичного фону.

Таким чином, цей скрипт забезпечує постійний фоновий звук у грі, який продовжує грати упродовж усіх сцен без переривань та дублювання. Створюється новий елемент гри “Background Music” (Рис. 3.26), і в нього розміщуємо створений раніше скрипт.

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.30 – Фінальна версія музичного скрипта

Щоб додати пісню, регулювати її гучність та ефекти для неї, потрібно додати компонент “Audio Source”. Audio Source – це компонент Unity, який дозволяє програвати звукові ефекти, музику та мовлення у вашій грі. Він може бути доданий до будь–якого об'єкта в сцені та налаштований для відтворення звукових файлів у 2D та 3D просторі. Audio Source має багато параметрів, які можуть бути налаштовані для контролю над звуком, включаючи:

– AudioClip: звуковий файл, який програватиметься;

– Volume: гучність звуку від 0 до 1;

– Pitch: висота звуку від –3 до 3;

– Loop: повторення звуку після відтворення;

– Spatial Blend: налаштування простору звуку в діапазоні від 0 (2D) до 1 (3D).

Min Distance та Max Distance: відстань, при якій звук буде найбільш гучним і відстань, при якій звук стає тихішим.

Audio Source може бути запущений програмно за допомогою скрипту, що робить його корисним для створення аудіоефектів, музичних тем, голосових ефектів та багато іншого. Він також може бути налаштований на відтворення звуку під час зіткнення з іншим об'єктом у сцені, що може бути використане для створення звукових ефектів зіткнень, пострілів та багато іншого. І вже потім можна додати до гри свої пісні чи звичайні мелодії.

Також варто додати візуальні ефекти. Візуальні ефекти в Unity – це графічні ефекти, які використовуються для покращення візуального сприйняття гри. Ці ефекти можуть створювати різні візуальні зміни, такі як зміна кольору, зміна текстури, зміна форми, рух об'єктів та багато інших. Unity надає безліч візуальних ефектів, які можна використовувати для створення гарної та цікавої ігрової графіки [7].

Деякі з цих ефектів включають:

– частинки: це ефект, який створює різні вибухи, дим, вогонь та інші елементи, використовуючи безліч дрібних частинок, які рухаються в заданому напрямку;

– об'ємне світло: це ефект, що створює враження об'ємності об'єктів, змінюючи колір світла в залежності від кута огляду гравця;

– постобробка: це ефект, який застосовується до остаточної картинки та дозволяє змінювати її колірний баланс, контраст, яскравість та багато іншого;

– рельєфний ефект: це ефект, що створює враження наявності рельєфу на об'єктах. Цей ефект можна використовувати для створення ілюзії тривимірного простору на 2D–ігровому полі;

– зернистість: це ефект, який створює зернистість на екрані, щоб створити враження фільму або старого фільму;

– глибина різкості: це ефект, який створює враження глибини та розмитості на екрані.

Всі ці візуальні ефекти в Unity можуть бути використані для створення гарної картинки [9].

По-перше потрібно створити не порожній елемент гри, а відразу вибрати розділ “Effects”, а в ньому “Particle System”. І вже в ньому налаштовувати власні ефекти за своїми бажаннями. У розділі “Render”, у вкладці “Materials’, у цій грі обраний вид “Sprite Default” (Рис. 3.27). Sprite Default в Unity – це налаштування для спрайту, яке визначає його поведінку та властивості, такі як розмір, колір, прозорість тощо. У налаштуваннях Sprite Default можна змінити стандартний спрайт для всіх нових спрайтів, що створюються в проєкті. Їх великий плюс у тому, що вони не сильно навантажують комп’ютера.

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.31 – Меню вибору матеріалу

У розділі “Texture Sheet Animation”, в першій вкладці обрати “Sprite”. Texture Sheet Animation – це зручний спосіб створення анімації з одного

зображення, та його використання може значно скоротити кількість ресурсів, необхідних для створення анімації [8]. У наступній комірці обрати “Particle” (Рис. 3.28).

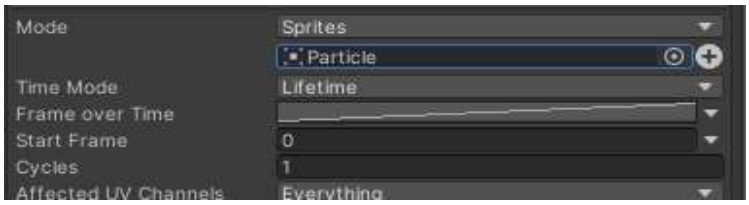


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.32 – Налаштування анімації

Ще потрібно налаштувати колір кульок, що вилітають, так як їх два види (Рис. 3.29), і що б це виглядало красивіше потрібно їх зробити різними. І ще визначитися з їх розміром, щоб вони не займали весь екран.

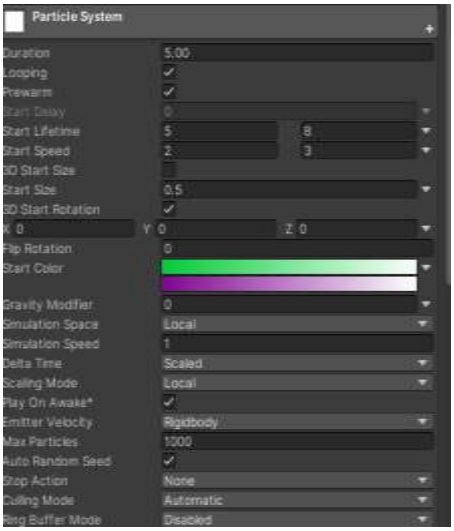


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.33 – Налаштування візуальних ефектів

Це і було фінальне налаштування гри в стилі Ранер, і вона готова до публікації в інтернеті (Рис. 3.30).

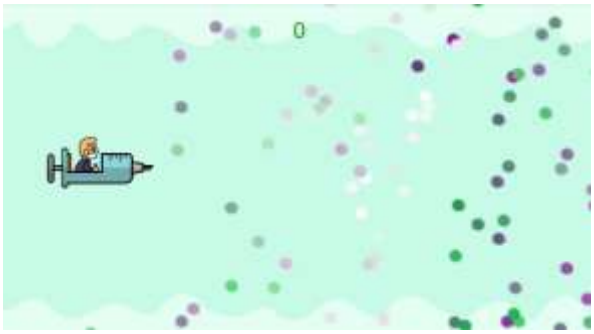


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.34 – Фінальній вигляд гри

ВИСНОВКИ

У кваліфікаційній роботі було розроблено сценарій та реалізовано гру у стилі “Ранер” на Unity. У процесі розробки були розглянуті основні концепції та інструменти, що використовуються під час створення ігор на Unity, такі як створення ігрового об'єкта, додавання компонентів, налаштування фізики, використання анімацій, звукових ефектів, ефектів візуалізації, використання скриптів на мові C# та багато інших.

Для розробки гри була створена 2D сцена з використанням спрайтів та колайдерів, налаштовані фізичні властивості об'єктів, додано скрипти мовою C#, обробні дії користувача та автоматичні дії об'єктів, а також було додано звукові та візуальні ефекти для покращення ігрового середовища. Проведено тестування та налагодження проєкту.

Розглянуто деякі концепції, що використовуються в іграх Ранер, такі як генерація рівнів, набір очок. Створення гри Ранер на Unity – це цікавий і захоплюючий процес, який дозволяє отримати досвід у роботі з ігровими концепціями та інструментами, а також навчитися використовувати різні аспекти розробки ігор, такі як анімації, звукові та візуальні ефекти, управління ігровими об'єктами та багато інші.

ПЕРЕЛІК ПОСИЛАНЬ

1. Костер Р. Теорія розваг для ігрового дизайну. Київ : ArtHuss, 2022. 288 с.

2. Меннінг Д., Батфілд-Еддісон П. Unity для розробника. Мобільні мультиплатформні ігри. Львів : Піраміда, 2018. 304 с.

3. Офіційний форум від Unity для розробника ігор. URL : https://forum.unity.com/ (дата звернення 18.02.2023).

4. Ресурс IT-навчання. URL : https://itproger.com/ua/course/unity (дата звернення 15.03.2023).

5. Харрисон Ф. Вивчаємо C# через розробку ігор Unity. Київ : Віват, Т.5. 2022. 400 с.

6. Albahari J., Albahari B. C# 10 Pocket Reference: Instant Help for C# 10 Programmers. 1st Ed. California: O'Reilly Media, 2022. 270 p.

7. Bond J. G. Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C#. Boston : Addison-Wesley Professional, 2017. 1024 p.

8. Joe Hocking. Unity in Action: Multiplatform Game Development in C#. New York : Manning Publications, 2015. 352 p.

9. А. Ковальов. Unity. Розробка ігор для мобільних пристроїв. Київ : English Student, 2021. 496 с.

10. Paris Buttfield-Addison, Jon Manning, Tim Nugent. Unity Game Development Cookbook. Sebastopol: O'Reilly Media, 2019. 405 p.