МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

# Кваліфікаційна робота бакалавра

на тему: Розробка десктопного ігрового 2D застосунку з використанням Unity game engine

Виконав студент групи **К-19б**
спеціальності 122 Комп'ютерні науки
**Трюхан Іван Олегович**

Керівник **ст. викладач**
**Вохменцева Т.Б.**

Консультант **д.ф., доцент**
**Бучинська І.В.**

Рецензент **к.ф.-м.н., професор**
**Ковальчук В.В.**

Одеса 2023

**ЗМІСТ**

# ПЕРЕЛІК СКОРОЧЕНЬ,УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Геймплей − це загальний термін, що використовується для опису взаємодії гравця з комп'ютерною грою.

Ігровий движок − це комплексне програмне забезпечення, яке використовується для розробки і створення комп'ютерних ігор.

Ігровий застосунок − програмний продукт, створений для гри.

Паттерн − це повторно використовуваний рекомендований підхід або шаблон проектування.

Плагін − це програмний модуль або розширення, яке додає додаткову функціональність або можливості до існуючої програми.

Платформер − це жанр комп'ютерних ігор, в яких головним елементом геймплею є переміщення головного героя по платформам.

Asset − ресурс, такий як модель, текстура, звук або анімація, використовується в грі.

Collider − компонент Unity, що визначає область зіткнення об'єкта.

Component − компонент, будь-яка частина функціональності об'єкта в Unity.

Coroutine − корутина, асинхронна функція, що дозволяє виконувати код із затримкою.

GameObject − ігровий об'єкт, основна одиниця конструкції в Unity.

Prefab − префабрикат, шаблон гри, що містить в собі компоненти та властивості об'єкта.

Rigidbody − компонент Unity, що відповідає за фізичну поведінку об'єкта.

Script − сценарій, програмний код для керування поведінкою об'єктів.

2D − двовимірний.

Unity − Unity game engine.

UI − інтерфейс користувача (User Interface).

# ВСТУП

У сучасному світі відеоігри стали невід'ємною частиною культури та розваг. Вони пропонують можливість поринути в захоплюючі віртуальні світи, випробувати пригоди, подолати складнощі та насолодитися неповторною взаємодією з віртуальними персонажами та оточенням. Одним із популярних жанрів відеоігр є 2D платформери. Платформери − це жанр відеоігор, в яких гравець керує персонажем, рухаючись по платформах, долаючи перешкоди та борючись з ворогами. Під час виконання кваліфікаційної роботи бакалавра було досліджено створення та розробку 2D платформерів з використанням ігрового движка Unity 2D.

Метою кваліфікаційної роботи бакалавра є розробка стабільного десктопного 2D застосунку, який буде працювати на пристроях з обмеженими ресурсами, а також створення інтуїтивно зрозумілого та зручного для користувача інтерфейсу.

У сучасному світі 2D платформери не втратили своєї популярності. Вони живі та активно розвиваються, приваблюючи як нових гравців, так і прихильників старих класичних ігор. З розвитком технологій та появою нових ігральних платформ 2D платформери отримали нові можливості в графіці, фізиці, штучному інтелекті та багато іншому.

Сьогодні розробники ігор продовжують вабити новими та захоплюючими 2D платформерами. Вони пропонують різноманітні механіки та особливості геймплею, такі як паралакс ефекти, різні стрибки та рухи героя, складні головоломки, боси та секрети на рівнях. Деякі ігри комбінують 2D та 3D елементи, створюючи унікальні та вражаючі візуальні ефекти. Тому, 2D платформери продовжують приваблювати гравців своєю унікальною атмосферою та захоплюючим геймплеєм.

Дана кваліфікаційна робота бакалавра, складається з 62 сторінок, та 23 рисунків.

# 1 АНАЛІЗ ІНФОРМАЦІЇ ПРО СТВОРЕННЯ ПЕРШИХ ВІДЕОІГР

Перші відеоігри з'явилися у далеких 1950-х та 1960-х роках, коли комп'ютери тільки починали набирати популярність. Прабатьками відеоігор вважаються експерименти та розробки дослідників та вчених, які прагнули створити інтерактивний розваги на основі електронних обчислювальних машин. Одною з перших ігор, яка отримала широку популярність, була "Pong" − проста симуляція тенісу, розроблена у 1972 році.

З плином часу та розвитком технологій відеоігри стали все більш складними та захоплюючими. У 1980−х роках були створені перші 2D платформери, які стали символами своїх грального ігрових консолей та здобули широку популярність. Наприклад, "Super Mario Bros." від Nintendo та "Sonic the Hedgehog" від SEGA стали іконами жанру 2D платформерів.

"Super Mario Bros." (1985) запропонував гравцям унікальні механіки, які стали стандартом для більшості платформерів. У грі можна було стрибати на ворогів, збирати монети, досліджувати різноманітні рівні з секретами та бонусами. Це був перший платформер, в якому було реалізовано такий рівень деталізації та майстерність дизайну рівнів, що вони стали прикладом для багатьох наступних ігор.

"Sonic the Hedgehog" (1991) вніс свій внесок у жанр 2D платформерів, запропонувавши гравцям більш швидкий та динамічний геймплей. Головним героєм гри був їжак Сонік, здатний розвивати величезну швидкість та подолати рівні з неймовірною швидкістю. Ця гра показала, що в 2D платформерах можна грати з високою швидкістю та відчувати адреналін.

# 2 ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ ЗАСОБІВ РОЗРОБКИ ДЛЯ СТВОРЕННЯ ДЕСКТОПНИХ ІГРОВИХ ЗАСТОСУНКІВ

## 2.1 Порівняльних аналіз існуючих ігрових движків

Для розробки гри головним інструментом вважається ігровий движок − програмне забезпечення для створення комп'ютерних ігор. Для порівняльного аналізу вибрано найпопулярніші ігрові движки сьогодення, а саме: Unreal Engine, Unity, Game Maker: Studio, Godot, CryEngine.

### 2.1.1 Опис ігрового движка CryEngine

CryEngine є потужним та популярним ігровим движком, розробленим компанією Crytek. Він широко застосовується в індустрії розробки відеоігор та надає розробникам безліч інструментів та можливостей для створення проєктів високої якості.[3]

Основні характеристики CryEngine:

− висока якість графіки (CryEngine відомий своєю вражаючою графікою та можливостями рендерингу);

− фізика та анімація (движок має потужну систему фізичного моделювання, яка дозволяє створювати реалістичні фізичні ефекти, такі як вибухи, колізії та симуляція рідин. Він також надає розширену систему анімації персонажів);

− інструменти редактора (CryEngine надає розробникам широкий набір інструментів редактора для створення ігрових світів, рівнів, скриптів та налаштування ігрових параметрів);

− багатоплатформність (CryEngine підтримує різні платформи, включаючи ПК, консолі та мобільні пристрої);

− система компонентів (CryEngine використовує модульну систему компонентів, яка спрощує розробку та впровадження нових

функціональностей).

До переваг ігрового движка, можна віднести:

− візуальна якість (CryEngine відомий своєю вражаючою графікою та можливістю створювати вражаючі візуальні ефекти);

− потужні інструменти (редактор CryEngine надає широкий спектр інструментів та ресурсів для розробки ігор);

− спільнота розробників (CryEngine має активну спільноту розробників, яка надає підтримку, посібники, навчальні матеріали та ресурси для розробки на цьому движку).

Недоліки ігрового движка:

− високі вимоги до системних ресурсів (розробка на CryEngine вимагає потужного обладнання та високої продуктивності комп'ютера. Це може бути проблемою для невеликих команд або індивідуальних розробників з обмеженими ресурсами);

− складність вивчення (CryEngine потребує більшого рівня освоєння порівняно з деякими іншими движками. Він вимагає певного рівня знань та досвіду, щоб використовувати його ефективно);

CryEngine є потужним ігровим движком з гарною графікою та широким набором інструментів.[4] Він підходить для створення високоякісних і візуально привабливих ігрових проєктів. Однак вивчення та робота з ним можуть бути складними задачами, а також вимагають високої продуктивності комп'ютера. Якщо розробник має достатні ресурси та досвід, CryEngine може бути відмінним вибором для створення ігрових проєктів. [3]

### 2.1.2 Опис ігрового движка Godot

Godot − це безкоштовний та багатоплатформовий движок з відкритим вихідним кодом для розробки ігор та інтерактивного контенту. Він надає розробникам потужні інструменти та гнучку середовище для створення ігор різних жанрів та форматів.

Основні можливості ігрового движка Godot:

- багатоплатформовість (Godot підтримує розробку ігор для різних платформ, включаючи Windows, macOS, Linux, Android, iOS, та інші);
- зручний редактор (вбудований редактор Godot має інтуїтивно зрозумілий інтерфейс, що спрощує створення та редагування ігрових ресурсів, сцен та скриптів);
- сценарій на основі вузлів (розробка в Godot здійснюється шляхом створення та налаштування вузлів у сценах. Це дозволяє легко структурувати ігрові об'єкти та їх взаємодію);
- потужна система анімації (Godot пропонує гнучку систему анімації, яка дозволяє створювати складні анімаційні ефекти та керувати ними за допомогою часових ліній та станів);
- підтримка різних мов програмування (Godot підтримує кілька мов програмування, включаючи GodotScript (схожий на Python) та C#. Це дає змогу розробникам вибрати мову для реалізації гри);

Переваги ігрового движка Godot:

- безкоштовність та відкритий вихідний код (Godot доступний безкоштовно та має активну спільноту розробників, що сприяє обміну знаннями та підтримці);
- потужні інструменти (Godot пропонує широкий набір інструментів для створення ігор, включаючи підтримку фізики, звуку, графіки, штучного інтелекту та багато іншого);
- гнучкість та розширюваність (Godot дозволяє розробникам створювати власні розширення та модулі, що сприяє інтеграції сторонніх бібліотек та функціональності);
- активна спільнота (у Godot є велика та активна спільнота розробників, де можна отримати підтримку, обмінятися досвідом та знайти готові рішення для різних завдань).

Недоліки ігрового движка Godot:

- відносна новизна (у порівнянні з деякими іншими движками, Godot є відносно новим на ринку розробки ігор, що може означати обмежену кількість готових рішень та документації);
- обмежені ресурси (незважаючи на активну спільноту, Godot може мати обмежений вибір готових ресурсів, таких як моделі, текстури та звукові ефекти);
- відсутність підтримки деяких платформ (на відміну від деяких комерційних движків, Godot може не підтримувати деякі менш популярні платформи або мати обмежену функціональність для них).

Godot є потужним і гнучким багатоплатформовим движком з відкритим вихідним кодом, який надає розробникам широкі можливості для створення ігор та інтерактивного контенту[4]. Безкоштовність, активна спільнота та потужні інструменти роблять Godot привабливим вибором для розробників. Однак варто враховувати його відносну новизну на ринку та можливі обмеження в виборі готових ресурсів та підтримці деяких платформ. В цілому, Godot надає розробникам засіб для творчої реалізації своїх ідей та створення високоякісних ігрових проєктів[3].

### 2.1.3 Опис ігрового движка GameMaker Studio

GameMaker Studio є популярним інтегрованим середовищем розробки ігор, яке надає розробникам потужні інструменти для створення ігор різних жанрів і форматів. Він має простий у використанні редактор, який дозволяє розробникам створювати графічні ресурси, анімації, рівні та логіку гри без необхідності програмування.

До основних особливостей GameMaker Studio, можна віднести:

- мова програмування GML (GameMaker Studio використовує мову програмування GML (GameMaker Language), яка є простою для вивчення та дозволяє розробникам створювати складну логіку гри.

GML має синтаксис, подібний до C, що полегшує перехід для розробників з досвідом у програмуванні);

- візуальний редактор (GameMaker Studio має інтуїтивно зрозумілий візуальний редактор, що дозволяє легко створювати та налаштовувати об'єкти, персонажі, рівні та інші елементи гри шляхом перетягування та розміщення);
- вбудовані ресурси (редактор GameMaker Studio має багато вбудованих ресурсів, таких як зображення, звуки, шрифти та інші, що полегшує розробку ігор без необхідності в пошуку або створенні власних ресурсів);
- підтримка різних платформ (GameMaker Studio дозволяє розробникам створювати ігри для різних платформ, включаючи Windows, macOS, Linux, Android, iOS та інші);
- фізика та колізії (движок має вбудовану підтримку фізики та колізій, що дозволяє розробникам легко налаштовувати рух об'єктів та взаємодію між ними).

Плюси GameMaker Studio:

- легкий для вивчення (GameMaker Studio має дружній інтерфейс та просту мову програмування GML, що дозволяє навчитися розробці ігор навіть початківцям);
- велике співтовариство (GameMaker Studio має активне співтовариство розробників, яке надає підтримку, допомогу та можливість обміну знаннями);
- розширені можливості (GameMaker Studio надає розробникам широкий спектр можливостей, включаючи підтримку фізики, анімації, звуків, мережевої гри та багато іншого).

Мінуси GameMaker Studio:

- обмежена гнучкість (в порівнянні з іншими движками, GameMaker Studio може мати обмежену гнучкість у створенні складних ігрових механік або спеціалізованих ефектів);

- висока ціна ліцензії (незважаючи на наявність безкоштовної версії, повна ліцензія GameMaker Studio може бути високою за вартістю, особливо для незалежних розробників);
- обмеження в швидкості (для деяких типів ігор, особливо з великою кількістю об'єктів або складними обчисленнями, GameMaker Studio може виявитися менш продуктивним порівняно з іншими двигунами).

GameMaker Studio є потужним інструментом для створення ігор, особливо для початківців і незалежних розробників. Він має простий у використанні інтерфейс та мову програмування, що дозволяє швидко створювати ігрові проєкти. Активне співтовариство та наявність ресурсів роблять його привабливим вибором для навчання та розробки. Однак, він може бути обмежений у складних механіках та вимагати високої ціни для повної ліцензії. В цілому, GameMaker Studio є вартою уваги платформою для створення ігор, здатною задовольнити потреби багатьох розробників[4].

### 2.1.4 Опис ігрового движка Unity game engine

Unity − це один з найпопулярніших ігрових движків, який використовується для розробки ігор різного жанру та формату[3]. Його потужність і гнучкість дозволяють розробникам створювати як 2D, так і 3D ігри з високоякісною графікою і реалістичною фізикою[1]. Основні характеристики та компоненти Unity:

- мультиплатформенність (Unity підтримує розробку ігор для різних платформ, включаючи ПК, консолі, мобільні пристрої, віртуальну реальність та інші. Це дозволяє розробникам досліджувати широкий спектр ринків і досягати більшої аудиторії);
- високоякісна графіка (Unity має потужний графічний рушій, який дозволяє створювати вражаючі візуальні ефекти і деталізовану графіку. Засоби освітлення, тіні, текстури та інші графічні ефекти

допомагають розробникам створювати привабливі візуальні світи);

− фізична система (Unity має вбудовану фізичну систему, що дозволяє моделювати реалістичну фізику у грі. Об'єкти можуть взаємодіяти між собою, реагувати на сили та симулювати рух і колізії. Це дозволяє створювати ігри з реалістичною поведінкою об'єктів);

− система скриптів (Unity використовує мову програмування C# для створення скриптів ігрової логіки. Це дозволяє розробникам контролювати поведінку об'єктів, створювати інтерактивність та реалізовувати складну логіку гри);

− редактор Unity (Unity має потужний інтуїтивно зрозумілий редактор, який дозволяє візуально створювати, редагувати та налаштовувати елементи гри);

− спільнота розробників та екосистема (Unity має велику та активну спільноту розробників, де можна знайти підтримку, документацію, уроки та поради).

Плюси використання ігрового движка Unity:

− мультиплатформність, що дозволяє розробникам створювати ігри для різних платформ;

− високоякісна графіка і потужність графічного рушія;

− реалістична фізика та колізії;

− можливість програмування скриптів на мові C#;

− інтуїтивно зрозумілий редактор;

− велика спільнота розробників та розширень.

Мінуси використання ігрового движка Unity:

− на великих проєктах можуть виникати проблеми з продуктивністю і оптимізацією;

− великий розмір вихідних файлів гри;

− вимагає певного рівня вивчення та досвіду для використання всіх функцій і можливостей.

Unity − це потужний та розширюваний ігровий движок, який підходить для розробки ігор різного жанру та формату[4]. Він має велику кількість функціональних можливостей, що дозволяють реалізувати ідеї розробників і створювати вражаючі ігрові проєкти[3]. Завдяки мультиплатформності, великій спільноті та екосистемі, Unity є популярним вибором для багатьох розробників[1].

### 2.1.5 Опис ігрового движка Unreal Engine

Unreal Engine – це потужний ігровий движком, розробленим компанією Epic Games. Він використовується для створення вражаючих ігрових проєктів різних жанрів та форматів.

Основні характеристики Unreal engine:

− Unreal Engine відомий своїм вражаючим рівнем візуальної якості;

− розширюваність та гнучкість – Unreal Engine надає широкі можливості для розширення та налаштування. ;

− Unreal Engine підтримує розробку ігор для різних платформ, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність;

− Unreal Engine надає два основних способи програмування − Blueprints та C++ (Blueprints − це візуальна система скриптування, яка дозволяє створювати логіку гри без необхідності програмування на мові C++. Unreal Engine також підтримує програмування на C++, що дозволяє розробникам мати більшу гнучкість та контроль над своїми проєктами);

− Unreal Engine має потужний редактор, який дозволяє візуально створювати, редагувати та налаштовувати різні аспекти гри, такі як сцени, персонажі, анімація, фізика та багато іншого;

− Unreal Engine має велику та активну спільноту розробників.

Плюси використання Unreal Engine:

− вражаюча візуальна якість і графічний движок;

− розширюваність та гнучкість;

− підтримка багатьох платформ;

− візуальна система програмування Blueprints;

− потужний редактор Unreal;

− велика спільнота розробників та підтримка.

Мінуси використання Unreal Engine:

− вимогливість до апаратного забезпечення;

− складний процес навчання та розуміння системи;

− великий розмір вихідних файлів гри.

Unreal Engine − це потужний ігровий движок, який надає розробникам широкі можливості для створення вражаючих ігрових проєктів. Він володіє вражаючою візуальною якістю, розширюваністю та гнучкістю, підтримкою багатьох платформ та потужним редактором. Однак, він вимагає достатньо потужного обладнання, може мати процес навчання та розуміння, а також має великі розміри вихідних файлів гри. З урахуванням його можливостей та обмежень, Unreal Engine є привабливим вибором для серйозних ігрових проєктів, якщо розробники готові вкласти час та зусилля в освоєння цього потужного інструменту.

## 2.2 Порівняльний аналіз мов програмування

### 2.2.1 Мова програмування C#

C# – це об'єктно-орієнтована мова програмування. Вона дозволяє розробникам створювати різні типи безпечних та надійних програм, що працюють у .NET[5]. Ця мова належить до добре відомого сімейства мов C і буде знайома кожному, хто працював з C, C++, Java або JavaScript[2].

Переваги мови C#:

− підтримка більшості продуктів Microsoft;

− безкоштовність ряду інструментів для невеликих компаній та деяких індивідуальних розробників: VisualStudio, Azure, WindowsServer,

ParallelsDesktop для MacPro та ін;

− типи даних мають фіксований розмір (32−бітний int та 64−бітний long), що підвищує «мобільність» мови та спрощує програмування;

− автоматичний «збір сміття», це означає, що у більшої кількості випадків не доведеться дбати про звільнення пам'яті;

− велика кількість «синтаксичного «цукору» − спеціальних конструкцій, розроблених для розуміння та написання коду;

− низький поріг входження, синтаксис C# має багато схожого з іншими мовами програмування, завдяки чому полегшується перехід програмістів;

− за допомогою Xamarin на C# можна писати програми для таких операційних систем, як iOS, Android, MacOS та Linux.

Недоліки мови C#:

− орієнтованість на платформу Windows;

− мова безкоштовна лише для невеликих фірм, індивідуальних програмістів, стартапів тощо.

### 2.2.2 Мова програмування Java

Java, строго типізована об'єктно−орієнтована мова програмування загального призначення, розроблена Sun Microsystems. Розробка здійснюється спільнотою, організованою в рамках процесу спільноти Java. Мова та основні технології, що її реалізують, розповсюджуються під ліцензією GPL[5].

Переваги використання мови Java:

− використання в корпоративних програмах, Java здатна підтримати будівельні блоки системи або різні бібліотеки, з їх допомогою створюють необхідні функції;

− запуск програм у «пісочниці» з усуненням поширених, вразливих об'єктів відповідно до політики безпеки;

− незалежне становище від платформ;
− можливість автоматичного керування пам'яттю з одночасним незалежним запуском потоків.

Недоліки використання мови Java:

− низька швидкість та безпека (всі мови з високим рівнем страждають малою продуктивністю, цьому сприяють різні функції – очищення пам'яті, налаштування, блокування);
− відсутній нативний дизайн (щоб використовувати інтерфейс, орієнтований для мови потрібно вивчити кожен, та вибрати шаблон інакше буде помітна невідповідність фрагментів);
− багатослівність та складність коду (мова з довгими, важкими пропозиціями допомагає під час вивчення).

### 2.2.3 Мова програмування C++

C++ – це мова програмування загального призначення, створена Б'єрном Страуструпом, як розширення мови програмування C. Мова значно розширилася з часом, і сучасна C++ тепер має об'єктно-орієнтовані, універсальні та функціональні можливості на доповнення до засобів низькорівневого маніпулювання пам'яттю. Вона майже завжди реалізується як скомпільована мова, і багато постачальників надають компілятори C++, у тому числі FreeSoftwareFoundation, LLVM, Microsoft, Intel, Oracle та IBM, тому вона доступна на багатьох платформах[5].

Переваги мови C++:

− портативність;
− об'єктно-орієнтованість;
− мульти-парадигма;
− низькорівневі маніпуляції;
− управління пам'яттю;
− підтримка великої спільноти;

− сумісність з C;

− масштабованість.

Недоліки мови C++:

− використання покажчиків;

− проблема безпеки;

− відсутність збирача сміття.

### 2.2.4 Мова програмування Python

Мова програмування Python – це високорівнева мова програмування загального призначення. Філософія її дизайну наголошує на зручності читання коду з використанням значних відступів. Її мовні конструкції та об'єктно-орієнтований підхід покликані допомогти програмістам писати чіткий, логічний код для невеликих та великих проєктів. Python має динамічну типізацію та складання сміття. Python підтримує кілька парадигм програмування, включаючи структуроване (особливо процедурне), об'єктно-орієнтоване та функціональне програмування. [5]

Переваги використання мови програмування Python:

− простота у використанні та вивченні;

− підвищена продуктивність;

− гнучкість;

− велика бібліотека;

− підтримуюча спільнота.

Недоліки використання мови програмування Python:

− швидкість;

− споживання пам'яті;

− мобільна розробка;

− доступ до бази даних;

− помилки часу виконання;

− простота.

# 3 ХАРАКТЕРИСТИКА ФАЙЛОВОЇ СИСТЕМИ ДЕСКТОПНОГО ІГРОВОГО ЗАСТОСУНКУ

Файлова система є важливим аспектом розробки десктопного ігрового застосунку. Вона забезпечує організацію та управління ресурсами, такими як зображення, звуки, моделі персонажів та інші елементи гри. Характеристика файлової системи десктопного ігрового застосунку дозволяє визначити її основні риси та функціональні можливості. В даному пункті буде розглянуто структуру файлової системи, формати файлів, способи організації ресурсів та інші аспекти, що стосуються управління файлами в контексті розробки десктопного ігрового застосунку (див.рис.1).

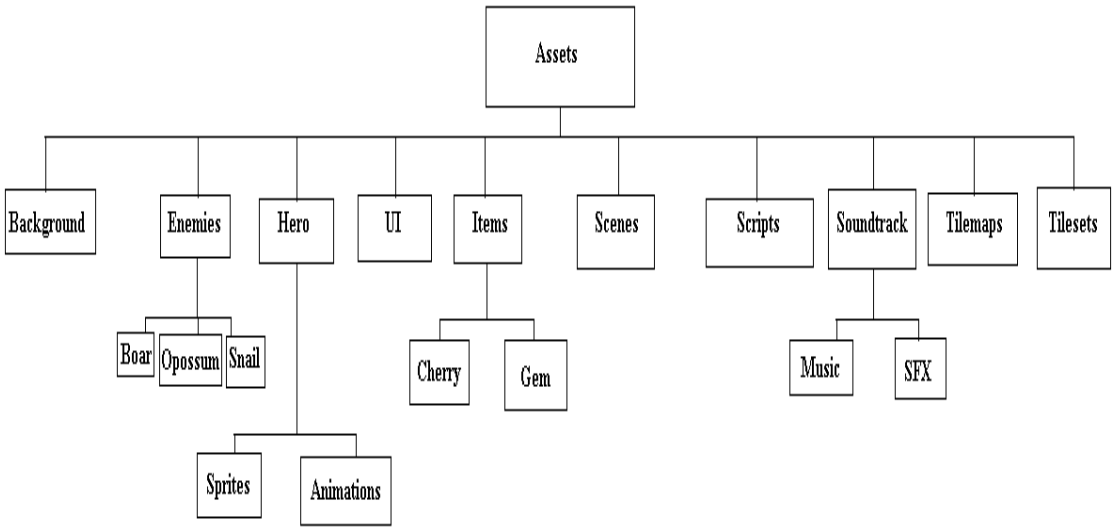


Рисунок 1 − Архітектура файлової системи

У директорії Background зберігаються спрайти об'єктів заднього плану, які використовуються у головному меню та інших сценах. Background відображається за головними об'єктами і зазвичай має статичний характер, тобто не взаємодіє з гравцем чи іншими об'єктами.

У директорії Enemies знаходяться директорії зі спрайтами ворогів. Це графічні зображення або графічні об'єкти, які використовуються в

комп'ютерних іграх і анімації. Вони є основними будівельними блоками для візуального представлення об'єктів, персонажів, тла, предметів тощо в 2D іграх.

У директорії Hero знаходяться директорії Sprites та Animations, які, в свою чергу, містять спрайти та готові анімації головного героя.

У директорії UI знаходяться спрайти елементів користувацького інтерфейсу, кнопок, панелей і т.д. UI (User Interface) - це інтерфейс користувача, який дозволяє взаємодіяти з ігровим застосунком. в 2D іграх UI використовується для створення графічних елементів, які відображаються на екрані гри і дозволяють гравцям взаємодіяти з грою.

У директорії Items знаходяться директорії зі спрайтами предметів, які є у грі. В даному випадку це спрайти, які відповідають за лікування героя, та кристаллів, збір яких є умовою завершення ігрового рівня.

У директорії Scenes знаходяться файли сцен, які містять усю інформацію про об'єкти в сцені, їх стан, налаштування, розташування і т.д. Це простір, де відбувається відтворення гри або конкретного рівня гри. Вона включає в себе всі елементи, об'єкти, налаштування та логіку, необхідні для відображення та функціонування гри.

У директорії Scripts знаходяться скрипти з програмним кодом, які, в свою чергу, прикріплюються до об'єктів усередині сцени. Скрипт – це програмний код, написаний на певній мові програмування, який використовується для керування поведінкою об'єктів і реалізації логіки гри.

У директорії Soundtrack знаходяться дві директорії Music та SFX, які містять фонову музику та звуки для об'єктів у сцені.

У директорії Tilemaps зберігаються карти тайлів. (тайлова карта представляє собою сітку, що складається з багатьох комірок (тайлів), кожна з яких може містити певне зображення або спрайт.

У директорії TileSets знаходяться атласи спрайтів, а також вже нарізані спрайти. Атлас спрайтів – це об’єднані та оптимізовані в одному файлі спрайти.

## 3.1 UML діаграми класів використовуваних у десктопному ігровому застосунку

Клас "CameraController"(див. рис.2) реалізує логіку слідкування камери за героєм.

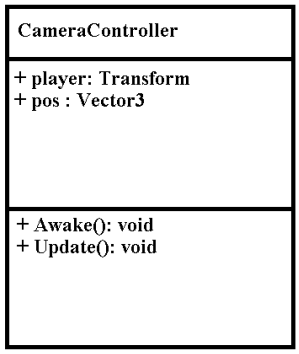


Рисунок 2 − Клас CameraController

Клас "Entity"(див. рис.3) реалізує загальні методи та змінні для всіх сутностей. Клас "Hero" відповідає за логіку станів та управління об'єктом. Класи "Boar", "Snail" і "Possum" керують поведінкою та станом об'єктів.

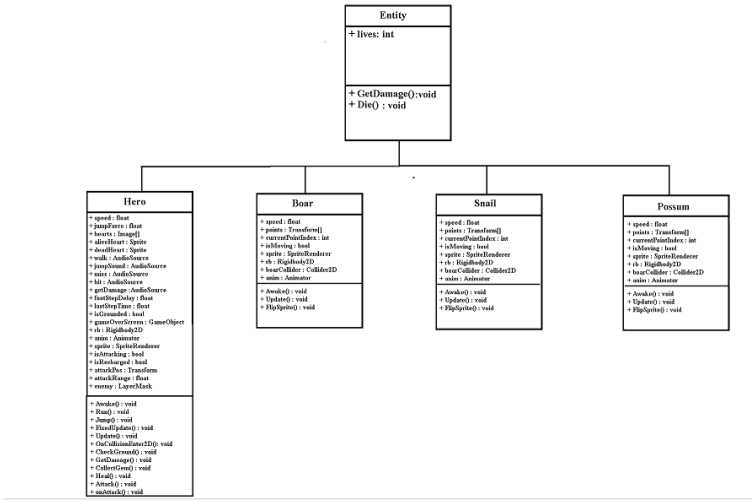


Рисунок 3 − Клас "Entity" і його нащадкові класи

Клас "Cherry"(див. рис.4) відповідає за управління об'єктами зі шару "heal", тобто тими, які відповідають за лікування героя.

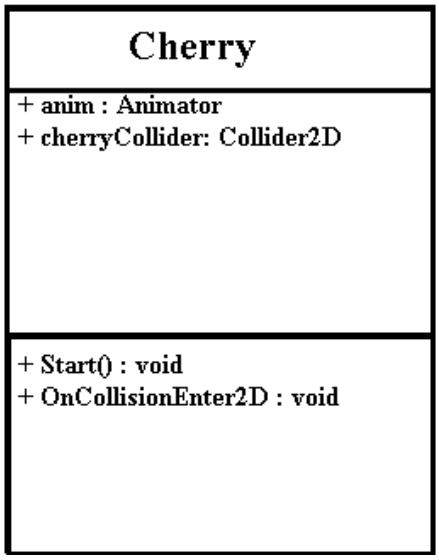


Рисунок 4− Клас Cherry

Клас "Gem"(див. рис.5) відповідає за об'єкти з шару "gems", збір яких є умовою для проходження ігрового рівня.

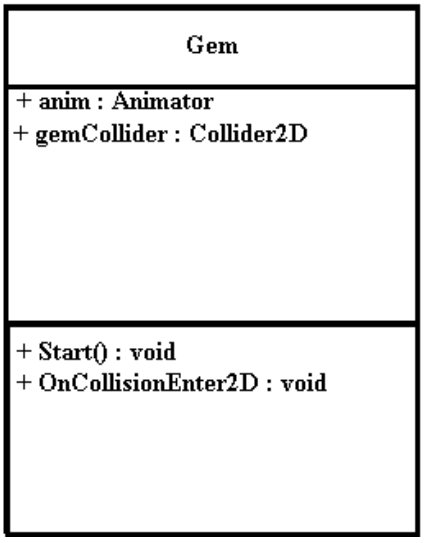


Рисунок 5 − Клас Gem

Клас "GameOver"(див. рис. 6) відповідає за керування прихованою частиною інтерфейсу, яка з'являється при смерті героя.

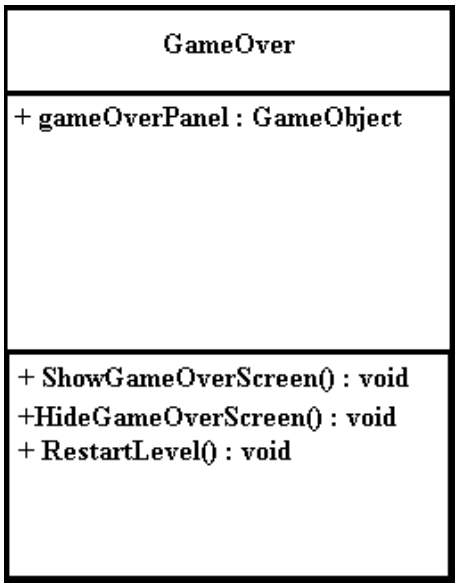


Рисунок 6 − Клас GameOver

Клас "Parallax_Background"(див. рис. 7) реалізує ефект паралаксу для заднього фону сцени, що створює відчуття глибини і перспективи.

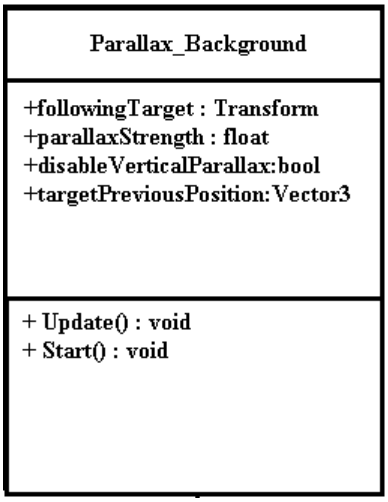


Рисунок 7 − Клас Parallax_Background

Клас "PauseMenu"(див. рис. 8) відповідає за управління початково прихованим меню паузи.

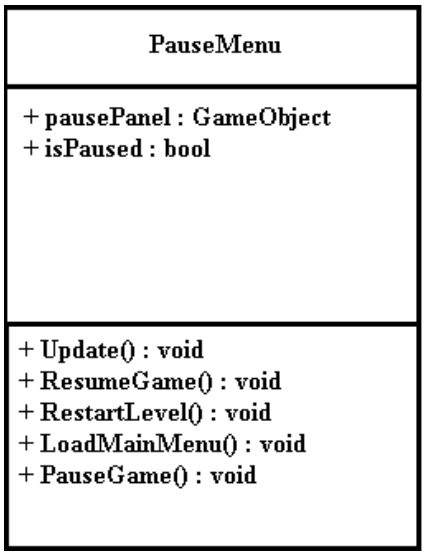


Рисунок 8 − Клас PauseMenu

# 4 ОПИС ВИКОРИСТОВУВАНИХ ЗАСОБІВ РОЗРОБКИ ДЕСКТОПНОГО ІГРОВОГО ЗАСТОСУНКУ

## 4.1 Опис Unity game engine

Unity − одна з найбільш швидкорозвиваючихся платформ для побудови ігор. Це не просто движок, а середовище для розробки комп'ютерних ігор, в якому поєднані різні програмні засоби, що використовуються при створенні програмного забезпечення − текстовий редактор, компілятор, відлагоджувач тощо[3].

При цьому завдяки зручності використання Unity робить процес створення ігор максимально простим і комфортним, а мультиплатформність движка дозволяє охопити якомога більше ігрових платформ і операційних систем. Серед ігрових движків Unity займає далеко не останнє місце. Його використовують як великі розробники, так і (набагато частіше) невеликі незалежні студії[1].

В першу чергу, движок Unity надає можливість розробляти ігри без особливих знань. Тут використовується компонентно-орієнтований підхід, в рамках якого розробник створює об'єкти (наприклад, головного героя) і додає до них різні компоненти (візуальне відображення персонажа і способи керування ним). Завдяки зручному інтерфейсу Drag & Drop та функціональному графічному редактору движок дозволяє малювати карти і розміщувати об'єкти в реальному часі і відразу ж перевіряти результат.

Другою перевагою движка є наявність великої бібліотеки ассетів та плагінів, за допомогою яких можна значно прискорити процес розробки гри. Їх можна імпортувати і експортувати, додавати в гру цілі заготовки − рівні, ворогів, паттерни поведінки штучного інтелекту та інше. Багато ассетів доступні безкоштовно, інші пропонуються за невелику суму, а за бажанням можна створювати власний контент, публікувати його в Unity Asset Store і отримувати від цього прибуток.

Третій сильний бік Unity − підтримка великої кількості платформ,

технологій, API. Ігри, створені на цьому движку, можна легко портувати між ОС Windows, Linux, OS X, Android, iOS, на консолі сімейства PlayStation, Xbox, Nintendo, на VR- і AR-пристрої[1]. Unity підтримує DirectX і OpenGL, працює з усіма сучасними ефектами рендерингу, включаючи останню технологію променевого прослідковування в реальному часі. Фізика твердих тіл, ragdoll і тканини, система Level of Detail, колізії між об'єктами, складні анімації − все це можна реалізувати з допомогою цього движка.

Стереотипна думка про те, що движок придатний тільки для невеликих інді-ігор і не здатний надати красиву графіку, давно вже не актуальна: достатньо подивитися технодемо ADAM, The Blacksmith і Book of the Dead від розробників Unity, щоб переконатися в його виняткових можливостях. Нарешті, Unity доступний безкоштовно, що відкриває перед незалежними розробниками двері в галузь ігор. Звичайно, є обмеження. Безкоштовна версія движка показує логотип Unity перед запуском гри. Pro-версія коштує 150 доларів на місяць, що не так вже й багато порівняно з іншими движками, при цьому базова версія має ту саму функціональність, що й професійна.

При всіх своїх перевагах, движок має й свої недоліки. Так, якщо команда захоче розробити щось складніше, ніж простий клікер або платформер, їй доведеться знайти програміста на C#, який напише скрипти і компоненти, впровадить їх у гру і зробить їх працюючими. З цього випливає інша проблема движка Unity − повільність.

Створення масштабних, складних сцен з багатьма компонентами може негативно вплинути на продуктивність гри, в результаті чого розробникам доведеться витратити додатковий час і ресурси на оптимізацію, а можливо − навіть видалення деяких елементів з проєкту. Крім того, застосунки, створені на Unity, досить "важкі": навіть найпростіша піксельна гра може займати кілька сотень мегабайт на ПК. Так, для жорсткого диска комп'ютерів це невеликий обсяг, але якщо проєкт розробляється й для мобільних платформ, варто подумати про оптимізацію його розміру[4].

Unity дозволяє швидко створювати об'єкти, розташовувати їх та

зв'язувати, створювати прості сцени, використовувати власний контент та зміст магазину активів. Оскільки движок має велику аудиторію користувачів, знайти рішення будь-якої проблеми не складе труднощів − спільнота та розробники з радістю допоможуть початкуючому розробнику з виникаючими питаннями, а офіційні та користувацькі блоги та навчальні курси (в тому числі й українською мовою) нададуть всі необхідні знання. Нижче наведено невеликий список найвідоміших проєктів створених за допомогою Unity.

Cuphead, рік створення 2017, розробники StudioMDHR Entertainment (Канада). Ця платформер-гра створена у жанрі RunAndGun. Вона була заснована на персонажах мультфільму The Cuphead Show. Також за своєю анімацією гра дещо нагадує мультфільм "Міккі Маус", тому її досить часто грають діти.

Temple Run, рік створення 2011, розробники чоловік і дружина Кіт Шепард і Наталія Лук'янова. Лише за три дні після виходу версії для Android, гру завантажили мільйон разів. Загальна кількість завантажень за два роки склала 180 мільйонів, а до 2014 року досягла мільярда. Звичайно, сьогодні повторити такий успіх не вийде, адже початок 2010-х років був "золотим віком" мобільних ігор. Але наступні роки покажуть, що на Unity можна створювати і великі проєкти.

Ori and the Blind Forest, рік створення 2015, розробники MOON STUDIOS. Це одна з найкрасивіших 2D-ігор за останні десять років: тут є вручну намальовані фони, в яких жодна деталь не повторюється протягом всієї гри, і захоплююча плавна анімація, і ефекти, а також естетика, близька духом до робіт студії Ghibli ("Принцеса Мононоке", "Унесені призраками", "Мій сусід Тоторо").

SuperHot, рік створення 2016, розробники SUPERHOT Team. Одним з найуспішніших комерційних проєктів року став вихід гри SuperHot − незвичайного шутера, в якому час рухається лише разом з рухом героя. Гру створила невелика команда з Польщі. Спочатку це був прототип, створений на гейм-джемі 7 Day FPS Challenge. Коли розробники зрозуміли, що їх грою

цікавляться, вони запустили кампанію на Kickstarter і зібрали необхідну суму за 24 години. До речі, сума була відносно невелика − всього 100 000 доларів. Фінансові показники SuperHot − дуже хороші для інді з невеликим бюджетом: за різними оцінками, гру купили від 1,2 до 1,5 мільйона разів на всіх платформах. Студія також випустила VR-версію, яка також відмінно себе проявила: продажі склали 800 тисяч копій.

HearthStone, рік створення 2014, розробники Blizzard Entertainment. Карткова онлайн-гра за мотивами всесвіту Warcraft. Її суть полягає у віртуальних сутичках один з одним за допомогою колоди карт з покроковою системою передачі ходів між суперниками протягом матчу. Перед виходом HearthStone поняття "карткова гра" в цифровому середовищі сприймалося досить холодно. Багатосторінковий перелік правил, маса перевантажених механік, загальна недружелюбність до гравця − це спадщина попередніх екземплярів жанру, з якими середньостатистичному користувачеві не дуже хотілося стикатися. Саме тому реліз альфа-версії HearthStone збудив геймерське співтовариство, викликавши велику увагу до гейм−продукту. Його відносна простота та привабливість не тільки повернула надію ветеранам гравцям, а й привела нову аудиторію. Можна сказати, що у жанру карткових ігор відкрився другий подих. HearthStone: Heroes of Warcraft неодноразово отримувала нагороди в різних номінаціях, зокрема "найкраща мобільна гра", "найкраща стратегічна гра", "найкраща багатокористувацька онлайн-гра". Крім того, вона стала одним з провідних кіберспортивних дисциплін у своєму жанрі.

Escape from Tarkov, рік створення 2017, розробники Battlestate Games. Escape From Tarkov − це багатокористувацький хардкорний шутер від першої особи. Згідно з сюжетом гри, дві конкуруючі приватні військові корпорації зіткнулися одна з одною в місті Тарков, спричинивши там паніку та жахливий хаос, що у свою чергу породив безліч бандитських кланів. Ця гра є одним з найунікальніших та особливих проєктів FPS останніх років, але є грою виключно нішевою, тобто не для всіх, а також має дуже високий поріг

входження. Проєкт розкрив можливості движка Unity на всі 100% і показав, що Unity підходить не тільки для малобюджетних інді-ігор.

### 4.2 Опис мови програмування C#

Мова програмування C#, також відома як C Sharp, є однією з найпопулярніших мов програмування, особливо у сфері розробки застосунків для платформи Microsoft. Створений групою інженерів компанії Microsoft під керівництвом Андерса Хейлсберга і Скотта Вільтаумота, C# був розроблений у період з 1993 по 2001 рік. Назва мови, C Sharp, має свою історію та символічне значення. Знак "#" позначає підвищення висоти звуку на півтону, а в разі C# він посилається на розвиток мови від C до C++ і потім до C# (C++++). Ця мова була створена з метою розробки застосунків для платформи Microsoft. На початку 2000-х років Microsoft розробила різні версії технологій та рішень, які використовуються для обміну повідомленнями та даними, а також для створення веб-застосунків. У рамках цього процесу була створена платформа .NET, яка включала кілька мов програмування, включаючи C#. Платформа .NET надавала середовище для розробки нових рішень та включала технологію активних серверних сторінок ASP.NET, яка дозволяла швидко створювати веб-застосунки, що працюють з базами даних.

Мова програмування C# початково використовувалася в сфері веб-розробки та має деякі спільні риси з мовою Java, оскільки обидві мови базуються на об'єктно-орієнтованому підході. Насправді, приблизно 75% синтаксичних можливостей C# збігається з Java. Він також запозичує близько 10% можливостей від C++ і 5% від Visual Basic.

C# надає програмістам об'єктно-орієнтований підхід до розробки масштабних та гнучких застосунків, які можуть бути легко розширені за необхідності. Мова має такі корисні функції, як інкапсуляція, успадкування, поліморфізм, перевантаження операторів та статична типізація. Розробники

постійно працюють над вдосконаленням мови C#. Кожна нова версія вносить корисні доповнення, такі як лямбди, динамічне зв'язування, асинхронні методи та інші, які покращують продуктивність та ефективність розробки застосунків за допомогою цієї мови[2].

C# є відносно молодою мовою програмування, але вже здобула широке поширення. Перша версія C# була випущена разом з Microsoft Visual Studio .NET в лютому 2002 року. Мова C# вважається універсальною і застосовується в різних галузях. Один з важливих аспектів застосування C# − розробка прогресивних бізнес-застосунків. Ця мова дозволяє створювати потужні та ефективні рішення для бізнесу. Вона також широко використовується для розробки відеоігор. C# використовується для створення ігор для різних платформ, включаючи Windows, macOS, Android та iOS. Особливо популярне поєднання мови C# та ігрового движка Unity дозволяє розробникам створювати якісні та захоплюючі ігри. C# також використовується для розробки програмного забезпечення, яке забезпечує захист операційних систем та застосунків. За допомогою C# створюються утиліти, які допомагають блокувати віруси та запобігати кібератакам. Великі компанії та організації використовують програмне забезпечення на основі C# для забезпечення безпеки систем.

Windows OS є прикладом широкого використання C# у розробці. Основні компоненти операційної системи Windows, такі як Skype, Internet Explorer, Visual Studio, Microsoft Office (включаючи Word, PowerPoint, Excel, Outlook та інші), Adobe (Photoshop, Lightroom), Mozilla Firefox та Winamp, створені з використанням C#. C# також вважається однією з найкращих мов програмування для розробки мобільних застосунків. З його допомогою можна створювати нативні застосунки для різних операційних систем, таких як iOS та Android, використовуючи інтегроване середовище розробки Xamarin. Серед значних проєктів, розроблених з використанням C#, можна виділити такі, як Slack, Pinterest, Tableau, The World Bank та багато інших. Більшість програм плиткового дизайну у Windows 8 також написані на C# та

XAML, що підтверджує його широке застосування в різних галузях розробки.

C# має кілька важливих переваг:

- незалежність від апаратного забезпечення (завдяки віртуальній машині .NET Framework, програми на C# можуть виконуватися на різних пристроях та операційних системах без необхідності адаптації коду. Це в свою чергу забезпечує високу переносимість застосунків);
- підтримка екосистеми Windows (C# розроблено компанією Microsoft та тісно інтегровано з платформою Windows. Це дозволяє розробникам ефективно використовувати інструменти та ресурси, надані Microsoft, для створення застосунків під Windows);
- керування пам'яттю, мова програмування (C# надає автоматичне керування пам'яттю за допомогою збору сміття. Це звільняє розробників від необхідності явного керування пам'яттю та усуває проблеми, пов'язані з витоками пам'яті або звільненням "мертвого" коду);
- сувора типізація (C# має сувору типізацію, що означає, що кожна змінна повинна бути оголошена з певним типом даних. Це сприяє більш передбачуваному та безпечному програмуванню, оскільки операції виконуються тільки з відповідними типами даних, що допомагає виявляти помилки на ранніх етапах розробки);
- велика спільнота (C# має активну та широку спільноту розробників. багато ресурсів, форумів, спільнот і чатів доступні для спілкування, обміну досвідом та отримання допомоги. Наявність великої спільноти спрощує процес вивчення та розробки на C# та надає доступ до багатьох ресурсів та експертів);
- синтаксичний цукор (C# пропонує деякі синтаксичні конструкції, що роблять код більш лаконічним і зрозумілим. Це дозволяє скорочувати кількість коду без ушкодження його логіки та покращує зрозумілість програми. Однак потрібно зберігати баланс та уникати

зайвого використання таких конструкцій, щоб не ускладнювати розуміння коду іншим розробникам).

C# також має деякі недоліки:

- невисока швидкість, використання віртуальної машини .NET Framework для виконання коду на C# може призвести до невеликого зниження продуктивності порівняно з деякими мовами, які компілюються у нативний код (однак сучасні версії .NET та оптимізації у компіляторі JIT дозволяють досягти прийнятних швидкостей виконання);
- безпека: код на C# може піддаватися декомпіляції, що може створювати уразливості у захисті інтелектуальної власності або конфіденційної інформації (хоча це стосується багатьох мов програмування, C# є більш уразливим до декомпіляції, ніж деякі інші мови);
- обмежена взаємодія з обладнанням: оскільки C# є високорівневою мовою програмування, його зазвичай не використовують для розробки низькорівневого програмного забезпечення, яке вимагає прямої взаємодії з апаратурою (для таких випадків часто використовують мови низького рівня, такі як C++).

Однак варто відзначити, що ці недоліки не є критичними і можуть бути контрольованими. більшість застосунків, включаючи ігри, бізнес-застосунки та веб-сервіси, можуть бути успішно розроблені з використанням C#. у разі потреби можна поєднувати C# з іншими мовами або використовувати спеціалізовані інструменти для вирішення певних завдань. Однією з ключових особливостей C# є його зрозумілий та зрозумілий синтаксис, який робить його доступним навіть для новачків у програмуванні. мова має зручні засоби для організації коду, такі як класи, об'єкти, успадкування та поліморфізм, що дозволяє розробникам створювати модульні та легко підтримувані застосунки.

C# перевершує базові конструкції мови завдяки своєму обширному

набору функцій та класових бібліотек, які значно спрощують процес розробки застосунків у різних сферах. Наприклад, для роботи з графікою та аудіо використовуються такі бібліотеки, як Unity та DirectX, що дозволяють створювати захоплюючі візуальні ефекти та звукове супроводження в іграх та застосунках. Крім того, C# має можливості роботи з мережею, базами даних та багатопотоковістю, що робить його ідеальним вибором для розробки складних та масштабних застосунків. C# також широко застосовується у серйозному програмному забезпеченні. Наприклад, система управління базами даних Microsoft SQL Server розроблена з використанням C# і пропонує потужні інструменти для зберігання, обробки та аналізу даних, широко використовувані в корпоративному середовищі. Ще одним прикладом є система управління вмістом (CMS) Sitecore, яка дозволяє створювати та керувати складними налаштованими веб-сайтами. Важливо зауважити, що наведені приклади програмного забезпечення є лише невеликою частиною багатьох проєктів, створених з використанням C#. Ця мова програмування широко використовується в різних галузях, таких як ігрова індустрія, веб-розробка, фінансові застосунки, медичні системи та інші. Гнучкість C#, його потужність та широка екосистема інструментів роблять його невід'ємним інструментом для розробників, які прагнуть створювати високоякісне програмне забезпечення. C# вважається відносно молодою мовою програмування. Вона може бути безпечним вибором як для новачків, так і для досвідчених фахівців, особливо для тих, хто вже має досвід роботи з мовами програмування родини C. C# є мовою програмування, яка активно розвивається і має багато переваг, що робить її привабливою для фахівців та потенційних роботодавців. Крім того, код, написаний на C#, легко зчитується та розуміється, що забезпечує швидку та успішну модифікацію вмісту[2]. Для роботи з C# рекомендується використовувати низку інструментів, які значно спрощують розробку і підвищують ефективність. Наприклад, WPF дозволяє створювати інтерфейси для різних екранів, Xamarin надає фреймворк для розробки

мультиплатформних програм, а платформа .NET спрощує зв'язування коду з клієнт-серверними утилітами веб-застосунків. Крім того, Entity Framework надає можливості роботи з базами даних у вигляді об'єктів, а LINQ полегшує фільтрацію та сортування даних різного типу. Для розробки рекомендується використовувати середовище Visual Studio, яке є класичним інструментом розробки. Хоча C# на даний момент може не бути найпопулярнішою мовою програмування, вона все більше поширюється і активно розвивається. Постійні покращення роблять її все більш функціональною, що дозволяє розробникам створювати високоякісне програмне забезпечення, придатне для різних платформ.

# 5 ОПИС РОЗРОБКИ ДЕСКТОПНОГО ІГРОВОГО 2D ЗАСТОСУНКУ З ВИКОРИСТАННЯМ UNITY GAME ENGINE

## 5.1 Опис реалізації «Головного меню»

Після запуску застосунку відразу з’явиться головне меню. В ньому присутнє найменування гри, вказана її версія, а також дві кнопки: "Press to START" − яка запускає перший рівень, і "Вихід" − яка закриває застосунок (див. рис.9)

Для кнопок було використано наступний код:

```
public class Button_group : MonoBehaviour {
       public void PlayGame(){
          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
       }

       public void ExitGame() {
          Application.Quit();
       }
```


Рисунок 9 − Головне меню

У методі PlayGame() використовується функція LoadScene, яка завантажує сцену за її індексом в Unity, з урахуванням того, що назви сцен можуть змінюватися. Код написаний таким чином, щоб отримати індекс

поточної сцени (головне меню) і завантажити наступну сцену, яка йде в списку сцен Unity. Код для кнопки "Exit" містить всього один рядок, який просто припиняє роботу застосунку.

### 5.2 Опис героя та його навколишнього середовища

Після натискання кнопки "Start", запускається перший рівень, а камера центрується на об'єкті "Hero". Для досягнення цього було використано наступний код:

```
public class CameraController : MonoBehaviour {
   [SerializeField] private Transform player;
   private Vector3 pos;
   privatevoid Awake(){
       if (!player)
       player = FindObjectOfType<Hero>().transform;
    }
   privatevoid Update() {
      pos = player.position;
      pos.z = -10f;
      pos.y += 3f;
      transform.position = Vector3.Lerp(transform.position, pos, Time.deltaTime);
   }
}
```

У методі Awake() відбувається ініціалізація змінної player шляхом пошуку об'єкта типу Hero у сцені і отримання посилання на його компонент transform. Якщо посилання на player не було встановлено в редакторі Unity, то відбувається автоматичний пошук об'єкта. Метод Update() викликається кожен кадр і відповідає за оновлення позиції камери. Усередині методу відбувається наступне:

- запис позиції ігрового об'єкта player в змінну pos за допомогою player.position;
- встановлення значення pos.z на −10f, щоб камера знаходилась на

певній відстані по осі Z від персонажа;

– збільшення значення pos.y на 3f, щоб камера знаходилась над персонажем;
– застосування ефекту плавного переходу (Lerp) між поточною позицією камери і позицією pos з використанням Time.deltaTime.

Отже, цей код забезпечує плавне слідування камери за персонажем у грі, зберігаючи певну відстань по осі Z і відступ по осі Y відносно персонажа.

На початку рівня на задньому фоні будуть зустрічатися текстові підказки(див. рис. 10), які стосуються управління персонажем.


Рисунок 10 − Старт першого рівня

При натисканні кнопок "A" або "D" персонаж розпочне рух по рівню вліво або вправо відповідно розташуванню кнопок на клавіатурі. Для переміщення персонажа було використано наступний код:

```
        Vector3 dir = transform.right * Input.GetAxis("Horizontal");
        transform.position = Vector3.MoveTowards(transform.position,transform.position +
dir, speed * Time.deltaTime);
        sprite.flipX = dir.x < 0.0f;
```

Створюється змінна dir типу Vector3, яка визначає напрямок руху персонажа по горизонтальній осі. Значення Input.GetAxis("Horizontal") представляє вхідне значення горизонтальної осі вводу (наприклад, кнопок "Вліво" або "Вправо"). Множення transform.right на значення горизонтальної осі дає вектор, який вказує напрямок руху вправо або вліво.

За допомогою методу Vector3.MoveTowards() відбувається переміщення позиції персонажа (transform.position) до нової позиції (transform.position + dir) з певною швидкістю (speed * Time.deltaTime) (див. рис. 11) [7].


Рисунок 11 − Рух персонажу

Щоб персонаж завжди дивився вправо, використовується перевірка dir.x < 0.0f. Якщо значення dir.x від'ємне (персонаж рухається вліво), тоді властивість sprite.flipX встановлюється в true, що відзеркалює спрайт герою горизонтально.  Для більш приємного візуалу, був доданий ефект паралаксу, для заднього фону гри. Ефект паралаксу в 2D іграх створює ілюзію глибини і перспективи, дозволяючи заднім планам рухатися повільніше за передні плани. Цей ефект надає грі відчуття глибини і додає пластичності до ігрового світу. Принцип роботи ефекту паралаксу полягає в тому, що кожен шар

заднього плану має свою швидкість руху, яка може бути повільнішою або швидшою, ніж швидкість переднього плану або камери. При русі камери чи персонажа, задні плани будуть рухатися зі своєю швидкістю, створюючи ефект глибини і руху віртуального простору. При русі персонажа, фон поділяється на віддалені об'єкти (море) і ближні (темно-зелені пагорби з рослинністю), що створює візуальний ефект об'ємності фону.

```
public class Parallax_Background : MonoBehaviour {
    [SerializeField] Transform followingTarget;
    [SerializeField] [Range(0f, 1f)] float parallaxStrength = 0.1f;
    [SerializeField] bool disableVerticalParallax;
    Vector3 targetPreviousPosition;
       void Start()  {
              if (!followingTarget)
              followingTarget = Camera.main.transform;
              targetPreviousPosition = followingTarget.position;
       }
       public void Update() {
          var delta = followingTarget.position – targetPreviousPosition;
          if (disableVerticalParallax) delta.y = 0;
          targetPreviousPosition = followingTarget.position;
          transform.position += delta * parallaxStrength;
       }
}
```

Змінна transform followingTarget, яка зберігає посилання на об'єкт, за яким буде слідкувати задній план. float parallaxStrength: Змінна, що визначає силу паралаксу. Чим вище значення, тим сильніше буде зсув заднього плану відносно руху наступної цілі. Змінна bool disableVerticalParallax, що визначає, чи повинен бути вимкнений вертикальний паралакс. Якщо встановлене значення true, то вертикальний рух наступної цілі не буде впливати на задній план. Попередня позиція цілі Vector3 targetPreviousPosition.

У методі Start() відбувається ініціалізація початкових значень. Якщо не вказаний об'єкт для слідкування (followingTarget), то він встановлюється на головну камеру. Запам'ятовується початкова позиція наступної цілі[7].

Метод Update() викликається кожен кадр[6]. Він розраховує зсув (delta) між поточною і попередньою позиціями цілі. Якщо вертикальний паралакс вимкнено (disableVerticalParallax), то значення delta.y встановлюється в 0. Потім оновлюється попередня позиція наступної цілі. І, нарешті, відбувається зсув позиції заднього плану на основі delta і parallaxStrength.

Для подолання різних перешкод і ворогів персонаж може стрибати. (див. рис. 12)


Рисунок 12 – Стрибок

```
private void Jump()  {
        rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
        jumpSound.Play();
}
```

Для цього застосовується фізична сила, яка додається до компонента Rigidbody2D (rb) персонажа. Щоб застосувати силу стрибка, використовується метод AddForce(). В якості аргументу передається вектор

напрямку стрибка, помножений на величину сили стрибка (transform.up * jumpForce), де transform.up представляє вертикальний напрямок у просторі. Для забезпечення миттєвого стрибка з заданою силою, використовується режим ForceMode2D.Impulse.

Далі на рівні зустрічаються кілька видів противників: Кабан, Улітка і Опосум. (див. рис.13)

Рисунок 13 – Види противників

Для реалізації логіки та стану їх поведінки використовується однотипний скрипт, в якому відрізняються лише швидкість і кількість балів здоров'я, представлено в Додатку А. Нижче описані змінні, які використовуються в даному програмному коді:

- speed – визначає швидкість руху;
- points – масив точок, до яких буде рухатися об'єкт;
- currentPointIndex – індекс поточної точки в масиві points;
- isMoving – прапорець, що вказує, чи рухається об'єкт в початковому напрямку;
- spriteRenderer – компонент, що відповідає за відображення спрайта об'єкта4
- anim – компонент, що керує анімацією;
- boarCollider – компонент Collider2D;
- rb – компонент Rigidbody2D.

У методі Update() перевіряється прапорець isMoving. Якщо кабан рухається в початковому напрямку, він рухається до поточної цільової точки

targetPosition з заданою швидкістю speed.

Коли кабан досягає цільової точки з заданою близькістю (Vector2.Distance(transform.position, targetPosition) < 0.2f), прапорець isMoving стає true, і кабан повертається, викликаючи метод FlipSprite(). Якщо кабан досягає крайньої точки руху, змінюється індекс currentPointIndex, щоб кабан рухався в зворотному напрямку. Якщо кількість життів lives кабана стає менше 1, то колайдер вимикається, Rigidbody2D стає кінематичним, швидкість стає рівною 0, і запускається анімація смерті через корутину DeathAnimation().[7] Для того, щоб герой міг атакувати противників(див. рис.14), було необхідно створити кілька методів.

Рисунок 14 – Атака героєм противника

```
private void Attack() {
   if (isGrounded && isRecharged) {
        state = states.attack;
        isAttacking = true;
        isRecharged = false;

        StartCoroutine(AttackAnimation());
        StartCoroutine(AttackCoolDown());
```

```
        }
    }

    private void OnAttack()  {
      Collider2D[]    colliders    =    Physics2D.OverlapCircleAll(attackPos.position,
attackRange, enemy);
      if (colliders.Length == 0) { attackMiss.Play();  }
      if (colliders.Length > 0) { attackHit.Play(); }
      for (int i = 0; i < colliders.Length; i++) {
          colliders[i].GetComponent<Entity>().GetDamage();
      }
    }

    private IEnumerator AttackAnimation() {
      yield return new WaitForSeconds(0.4f);
      isAttacking = false;
    }

    private IEnumerator AttackCoolDown() {
      yield return new WaitForSeconds(0.5f);
      isRecharged = true;
    }
```

Метод Attack() виконує атаку героя. Він перевіряє, чи герой знаходиться на землі (isGrounded) і чи заряджена атака (isRecharged). Якщо умови виконуються, метод встановлює стан героя в states.attack, встановлює прапорець isAttacking в значення true та заряджає атаку, встановлюючи прапорець isRecharged в значення false. Потім він запускає дві корутини: AttackAnimation() для анімації атаки та AttackCoolDown() для перезарядки атаки. Метод OnAttack() викликається при здійсненні атаки героя. Він використовує функцію Physics2D.OverlapCircleAll() для виявлення колайдерів ворогів у заданому радіусі від позиції атаки. Якщо колайдери не були виявлені, відтворюється звук промаху атаки. Якщо колайдери були виявлені, відтворюється звук успішної атаки. Потім проходиться по кожному виявленому колайдеру та викликається метод GetDamage() компонента Entity, щоб нанести шкоду ворогові.

AttackAnimation() – ця корутина відображає анімацію атаки героя. Вона

очікує 0.4 секунди, а потім встановлює прапорець isAttacking в значення false, щоб герой міг здійснити наступну атаку.

AttackCoolDown() – відображає перезарядку атаки героя. Вона очікує 0.5 секунди, а потім встановлює прапорець isRecharged в значення true, щоб герой знову міг здійснити атаку після перезарядки.

### 5.3 Опис реалізації користувацького інтерфейсу

На поточний момент інтерфейс гравця має всього два елементи:

− шкала здоров'я

− індикатор наявності у персонажа кристалів, збір яких є умовою завершення рівня.

При зіткненні героя з колайдером об'єкта з шару "enemy", герой отримує шкоду, а також на індикаторі життів в інтерфейсі зникає одне серце (див. рис.15).


Рисунок 15 − Отримання героєм шкоди

Це відбувається за допомогою наступного коду:

```
if (collision.gameObject.layer == LayerMask.NameToLayer("enemy")) {
   GetDamage();
}
```

МетодGetDamage() :

```
public override void GetDamage() {
   lives--;
   getDamageSound.Play();
   if (lives >= 0 && lives < hearts.Length) {
      hearts[lives].sprite = deadHeart;
   }

  if (lives <= 0) {
     // Все жизни потеряны, обновить отображение сердец
     foreach (Image heart in hearts) {
          heart.sprite = deadHeart;
     }
  }
}
```

Рядок (lives−−;) зменшує кількість життів героя на 1. Після зменшення життів, відтворюється звук отримання урону героєм за допомогою getDamageSound.Play(). Цей звук звертає увагу гравця на факт отримання ушкоджень героєм.

Умовна конструкція if (lives >= 0 && lives < hearts.Length) перевіряє, чи кількість життів знаходиться в допустимому діапазоні для відображення на UI. Якщо умова виконується, то рядок hearts[lives].sprite = deadHeart; встановлює спрайт серця в масиві hearts з індексом, що відповідає кількості життів героя. Таким чином, втрачені життя відображаються на UI як порожні серця, що нагадує гравцеві про його стан.

Умовна конструкція if (lives <= 0) перевіряє, чи у героя залишилися життя. Якщо життів не залишилося (дорівнює або менше 0), то виконується наступний блок коду. Цей блок коду відповідає за обробку ситуації, коли

герой втрачає всі свої життя. Наприклад, може бути викликана функція, яка завершує гру або повертає гравця до попереднього контрольного пункту.

Цикл foreach (Image heart in hearts) перебирає всі елементи масиву hearts, який представляє серця на UI. У середині циклу кожному серцю встановлюється спрайт deadHeart, що означає, що всі серця на UI відображаються як порожні. Це використовується для показу гравцеві, що він втратив усі свої життя і поточний стан гри не дозволяє йому продовжувати.

Для відновлення життів героя в грі присутні об'єкти з шару "heal". При підборі цих об'єктів, кількість життів героя збільшується на одиницю, а порожні серця на UI замінюються на повні серця. (див. рис. 16)


Рисунок 16 – Відновлення кількості життів

```
if (collision.gameObject.layer == LayerMask.NameToLayer("heal")) { Heal(); }
```

Метод Heal() :

```
public void Heal() {
  if (lives < 5)  { lives++; }

  if (lives > 0 && lives <= hearts.Length + 1) {
     hearts[lives - 1].sprite = aliveHeart;
  }
}
```

Умовна конструкція if (lives < 5) перевіряє, чи кількість життів героя не досягла максимального значення (у цьому випадку, 5). Якщо кількість життів

менше максимального значення, виконується наступний блок коду для відновлення життів героя.

Рядок «lives++;» збільшує кількість життів героя на 1. Після цього, умовна конструкція if (lives > 0 && lives <= hearts.Length + 1) перевіряє, чи кількість життів знаходиться в допустимому діапазоні для відображення на інтерфейсі користувача (UI). Рядок hearts[lives − 1].sprite = aliveHeart; встановлює спрайт серця в масиві hearts з індексом, що відповідає кількості життів героя мінус 1 (так як індекси масивів починаються з 0), на спрайт aliveHeart. Це оновлює відображення відновлених життів на інтерфейсі користувача, щоб гравець міг бачити свій поточний рівень життя.

Як умову завершення рівня, в сцену були додані об'єкти з шару "gems" (див. рис.17), які представляють кристали. Щоб завершити рівень, гравець повинен зібрати всі кристали, оскільки вони є необхідною умовою для прогресування до наступного рівня чи до завершення гри.

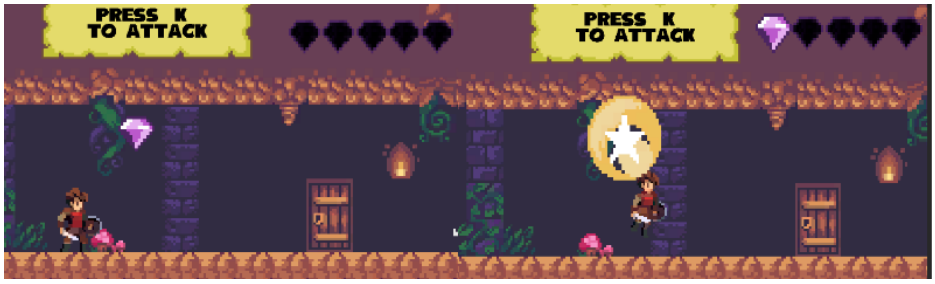


Рисунок 17 − Сбір кристаллів

За їх збір та заповнення шкали на інтерфейсі користувача (UI) відповідає метод, аналогічний методу лікування. Цей метод дозволяє накопичувати та відображати прогрес збіру кристалів :

```
if (collision.gameObject.layer == LayerMask.NameToLayer("gems")) { CollectGem(); }
public void CollectGem()  {
  if (gemsNumb < 5)  { gemsNumb++;  }
  if(gemsNumb>0 && gemsNumb <= gems.Length + 1){ gems[gemsNumb − 1].sprite = gem;}
  if (gemsNumb == 5) { SceneManager.LoadScene(2); }
```

```
}
```

За винятком того, що у випадку отримання п'яти кристалів, активується функція LoadScene, яка завантажує сцену з індексом 2, що відповідає другому рівню гри. Меню паузи (див. рис. 18)дозволяє призупинити гру, якщо це необхідно представлено в Додатку Б.


Рисунок 18 − Меню паузи

У методі Update() відбувається оновлення кожен кадр. Якщо користувач натискає клавішу Escape, перевіряється поточний стан паузи. Якщо гра була на паузі, викликається метод ResumeGame() для продовження гри. Якщо гра не була на паузі, викликається метод PauseGame() для постановки гри на паузу. Метод PauseGame() викликається для постановки гри на паузу. Час в грі зупиняється (Time.timeScale = 0f), змінна isPaused встановлюється в true, а панель паузи (pausePanel) стає активною (SetActive(true)).

Нижче описані методи для кнопок, що присутні на панелі паузи.

Метод ResumeGame() викликається або при натисканні клавіші Escape, або кнопки "Продовжити" в меню паузи, для продовження гри після паузи. Час в грі відновлюється (Time.timeScale = 1f), змінна isPaused

встановлюється в false, а панель паузи (pausePanel) стає неактивною (SetActive(false)).

Метод RestartLevel() викликається при натисканні кнопки перезапуску рівня в меню паузи. Він завантажує поточну сцену за допомогою SceneManager.LoadScene(), встановлює час в грі рівним 1 (Time.timeScale = 1f).

LoadMainMenu() – викликається при натисканні кнопки завантаження головного меню в меню паузи. Він завантажує головне меню, яке відповідає сцені 0, за допомогою SceneManager.LoadScene(), встановлює час в грі рівним 1 (Time.timeScale = 1f) [6]. Коли гравець втрачає всі очки здоров'я, з'являється екран "Кінець гри" (Gameover) (див. рис.19).


Рисунок 19 – Екран “Game Over”

Для цього у методі Update() створено умовну конструкцію, яка перевіряє кількість життів героя:

```
if (lives < 1 && state != states.death)  {
   state = states.death;
   StartCoroutine(GameOverCoroutine());
}
```

При досягненні змінної lives значення 0, стан аніматора героя

переходить у стан death для відтворення анімації смерті, після чого запускається корутина із затримкою часу:

```
private IEnumerator GameOverCoroutine()  {
        yield return new WaitForSeconds(1.5f);
        gameOverScreen.SetActive(true);
        Time.timeScale = 0f;
}
```

Після затримки у 1.5 секунди (щоб анімація смерті героя встигла завершитися), корутина робить приховану панель GameOver активною та зупиняє час в сцені (Time.timeScale = 0f).

### 5.4 Опис реалізації анімацій

Окремим пунктом варто зазначити анімації, процес їх створення включає в себе достатню кількість кроків.  Підготовка спрайтів: необхідно створитиабо імпортувати спрайти(див. рис. 20), які будуть використовуватися для анімації об'єкта.[6].

Рисунок 20 − Підготовка спрайтів з атласу спрайтів героя

Зазвичай спрайти представляють різні кадри анімації, які будуть відтворюватися послідовно.(див. рис. 21)

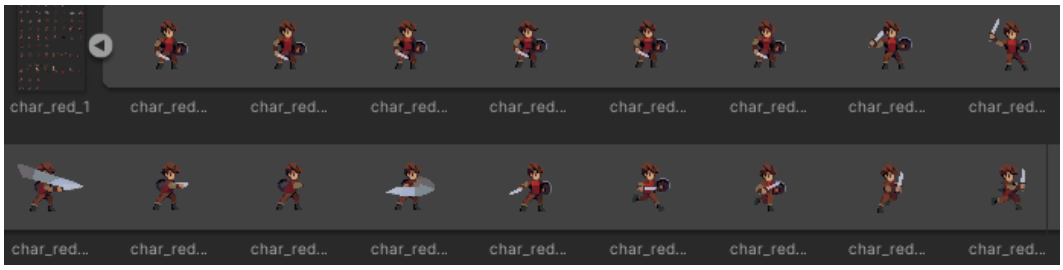


Рисунок 21 − Нарізанні спрайти , готові до використання

Створення анімаційного контролера: анімаційний контролер є складовою частиною об'єкта і визначає логіку відтворення анімації. Він створюється за замовчуванням при створенні першої анімації об'єкта).

Створення станів анімації: в анімаційному контролері необхідно створити стани анімації(див. рис. 22) для кожної групи кадрів. Стан анімації представляє окрему анімацію, яка може бути відтворена.

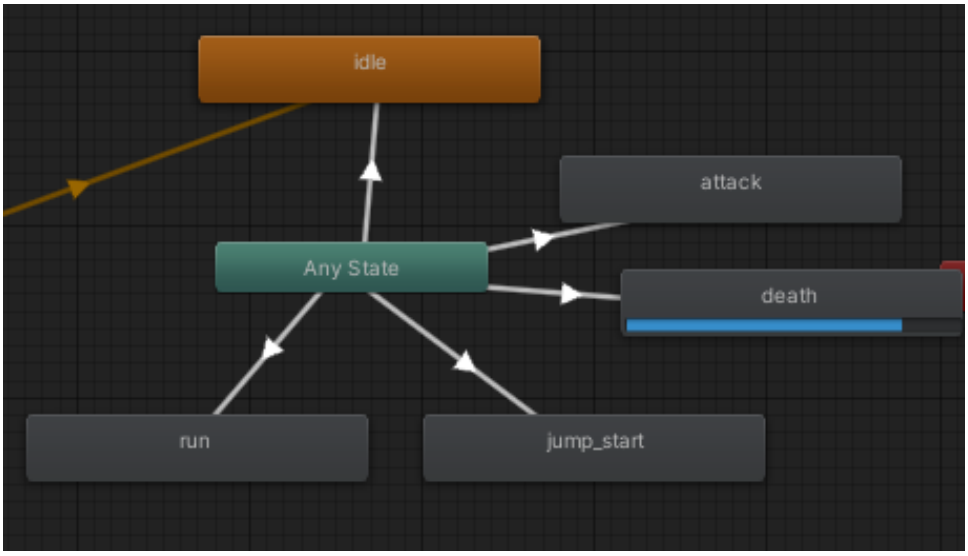


Рисунок 22 − Стани анімації та переходи між ними

Налаштування переходів між станами: потрібно створити переходи між станами анімації. Налаштування параметрів та анімацій: в станах анімації потрібно налаштувати параметри(див. рис.23), такі як тривалість анімації, швидкість відтворення та інші властивості.

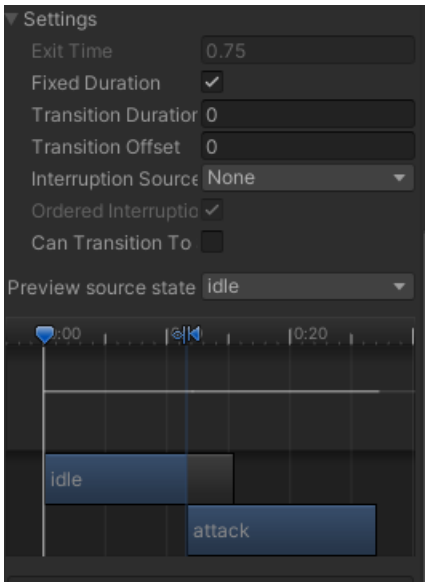


Рисунок 23 − Налаштування анімації

Відтворення анімації: в коді об'єкта потрібно додати код для керування відтворенням анімації.

```
private Animator anim;
private states state     {
  get { return (states)anim.GetInteger("state"); }
  set { anim.SetInteger("state", (int)value); }
}

private void Awake()  {
  anim = GetComponent<Animator>();
}
```

Властивість state надає доступ до параметра "state" в компоненті Animator. Вона має гетер та сетер, які використовують методи GetInteger та

SetInteger об'єкта anim для отримання поточного значення параметра та встановлення нового значення параметра "state" відповідно. При отриманні значення властивості воно приводиться до типу states, який є користувацьким перелічуванням. Метод Awake() виконується при запуску об'єкта і використовується для ініціалізації змінних та компонентів[7]. У даному випадку, всередині методу Awake() відбувається отримання посилання на компонент Animator, прикріплений до того ж об'єкта, на якому знаходиться цей скрипт, і присвоєння його змінній anim.

Як приклад, можна використовувати метод, який викликається при атаці :

```
private void Attack() {
  if (isGrounded && isRecharged) {
     state = states.attack;
     isAttacking = true;
     isRecharged = false;
     StartCoroutine(AttackAnimation());
     StartCoroutine(AttackCoolDown());
  }
}
```

У першу чергу він змінює значення state на attack, що дозволяє запустити анімацію атаки, а в процесі анімації виконувати інші дії. Також потрібно створити перерахування станів.

```
public enum states {
  idle,
  run,
  jump_start,
  attack,
  death
}
```

# ВИСНОВКИ

Дана кваліфікаційна робота бакалавра присвячена розробці десктопного ігрового 2D застосунку з використанням Unity game engine. У процесі дослідження були вивчені можливості Unity для створення ігор і виявлені його переваги порівняно з іншими двигунами. За допомогою Unity були реалізовані основні компоненти гри, такі як графіка, фізика, штучний інтелект та керування персонажем. Були створені різні рівні гри, а також реалізовані механіки геймплею, включаючи керування, зіткнення та взаємодію об'єктів.

Була приділена особлива увага оптимізації продуктивності та виправленню помилок для забезпечення плавного та стабільного ігрового досвіду. Також були реалізовані різні анімації та ефекти для створення привабливого візуального враження.

Для реалізації логіки та функціональності гри використовувалася мова програмування C#. C# є однією з основних мов програмування, яку підтримує Unity game engine. Вона надає розширені можливості для програмування інтерактивного ігрового процесу. За допомогою C# були створені скрипти, які контролюють різні аспекти гри, такі як рух персонажів, взаємодія з об'єктами, управління штучним інтелектом та багато іншого. Мова програмування C# дозволяє розробникам створювати складну логіку гри, використовуючи конструкції мови, такі як умовні оператори, цикли, функції та класи.

Один з переваг використання C# у Unity полягає в його інтеграції з іншими компонентами Unity, такими як компоненти графіки, фізики та анімації. Розробники можуть легко взаємодіяти з цими компонентами, використовуючи C# для керування їхньою поведінкою та параметрами. Крім того, C# надає можливість створювати власні класи та структури для організації коду гри. Це сприяє більшій модульності, повторному використанню коду та покращує скалірування проєкту. Мова C# також

підтримує розширення Unity через використання спеціальних API та бібліотек.

Завдяки мові програмування C# розроблений десктопний ігровий 2D застосунок отримав високу гнучкість, продуктивність та розширюваність, що дозволило здійснити повний контроль над грою та забезпечити багатофункціональний ігровий досвід для користувача.

У результаті, розроблений десктопний ігровий 2D застосунок з використанням Unity game надає користувачу захоплюючий ігровий процес з якісною графікою,механіками та плавною анімацією

.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ


1. Unity – найпопулярніший ігровий движок. URL : https://habr.com/ru/companies/pixonic/articles/577250/(дата звернення : 17.05.2023)
2. Короткий огляд мови C#. URL : https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/ (дата звернення 17.05.2023)
3. Рейтинг найпопулярніших ігрових движків. URL : https://itproger.com/ua/news/top-5-luchshih-igrovih-dvizhkov-reyting-2022-goda(18.05.2023)
4. Огляд ігрових движків. URL : https://logincasino.com/blog/obzor-igrovih-dvijkov76513.html (дата звернення 19.05.2023)
5. Огляд кращих мов програмування для ігр. URL : https://highload.today/put-v-gejmdev-10-luchshih-yazykov-dlya-sozdaniya-igr/ (дата звернення 21.05.2023)
6. Dave Calabrese. Unity 2D Game Development Packt. P: 2014. 126p.
7. Jared Halpern. Developing 2D Games with Unity: Independent Game Programming with C# by Halpern Apress L. P: 2018. 383p.

# ДОДАТКИ

# Додаток А – Реалізація класу Boar

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

public class Boar : Entity {
   public float speed = 6f;
   public Transform[] points;
   private int currentPointIndex = 0;
   private bool isMoving = false;
   public SpriteRenderer spriteRenderer;
   private Animator anim;
   private Collider2D boarCollider;
   private Rigidbody2D rb;

   private states state {
          get { return (states)anim.GetInteger("state"); }
          set { anim.SetInteger("state", (int)value); }
   }
   private void Awake() {
          lives = 3;
          anim = GetComponent<Animator>();
          spriteRenderer = GetComponentInChildren<SpriteRenderer>();
          boarCollider = GetComponent<Collider2D>();
          rb = GetComponent<Rigidbody2D>();
   }

   private void Update() {
     if (!isMoving) {
        if (currentPointIndex >= 0 && currentPointIndex < points.Length){
           Vector3 targetPosition = points[currentPointIndex].position;
           transform.position      =      Vector2.MoveTowards(transform.position,
targetPosition, speed * Time.fixedDeltaTime);

        if (Vector2.Distance(transform.position, targetPosition) < 0.2f) {
         isMoving = true;
         FlipSprite();
        }
      }
```

```
        }
        else {
            if (currentPointIndex > 0) currentPointIndex--;
            else currentPointIndex++;
            isMoving = false;
        }
        if (lives < 1)  {
                  boarCollider.enabled = false;
                  rb.isKinematic = true;
                  speed = 0;
                  StartCoroutine(DeathAnimation());
        }
      }

      private void FlipSprite() {
          Vector3 scale = transform.localScale;
          scale.x *= -1; // Изменяем знак масштаба по оси X для разворота спрайта
кабана
          transform.localScale = scale;
      }
      private IEnumerator DeathAnimation() {
           state = states.death; // Установка состояния смерти
           yield return new WaitForSeconds(1f); // Ожидани еокончания анимации
           Die();
      }
      public enum states {
            idle,
            death
      }
   }
```

# Додаток Б – Реалізація класу PauseMenu

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

public class PauseMenu : MonoBehaviour {
  public GameObject pausePanel;
  private bool isPaused = false;

  private void Update()  {
    if (Input.GetKeyDown(KeyCode.Escape)) {
      if (isPaused) ResumeGame();
      else PauseGame();
    }
  }

  public void PauseGame() {
    Time.timeScale = 0f;
    isPaused = true;
    pausePanel.SetActive(true);
  }

  public void ResumeGame()  {
    Time.timeScale = 1f;
    isPaused = false;
    pausePanel.SetActive(false);
  }

  public void RestartLevel()     {
    SceneManager.LoadScene(1);
    Time.timeScale = 1f;
  }

  public void LoadMainMenu() {
    SceneManager.LoadScene(0);
    Time.timeScale = 1f;
  }
}
```