Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп’ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему: «Дослідження особливостей технологій програмної реалізації екшн гри "Castle Transylvania"»

Виконав: студент групи К-20-2
Спеціальність 122 «Комп’ютерні науки»
Фаріон І. С.
Керівник: к.ф.-м.н., доц. Рудянова Т. М.
Рецензент: Університет митної справи та фінансів
(місце роботи)
доцент кафедри транспортних технологій та міжнародної логістики
(посада)
к.т.н., доц. Разгонов С. А.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Фаріон І.С. «Дослідження особливостей технологій програмної реалізації екшн гри "Castle transylvania"».

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2024.

У цій кваліфікаційній роботі був створений прототип екшн гри «Castle transylvania», що має велике значення в галузі геймдеву. Гра «Castle transylvania» є важливою, оскільки вона використовує унікальні підходи для екшн ігор. Проблема полягає в недостатньому розвитку українського геймдеву через обмежену кількість оригінальних концепцій. Більшість українських компаній створюють казуальні мобільні ігри, які не пропонують новаторських ідей та не привертають увагу геймерів з усього світу, не розвиваючи таким чином український ринок ігор.

Метою кваліфікаційної роботи є дослідження особливостей технологій програмної реалізації екшн ігор та розробка гри «Castle transylvania» з унікальними ігровими механіками.

Рішенням цієї проблеми є розробка проекту з унікальним геймплеєм та візуальним стилем. Унікальний геймплей передбачає розробку специфічного інтерфейсу та механік для такого типу ігор. Гра була розроблена з використанням движка Unity, який на 2024 рік є одним з найпопулярніших в Україні.

В результаті дослідження було розроблено практичні рекомендації щодо подальшого розвитку та вдосконалення гри, включаючи розширення ігрового контенту, вдосконалення існуючих механік та активну підтримку спільноти гравців.

Ключові слова: геймдев, екшн гра, ігровий персонаж, Unity, розробка, ігрова індустрія.

ANNOTATION

Farion I.S. "Investigation of the features of technologies of software implementation of the action game "Castle transylvania"".

Qualification work for obtaining a bachelor's degree in the specialty 122 "Computer Science". – University of Customs and Finance, Dnipro, 2024.

In this qualifying work, a prototype of the action game "Castle transylvania" was created, which is of great importance in the field of game development. Castle transylvania is important because it uses unique approaches to action games. The problem lies in the insufficient development of Ukrainian game development due to the limited number of original concepts. Most Ukrainian companies create casual mobile games that do not offer innovative ideas and do not attract the attention of gamers from around the world, thus not developing the Ukrainian game market.

The purpose of the qualification work is to study the features of technologies for the software implementation of action games and the development of the game "Castle transylvania" with unique game mechanics.

The solution to this problem is to develop a project with a unique gameplay and visual style. The unique gameplay involves the development of a specific interface and mechanics for this type of game. The game was developed using the Unity engine, which by 2024 is one of the most popular in Ukraine.

As a result of the study, practical recommendations were developed for further development and improvement of the game, including expansion of game content, improvement of existing mechanics and active support of the player community.

Keywords: gamedev, action game, game character, Unity, development, game industry.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ООП – Об’єктно-орієнтоване програмування.

GameDev – Game development (з англ. «Розробка ігор»).

Геймдев – GameDev.

Інді-студія – незалежна студія.

Геймплей – ігровий процес.

Контент – інформація, яка направлена на кінцевих користувачів чи аудиторію.

ПК – Персональний комп’ютер

ЗМІСТ

ВСТУП

Ігрова індустрія – це сфера, яка спеціалізується на створенні та розробці та впровадженні ігор. Вона збирає разом творчих фахівців – розробників, дизайнерів, художників, програмістів та багатьох інших, які спільно працюють над створенням цікавих та захоплюючих ігор для гравців. Ця сфера включає в себе різноманітні типи ігор, такі як комп'ютерні ігри, консольні ігри, мобільні ігри, онлайн-ігри та ігри для віртуальної реальності. Розробка відеоігор потребує поєднання креативності, технічних знань і спільної роботи команди.

Ігрова індустрія є надзвичайно популярною та має широку аудиторію гравців по всьому світу. Ігри стають засобом розваг, сприяють соціалізації та дозволяють втілювати фантазії. Вони бувають веселі, захоплюючи, навчальними або можуть викликати різноманітні емоції. Крім того, ігрова індустрія має значний економічний вплив. Вона створює робочі місця, приносить дохід від продажу ігор та послуг, і сприяє розвитку технологій. Багато компаній залучають інвестиції для створення нових ігор та інноваційних розробок. Ігрова індустрія постійно змінюється та розвивається, впроваджуючи нові технології, тренди та ідеї. Вона стала невід’ємною частиною сучасної культури та медіа і продовжує радувати гравців новими ігровими витворами.

Тема дипломної роботи «Дослідження особливостей технологій програмної реалізації екшн гри "Castle Transylvania"» є актуальною із декількох причин, наведемо їх. Це:

- стрімкий розвиток ігрової індустрії;
- популярність двигуна Unity;
- широка розповсюдженість та актуальність ігор;
- реалізація унікального інтерфейсу;
- геймплей та взаємодія.

З розвитком технологій ігри стали насправді актуальними в нашому світі. Вони допомагають зняти стрес, покращують настрій і дозволяють провести деякий час з насолодою, вчать приймати оптимальні рішення. Багато сучасних

ігор підтримують мультиплеєрний режим, що дає гравцям можливість спілкуватися, співпрацювати або змагатися один з одним.

Це відкриває широкі можливості для соціальної взаємодії та спілкування з іншими гравцями з усього світу. Крім того, деякі ігри сприяють розвитку розумових, логічних та стратегічних навичок. Індустрія ігор постійно розвивається та впроваджує нові технології, такі як віртуальна реальність, доповнена реальність, штучний інтелект та інші. Ігри стають платформою для впровадження та випробування новаторських ідей та технологій.

Створення унікального інтерфейсу є своєрідною інновацією та внеском у галузь ігрової індустрії, нових ідей та нового дихання у розумінні інтерфейсу користувача.

Гра розробляє унікальні сценарії гри, застосовуючи особливі механіки та спілкування між гравцями. Це формує неповторні враження та значно підвищує задоволення від ігрового процесу.

Метою кваліфікаційної роботи є дослідження особливостей технологій програмної реалізації екшн гри під назвою «Castle Transylvania», з унікальними ігровими механіками.

Основними завданнями кваліфікаційної роботи є:

- дослідження гейм-індустрії;
- аналіз існуючих ігор-конкурентів;
- визначення програмного забезпечення для розробки та обґрунтування обраних засобів реалізації гри;
- вибір та обґрунтування парадигми програмування для проектування;
- постановка технічного завдання;
- створення програми гри, її опис.

Об’єктом дослідження є: технології розробки екшн ігор.

Предметом дослідження є: розробка гри «Castle Transylvania», на ігровому двигуні Unity з об’єднанням жанрів екшн та roguelike.

У кваліфікаційній роботі були використанні такі методи дослідження: метод узагальнення, метод аналізу та синтезу, метод аналогії, описовий метод.

У результаті виконання кваліфікаційної роботи були підтвердженні наступні результати навчання, які відповідають освітній програмі 122 «Комп’ютерні науки»:

- Здатність до логічного мислення, побудови логічних висновків, використання формальних мов і моделей алгоритмічних обчислень, проектування, розроблення й аналізу алгоритмів, оцінювання їх ефективності та складності, розв’язності та нерозв’язності алгоритмічних проблем для адекватного моделювання предметних областей і створення програмних та інформаційних систем. Проектувати, розробляти та аналізувати алгоритми розв’язання обчислювальних та логічних задач, оцінювати ефективність та складність.
- Застосувати знання основних форм і законів абстрактно-логічного мислення, основ методології наукового пізнання, форм і методів вилучення, аналізу, обробки та синтезу інформації в предметній області комп’ютерних наук.
- Розробляти програмні моделі предметних середовищ, вибирати парадигму програмування з позицій зручності та якості застосування для реалізації методів та алгоритмів розв’язання задач в галузі комп’ютерних наук.
- Застосовувати знання методології та CASE-засобів проектування складних систем, методів структурного аналізу систем, об’єктно орієнтованої методології проектування при розробці і дослідженні.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку використаних джерел, що містить 41 джерел, додатку на 22 сторінках. Кваліфікаційна роботу містить 18 рисунків, загальна кількість сторінок – 53 сторінки.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз ринку ігор

Ринок ігор є одним з найдинамічніших і найприбутковіших сегментів індустрії розваг. Завдяки постійному технологічному прогресу та змінюючись споживчим уподобанням, ринок ігор демонструє значний ріст і різноманітність. За даними аналітичних компаній, глобальний ринок відеоігор у 2023 році оцінювався приблизно в 180 мільярдів доларів США. Очікується, що до 2026 року цей показник зросте до понад 250 мільярдів доларів, завдяки збільшенню кількості гравців, розвитку мобільних платформ та розширенню впливу нових технологій, таких як віртуальна та доповнена реальність.

Ринок ігор можна розділити на декілька основних сегментів: мобільні ігри, консольні ігри, ПК-ігри, ігри для VR/AR та кіберспорт. Найбільший сегмент за обсягом доходів включає ігри для смартфонів та планшетів. Консольні ігри орієнтовані на такі платформи, як PlayStation, Xbox та Nintendo, а ПК-ігри включають як високоякісні графічні проекти, так і інді-ігри. Ігри для VR/AR, що стають все більш популярними, пропонують новий рівень занурення в гру. Кіберспорт, організовані змагання з відеоігор, приваблює велику аудиторію та інвесторів. Мобільні ігри продовжують демонструвати найшвидші темпи росту завдяки доступності смартфонів та постійному покращенню їх технічних характеристик.

Технології віртуальної та доповненої реальності стають все більш доступними та популярними, пропонуючи новий рівень занурення в гру, що приваблює як розробників, так і гравців. Кіберспорт перетворився на глобальний феномен з багатомільйонними аудиторіями та значними грошовими призами, що сприяє розвитку нових ігрових жанрів та платформ для трансляцій, такі як Twitch та YouTube Gaming. Ігри з соціальними аспектами, де гравці можуть взаємодіяти один з одним у реальному часі, набувають все більшої популярності. Це включає в себе як великі мультиплеєрні онлайн ігри, так і ігри з кооперативним режимом.

До основних гравців ринку належать великі видавці та розробники, такі як Tencent, Sony Interactive Entertainment, Microsoft, Nintendo та Activision Blizzard. Tencent є найбільшою у світі ігровою компанією за доходами, відомою завдяки таким іграм, як Honor of Kings та PUBG Mobile. Sony Interactive Entertainment, видавець та виробник PlayStation, відома такими хітами, як The Last of Us та God of War. Microsoft, видавець та виробник Xbox, володіє франшизами Halo та Minecraft. Nintendo відома своїми унікальними ігровими серіями, такими як Mario, Zelda та Pokémon. Activision Blizzard є видавцем популярних серій, таких як Call of Duty та World of Warcraft.

Очікується, що ринок відеоігор продовжить своє зростання, підтримуючи нові технології та моделі монетизації, такі як підписки та хмарні ігри. Крім того, розвиток штучного інтелекту та машинного навчання може відкрити нові горизонти для створення більш інтерактивних та персоналізованих ігрових дослідів.

Ринок ігор знаходиться на піку свого розвитку, демонструючи стійке зростання та інновації. Для успішного функціонування на цьому ринку необхідно враховувати сучасні тенденції та інвестувати в нові технології, щоб залишатися конкурентоспроможними. Індустрія ігор впливає на широкий спектр інших галузей, включаючи апаратне забезпечення, програмне забезпечення, мережеві технології та навіть культуру і суспільство в цілому. Ігрові платформи продовжують інтегрувати соціальні мережі та сервіси стрімінгу, що сприяє подальшому зростанню взаємодії між гравцями та спостерігачами.

З огляду на тенденції зростання кількості мобільних пристроїв, поширення швидкісного інтернету та розвиток хмарних технологій, можна очікувати, що нові форми ігрових дослідів стануть доступними для ширшого кола користувачів. Це, у свою чергу, може призвести до зміни бізнес-моделей, де фокус буде зроблено на підписки, ігри як сервіс (GaaS) та інші способи монетизації, які пропонують стійкіший потік доходів порівняно з традиційними одноразовими покупками.

Розвиток штучного інтелекту та машинного навчання також відкриває нові можливості для індустрії. Вони можуть бути використані для створення більш реалістичних та адаптивних ігрових персонажів, персоналізованих сюжетів, що реагують на дії гравця, та навіть для автоматизації деяких аспектів розробки ігор. Такі інновації можуть значно підвищити якість ігрового досвіду та привабити нових гравців.

Крім того, індустрія ігор відіграє важливу роль у культурному обміні та формуванні сучасних суспільних тенденцій. Ігри стали не лише способом розваги, але й інструментом навчання, соціалізації та вираження культурних цінностей. Це робить ринок ігор важливим елементом глобальної економіки та культури, що продовжує еволюціонувати та адаптуватися до нових викликів і можливостей.

1.2 Опис розвитку Game Development

Розвиток Game Development, або розробки відеоігор, має багату і різноманітну історію, яка охоплює кілька десятиліть. Витоки відеоігор сягають 1950-х років, коли перші ігри були створені на комп'ютерах для наукових і військових досліджень [1]. Наприклад, гра OXO (1952), створена Олександром Дугласом для комп'ютера EDSAC, була симуляцією хрестиків-нуликів. Tennis for Two (1958), розроблена Вільямом Гігінботамом, вважається однією з перших ігор для розваг, яка використовувала аналоговий комп'ютер.

Наступний період знаменувався стрімким розвитком аркадних ігор. У 1972 році компанія Atari випустила Pong, яка стала надзвичайно популярною. Цей успіх відкрив дорогу для інших аркадних хітів, таких як Space Invaders (1978) і Pac-Man (1980) [2]. З'явилися перші домашні консолі, такі як Magnavox Odyssey і Atari 2600, що дозволило гравцям грати вдома.

З розвитком домашніх комп'ютерів (Commodore 64, ZX Spectrum, Amiga) і консолей (Nintendo Entertainment System (NES), Sega Genesis) відеоігри стали ще

доступнішими. У цей час з'явилися такі культові ігри, як Super Mario Bros (1985), The Legend of Zeld (1986) і Sonic the Hedgehog (1991) [3-4].

Перехід до тривимірної графіки в середині 1990-х років відкрив нові можливості для геймплею та дизайну. Doom (1993) та Quake (1996) від id Software стали основоположниками жанру шутерів від першої особи. Sony PlayStation, випущена в 1994 році, і Nintendo 64 (1996) визначили новий стандарт якості для консолей. Розробка ігор стала більш складною та затратною [5].

У 2000-х роках інтернет став важливою частиною геймплею. Масові багатокористувацькі онлайн-ігри (MMORPG) такі, як World of Warcraft (2004), здобули мільйони гравців по всьому світу. Консолі сьомого покоління (Xbox 360, PlayStation 3) стали мультимедійними центрами. Соціальні мережі та мобільні платформи дали поштовх до появи нових жанрів та бізнес-моделей, наприклад, free-to-play ігор [6].

Сучасний етап розвитку відеоігор характеризується розвитком віртуальної реальності (VR) і доповненої реальності (AR), зростанням популярності інді-ігор, розвитком стрімінгових сервісів (Twitch, YouTube Gaming) та платформ для цифрової дистрибуції (Steam, Epic Games Store) [7]. Ігрова індустрія стала однією з найприбутковіших галузей розваг у світі.

Сучасні ігри використовують передові технології, такі як трасування променів (ray tracing) для реалістичної графіки, штучний інтелект для складних NPC, та потужні фізичні движки для реалістичної симуляції світу [8]. Розробка ігор тепер вимагає співпраці великих команд, включаючи програмістів, художників, дизайнерів і тестувальників.

Також у той час мобільні ігри, хоча й існували вже давно, вийшли на новий рівень у 2008 році з появою першого магазину додатків (App Store). Кожен власник смартфона змінив свої звички, почавши активно завантажувати мобільні додатки [9]. Найпопулярнішою грою у 2023 році стала Honkai Star Rail, випущена у тому році. Китайська компанія miHoYo, яка була заснована у 2012 році, стала відомою своїми успішними проектами у жанрах як екшн-RPG, так і кардколекційних ігор. Одним з їх найвідоміших творінь є Genshin Impact, яка

стала популярною як на азіатських, так і на західному ринках. В 2023 році miHoYo випустили ще один глобальний хіт – Honkai Star Rail, який став ще більшим успіхом і отримав численні нагороди. Компанія відома своєю увагою до деталей у геймдизайні, відмінною графікою та вражаючою масштабністю своїх ігор. miHoYo активно розвивається як міжнародна компанія, зосереджуючись на тому, щоб створювати ігри, які привертають гравців з усього світу [10].

З появою Інтернету з'явилися численні ігри, які стали надзвичайно популярними і зуміли залучити мільйони гравців по всьому світу. Серед них можна виділити кілька категорій та окремих проектів, що стали справжніми феноменами своєї епохи. Браузерні ігри, такі як Neopets (1999), одна з перших браузерних ігор, яка дозволяла гравцям доглядати за віртуальними домашніми тваринами, виконувати завдання та брати участь у міні-іграх, швидко стала популярною серед дітей та підлітків. RuneScape (2001), багатокористувацька рольова онлайн-гра, дозволяла гравцям досліджувати відкритий світ, виконувати завдання та боротися з монстрами. Завдяки своїй доступності та багатогранності, RuneScape привабила велику кількість гравців [11].

Таким чином, історія розробки відеоігор – це історія інновацій та креативності, яка перетворилася з невеликої ніші на глобальну індустрію, що продовжує розвиватися і впливати на сучасну культуру.

Якщо говорити про ситуацію в Україні, то згідно з даними компанії UNIT.City на 2023 рік, найпопулярнішою платформою для розробки ігор є Unity – 69%. Щодо жанрів, розробники найчастіше вибирають Action – 40% і Adventure – 42%. Щодо мобільних ігор, тут популярність платформ представлена так: Android (82%) та iOS (79%) [12].

Сьогодні Україна є домом для численних ігрових студій, від великих компаній до незалежних інді-розробників. Київ, Харків, Львів та Одеса є головними центрами GameDev в країні. Українські розробники активно працюють у різних жанрах, від мобільних ігор до великих ААА проектів. Однією з останніх значних подій стало оголошення GSC Game World про розробку

"S.T.A.L.K.E.R. 2", що стало очікуваною подією для шанувальників серії. Інші успішні проекти включають "Sherlock Holmes" від Frogwares, "Metro Exodus" від 4A Games та "Cossacks 3" від GSC Game World. Зокрема, з початком повномасштабного вторгнення, українські розробники активно підтримують ЗСУ.

У список провідних компаній в Україні на 2023 рік увійшли такі компанії, як Ubisoft, Playrix, Plarium та Room 8 Group [13]. У таких студіях команда в основному розділяється на три групи: технічні спеціалісти, художники та ідеологи проекту [14].

Картина GameDev спеціалізацій можна побачити на рисунку 1.1:

Рисунок 1.1 – Картина геймдеву

Перша група складається з програмістів, веб-розробників, розробників для мобільних пристроїв, системних адміністраторів, адміністраторів баз даних та інших фахівців. Це найбільш високооплачувана частина команди, яка відповідає за всю технічну складову гри.

Друга група об'єднує фахівців, які відповідають за візуальну складову гри: 3D-моделерів, аніматорів, ілюстраторів та дизайнерів. Початковий внесок дизайнерів у процес розробки набагато важливіший, ніж у більшості інших учасників команди.

Третя група складається з ідеологів, які розробляють концепцію і керують усією розробкою: гейм-дизайнерів та продюсерів.

1.3 Розвиток інді-студій та самостійної розробки

Розвиток інді-студій та самостійної розробки в ігровій індустрії став об'єктом уваги для багатьох дослідників та практиків у сучасному світі. Цей процес відображає трансформацію ігрової індустрії та розширення можливостей для творчості та підприємництва.

Перш за все, слід зазначити, що інді-розробники стали значним гравцем на ринку ігор завдяки доступності інструментів розробки, таких як Unity, Unreal Engine, а також розвитку цифрових платформ поширення, які дозволяють незалежним розробникам легко розповсюджувати свої продукти.

Одним із ключових факторів успіху інді-студій є їхня здатність створювати унікальний та оригінальний контент, який відрізняється від традиційних ААА-проектів. Це дозволяє їм привертати увагу гравців та займати власну нішу на ринку.

Напрямок самостійної розробки також набуває все більшої популярності, особливо серед інді-розробників. Багато стартапів та невеликих студій вибирають шлях самостійної розробки для збереження творчої волі та незалежності від великих видавництв.

У цілому, розвиток інді-студій та самостійної розробки відкриває нові можливості для інновацій та креативності в ігровій індустрії. Він сприяє розмаїттю контенту, підвищенню конкуренції та забезпечує гравцям доступ до різноманітних ігор, які задовольняють різноманітні смаки та вподобання.

Також поява платформи Steam, яка забезпечила можливість поширювати ігри та інший контент незалежно від великих видавництв. Це призвело до того, що у 2017 році був запроваджений Steam Direct, який спростив процес виходу ігор на платформу [15].

Також запуск Microsoft літньої промоакції у 2008 році, під час якої були представлені ігри Braid та Super Meat Boy [16]. Це було першим випадком

підтримки інді-розробників від цілої консольної платформи і мала велике значення для інді-сегменту.

Важливу роль у розвитку інді-індустрії в Україні відіграють локальні та міжнародні події, такі як GameDev Conference (GDC) в Києві, Lviv GameDev Conference та інші зустрічі розробників. Ці заходи дозволяють розробникам обмінюватися досвідом, знаходити партнерів і інвесторів, а також презентувати свої проекти широкій аудиторії. Інді-розробники активно використовують платформи краудфандингу, такі як Kickstarter і Indiegogo, для фінансування своїх проектів. Це дозволяє їм зберігати творчу незалежність та отримувати фінансування без посередників [17]. Цифрові платформи, такі як Steam, GOG, itch.io, надають можливість публікувати ігри безпосередньо, досягаючи глобальної аудиторії.

Незважаючи на успіхи, інді-розробники в Україні стикаються з численними викликами, такими як обмежене фінансування, конкуренція на глобальному ринку та технічні труднощі. Однак, ентузіазм та креативність українських розробників дозволяють їм створювати унікальні та цікаві проекти, що знаходять своїх шанувальників у всьому світі. Розвиток інді-студій та самостійної розробки відеоігор в Україні є важливою частиною національної ігрової індустрії. Незалежні розробники та інді-студії демонструють високу креативність та інноваційний підхід, створюючи ігри, що здобувають визнання як на локальному, так і на міжнародному рівні. З розвитком технологій та підтримкою спільноти, українські інді-ігри мають всі шанси продовжувати своє зростання та впливати на глобальну індустрію відеоігор.

Також є і повністю експериментальні та унікальні рішення. Жанр Roguelike, на 2023 рік, є дуже популярним в середовищі інді-розробки. Сам жанр можна описати наступними термінами:

- випадкова або процедурна генерація рівнів;
- перманентна смерть (тобто, якщо гравець помер, то перевантаження рівня неможливе);
- гра повинна показувати протистояння гравця з середовищем.

За останнє десятиліття термін «інді» перестав асоціюватися з простими іграми і тепер є синонімом винахідливості в ігровому дизайні. Консолі останнього покоління активно підтримують такі студії, визнаючи їх потенціал, і навіть існують випадки придбання інді-компаній великими видавцями. Наприклад, великі компанії, такі як Devolver Digital, пропонують свої ресурси для видавництва ігор інді-розробників. Їхній слоган – «Devolver Digital рекомендує тільки найвишуканіші відеоігри для витонченого геймера та його витонченого смаку» [18]. Крім видавців, до співпраці з інді-командами залучаються також розробники ігрових двигунів.

Станом на 2023 рік можна легко завантажити ігровий двигун комерційного рівня і почати з ним працювати. Проте, незважаючи на всі ці переваги, існують і недоліки інді-розробки [19]. Наприклад, взаємодія з аудиторією та її залучення залишається великою проблемою для більшості інді-студій, оскільки їм важко донести свої проекти до цільової аудиторії по всьому світу.

1.4 Сучасний стан екшн-рогалік ігор у GameDev

Сучасний стан екшн рогалік ігор у GameDev характеризується зростаючою популярністю цього жанру, його різноманітністю та інноваційністю. Екшн рогалік ігри, або roguelike-екшни, поєднують інтенсивний бойовий геймплей з елементами процедурної генерації рівнів, постійною смертю персонажів та їх повторним народженням, а також часто містять елементи RPG. Roguelike-екшни набули великої популярності серед гравців завдяки своїй високій повторюваності та виклику, який вони пропонують. Кожна нова сесія гри відрізняється від попередньої завдяки випадковій генерації, що робить кожен забіг унікальним. Постійна смерть персонажів та необхідність починати знову з самого початку стимулює гравців покращувати свої навички та стратегії.

Останніми роками кілька ігор цього жанру здобули велику популярність та визнання серед критиків. Однією з найбільш успішних у жанрі стала Hades від Supergiant Games, яка поєднує швидкий бойовий геймплей з глибоким сюжетом

та чудовою графікою. Гравці керують Загреєм, сином Аїда, і намагаються втекти з підземного світу, долаючи численних ворогів та босів. Інша помітна гра - Dead Cells від Motion Twin, яка пропонує динамічний бойовий геймплей з елементами метроідванії, де гравці досліджують великий, процедурно згенерований світ, збираючи ресурси та вдосконалюючи свого персонажа.

Сучасні roguelike-екшни постійно вдосконалюються, включаючи нові механіки та інновації. Основними елементами залишаються процедурна генерація, постійна смерть та розвиток персонажа через збирання предметів та поліпшення здібностей. Водночас, розробники експериментують з новими підходами, такими як розгалужені сюжети та глибокі персонажі, як у Hades, де сюжет та персонажі розвиваються через численні проходження, додаючи глибини та емоційного залучення. Інтеграція багатокористувацьких режимів, як у Risk of Rain 2, демонструє, як кооперативний геймплей може бути ефективно впроваджений у roguelike-екшни, дозволяючи гравцям співпрацювати для досягнення спільної мети. Висока якість візуального оформлення та анімації, як у Dead Cells та Hades, робить ігри привабливими для ширшої аудиторії [20-21].

Попри успіхи, розробники roguelike-екшнів стикаються з низкою викликів. Збереження балансу між складністю та задоволенням від гри, створення нових механік, що не будуть повторювати вже існуючі, та підтримка інтересу гравців через численні проходження – це лише деякі з них. Проте, з огляду на постійний інтерес до жанру та творчий підхід розробників, roguelike-екшни мають усі шанси на подальший розвиток і популярність.

Сучасний стан екшн рогалік ігор у GameDev характеризується високою популярністю, різноманітністю підходів та інноваційними механіками. Цей жанр продовжує розвиватися, пропонуючи гравцям унікальні та захоплюючі ігрові досвіди. Зі зростанням технологічних можливостей та творчого потенціалу розробників, екшн рогалік ігри мають великі перспективи на майбутнє в індустрії відеоігор.

1.5 Аналіз існуючих конкурентів екшн-ігор

Найпопулярнішою асоціацією може бути, гра Vampire Survivors. Створена компанією Poncle, вона завоювала серця гравців своєю унікальною сумішшю екшну та roguelike елементів [22]. У цій грі ви опиняєтеся в жорстокому світі, де кожен куток загрожує вампірами та іншими небезпечними загрозами. Ваша мета - вижити якнайбільше часу, протистояючи безперервним хвилям ворогів та набираючи ресурси для поліпшення вашого обладнання та навичок.

Гра вражає різноманіттям персонажів, кожен з яких має свої унікальні навички та можливості. Це не лише додає грі глибину, але й робить кожен проходження унікальним. Крім того, система прокачування дає можливість постійно поліпшувати вашого персонажа, збираючи ресурси та отримуючи нові предмети і здібності.

Гра була реалізована на двигуні Unity, що забезпечує відмінну якість графіки та геймплею. Її реліз відбувся 20 жовтня 2022 року, і вона доступна для гравців на платформі Windows.

Рисунок 1.2 – Інтерфейс гри Vampire Survivors

Як видно з рисунку 1.2, гра має деталізовані моделі та реалістичні анімації, які створюють атмосферу загрози та напруженості. Також в грі присутні різноманітні локації, такі як божевільний ліс, бібліотека, вежа Галло та гора Луночар. Звуковий дизайн також додає настрою та іммерсії, підкреслюючи

динаміку гри та події на екрані. В цілому, Vampire Survivors відповідає поточним тенденціям на ринку ігор та має потенціал здобути велику популярність серед широкого кола гравців.

Іншою, менш популярною асоціацією, може стати Brotato [23]. Це також інді-гра в жанрі екшн та roguelike, але на відміну від попередньої гри, вона має інший графічний стиль і атмосферу, а персонажі тут називаються «картоплями». Гравець керує картопляним персонажем, озброєним різноманітною зброєю, та бореться з хвилями ворогів на обмежених аренах. Основна механіка гри полягає в управлінні персонажем з виглядом зверху, де гравець самостійно керує стрільбою та напрямком атаки. Гравець має можливість вибирати з широкого асортименту зброї, включаючи вогнепальну зброю, холодну зброю та вибухові пристрої. Кожна зброя має свої унікальні характеристики та способи атаки, що дозволяє гравцю комбінувати їх для створення унікальних бойових стилів.

Локації в Brotato здебільшого аркадні та прості за структурою, часто представляючи собою відкриті арени, де відбуваються битви з хвилями ворогів. Кожна арена має свої особливості та виклики, що додає різноманіття в ігровий процес. Деякі арени можуть містити перешкоди або об'єкти, які гравець може використовувати для тактичної переваги.

Саундтрек гри складається з веселих та енергійних мелодій, які доповнюють загальну атмосферу. Brotato пропонує гравцям захоплюючий досвід, поєднуючи динамічні битви з яскравим і гумористичним стилем. Гра цікава для широкого кола гравців завдяки своїй доступності та розважальності.

Видно з рисунку 1.3, що графіка гри яскрава та кольорова, з мультяшним стилем, що надає грі веселого та легковажного тону. Художнє оформлення персонажів, ворогів та локацій виконано в комічному стилі, що підкреслює гумористичний характер гри. Анімації рухів та атак зроблені плавно і виразно, що додає динаміки та захопливості геймплею. Brotato має менш серйозний і більш гумористичний підхід до тематики виживання. Гра орієнтується на швидкий та динамічний екшн, поєднаний з гумористичними елементами.

Рисунок 1.3 – Інтерфейс гри Brotato

Також варто згадати гру Children of Morta, у якій приймали участь українські розробники [24]. Це незвичайна інді-гра, яка поєднує в собі елементи RPG та roguelike. Гравець відправляється в епічну подорож разом з родиною Бергсонів - героями, які об'єднуються, щоб протистояти зловісним силам, що загрожують їхньому світу. Керуючи різними членами родини, кожен з яких має свої унікальні навички та стиль гри, гравець вирушає в пригоди, де бореться з ворогами, знаходить скарби та вдосконалює свої навички. Центральним елементом Children of Morta є його глибока історія, яка розкривається через міжрівневі анімації та розповіді, що розкривають таємниці давніх легенд і відносин між членами родини.

Саундтрек Children of Morta додає глибину та емоційність до геймплею, створюючи настрій кожної сцени та події. Звукові ефекти підкреслюють бойові сцени та додають напруження до кожного етапу пригод.

Гра славиться своїм прекрасним піксельним художнім оформленням, яке створює дивовижний світ, наповнений магією та загрозами. Кожен персонаж, монстр та локація має унікальний дизайн, що створює неповторну атмосферу гри. Як видно з рисунку 1.4, інтерфейс досить інтуїтивний та збалансований. Головним елементом є, звичайно ж, графічне відображення гри, де відбуваються основні дії.

Рисунок 1.4 – Інтерфейс гри Children of Morta

Не дуже популярною грою є Death Must Die [25]. Гра представляє собою інді-гру в жанрі roguelike з елементами екшну. Гравець керує персонажем, який досліджує випадково згенеровані рівні, бореться з ворогами та збирає ресурси. Основні елементи геймплею включають бойову систему, де гравці використовують різноманітну зброю та здібності для боротьби з ворогами. Комбінація ближнього та дальнього бою, а також спеціальні здібності персонажа додають глибини бойовій механікі. Система прогресії дозволяє гравцям покращувати свого персонажа, збираючи ресурси та знаходячи нові предмети, що підвищують їхні здібності.

Розвиток гравців та вибір стратегій є важливим аспектом гри. Гравці можуть розвивати свого персонажа, використовуючи різні стратегії залежно від доступних ресурсів та ігрових ситуацій. Важливі аспекти розвитку включають навички та здібності, які гравці можуть обирати для адаптації персонажа під свій стиль гри, а також різноманітність зброї та предметів, що відкриває нові стратегії та підходи до бою.

Динаміка гри та відновлення інтересу гравців можуть бути забезпечені кількома методами. Регулярні оновлення з новим контентом, таким як нові рівні, вороги та предмети, підтримують інтерес гравців. Крім того, наявність різних

ігрових режимів, таких як сюжетний режим та режим виживання, надає додаткові виклики та можливості для гравців.

Активна спільнота гравців може значно вплинути на розвиток гри. Форумі, соціальні мережі та платформи для обміну відео стають місцем для обговорення стратегій, обміну досвідом та надання зворотного зв'язку розробникам.

Технічна сторона гри включає оптимізацію продуктивності, важливу для плавної роботи на різних пристроях, регулярне виправлення помилок для поліпшення ігрового досвіду та підтримку різних платформ, таких як ПК, консолі та мобільні пристрої.

Візуальний стиль та графіка в Death Must Die відрізняються унікальним підходом, який може поєднувати ретро-графіку з сучасними елементами. Атмосфера гри часто підкреслюється використанням темних кольорів та грізних образів ворогів, що створює напружену та захоплюючу атмосферу. Під час гри можуть з'являтися діалогові вікна, що повідомляють про сюжетні події, взаємодію з NPC або вибір діалогових опцій (рисунок 1.5). Вікна діалогів зазвичай розташовані в нижній частині екрану, щоб не закривати основне ігрове поле. Короткі повідомлення можуть з'являтися в центрі екрану для інформування гравця про важливі події чи зміни в грі. Ці елементи інтерфейсу допомагають гравцям легко орієнтуватися в грі, приймати стратегічні рішення та насолоджуватися ігровим процесом, створюючи зручне та привабливе середовище для гравців.

Рисунок. 1.5 – Діалогове вікно гри Death Must Die

1.6 Висновки до першого розділу

Працюючи з великими компаніями, розробники ігор отримують не лише фінансову підтримку, але й доступ до передових технологій та професійних знань. Великі студії, такі як Electronic Arts, Ubisoft, CD Projekt, активно інвестують в інновації та розробку нових технологій, що сприяє загальному прогресу в галузі. Однак, сміливі інді-студії зробили великий внесок у розвиток ігрової індустрії, демонструючи, що для створення успішних ігор не обов'язково мати величезні бюджети – важливими є лише креативність, час і терпіння. Інді-розробники, такі як Supergiant Games, які створили Hades, або Team Cherry, відомі завдяки Hollow Knight, показують, що інноваційні ігрові концепції та високий рівень виконання можуть досягати великого успіху.

Аналізуючи ігрову індустрію в Україні, можна побачити, що вона швидко зростає та має значний економічний вплив. З кожним роком все більше українських студій отримують міжнародне визнання та успіх. Індустрія приваблює іноземних інвесторів та сприяє розвитку економіки країни. Наприклад, такі компанії, як GSC Game World, творці серії ігор S.T.A.L.K.E.R., та 4A Games, відомі своєю серією Metro, демонструють високий рівень

професіоналізму та конкурентоспроможності на глобальному ринку. Сьогодні українські студії розробки ігор отримують визнання на світовому ринку, створюючи високоякісні продукти, що конкурують з роботами відомих міжнародних компаній. Це сприяє підвищенню репутації України як центру інновацій та креативності в ігровій індустрії.

Таким чином, українська ігрова індустрія має великий потенціал для подальшого розвитку та зростання. Залучення нових інвестицій, розвиток інноваційних технологій та підвищення рівня освіти в галузі програмування та ігрового дизайну сприятимуть зміцненню позицій України на світовому ринку ігор та створенню нових можливостей для талановитих розробників. Зростаюча популярність освітніх програм з розробки ігор, підтримка стартапів та створення спеціалізованих шкіл і акселераторів допоможуть формувати нове покоління професіоналів, готових вирішувати складні завдання та створювати інноваційні проекти. Україна може стати важливим гравцем у світовій кіберспортивній індустрії. Все це разом створює сприятливі умови для подальшого розширення та зміцнення позицій української ігрової індустрії на світовій арені.

Розвиток кіберспорту в Україні також значно впливає на місцеву економіку та культуру. Відкриття кіберспортивних арен, проведення міжнародних турнірів та участь українських команд у світових змаганнях підвищують престиж країни та стимулюють інтерес до технологій і спорту серед молоді. Відомі українські кіберспортсмени та команди, такі як Natus Vincere (Na'Vi), стають прикладом для наслідування та мотивують молодь займатися кіберспортом.

Підсумовуючи, українська ігрова індустрія знаходиться на підйомі та має всі передумови для подальшого успіху. Підтримка держави, залучення інвестицій, розвиток освітніх програм та інфраструктури – це ключові фактори, які сприятимуть зміцненню позицій України на світовому ринку ігор та кіберспорту. Українські розробники вже показали свою спроможність створювати високоякісні продукти та досягати великих успіхів, і цей тренд, без сумніву, продовжуватиметься.

РОЗДІЛ 2 АНАЛІЗ ГОТОВИХ РІШЕНЬ ТА ЗАСОБІВ РОЗРОБКИ ІГОР

2.1 Основні етапи проектування гри

Розглянемо складові частини розробка екшн-гри з елементами roguelike, які можна розділити на наступні етапи:

- розробка інтерфейсу;
- розробка локацій гри;
- розробка механік гри.

Розробка інтерфейсу є важливим аспектом створення будь-якого програмного забезпечення, включаючи екшн-гру з елементами roguelike. Інтерфейс користувача виступає як посередник між гравцем і грою, забезпечуючи зручний та інтуїтивно зрозумілий спосіб взаємодії з ігровим середовищем.

Розробка локацій є одним із ключових аспектів, що визначають глибину, атмосферу та ігровий досвід. Локації повинні бути різноманітними, цікавими та викликати бажання гравця досліджувати кожен куточок ігрового світу. Перш за все, слід розробити концепцію кожної локації, що включає тему, атмосферу, кольорову палітру та загальний настрій. Створення початкових нарисів та ескізів допомагає візуалізувати ідеї та узгодити бачення між членами команди. Розміщення об'єктів та ворогів повинно бути стратегічним, щоб забезпечити цікавий і збалансований ігровий процес. Таким чином, розробка локацій є багатогранним процесом, що включає концептуальне планування, архітектурне проектування, візуалізацію та тестування. Успішно розроблені локації значно підвищують загальний ігровий досвід, роблячи гру захоплюючою та привабливою для гравців.

Розробка механік гри є важливою складовою, яка визначає, як саме гравці взаємодіятимуть з ігровим світом та яким буде їхній ігровий досвід. Механіки гри повинні бути ретельно сплановані, щоб забезпечити цікавий, збалансований та захоплюючий ігровий процес. Основні механіки включають бойову систему,

яка повинна мати різноманітні атаки. Важливо розробити різні типи зброї та вміння, які гравець може використовувати під час бою. Прогресія та розвиток персонажа реалізується через систему рівнів, де гравці можуть покращувати свої навички та здібності, здобуваючи досвід за перемогу над ворогами. Процедурна генерація контенту забезпечує унікальний досвід для кожного проходження гри, роблячи її непередбачуваною та цікавою. Розробка механік є складним та багатоетапним процесом, який вимагає ретельного планування, тестування та оптимізації. Успішно реалізовані механіки гри здатні забезпечити захоплюючий та непередбачуваний ігровий досвід, роблячи гру цікавою та привабливою для широкого кола гравців.

2.2 Узагальнення та аналіз інформації про C#

C# (C-Sharp) є сучасною мовою програмування, розробленою компанією Microsoft як частина платформи .NET. З моменту свого впровадження в 2000 році, C# став однією з найбільш популярних мов програмування завдяки своїй простоті, універсальності та потужності [26]. Основні характеристики C# включають підтримку об’єктно-орієнтованого програмування (ООП), що дозволяє розробникам створювати ефективні та масштабовані рішення, уникати дублювання коду через використання класів і об'єктів, а також застосовувати принципи інкапсуляції та поліморфізму. Це полегшує розробку складних систем, де різні частини гри можуть взаємодіяти одна з одною у структурованій та передбачуваній манері. Крім того, C# є строго типізованою мовою, що дозволяє уникати багатьох помилок на етапі компіляції, забезпечуючи більш надійний код. Мова постійно оновлюється і включає в себе сучасні функції, такі як асинхронне програмування (async/await), властивості автоматичного реалізування, лямбда-вирази, мовні інтегровані запити (LINQ) та багато інших. Завдяки тісній інтеграції з платформою .NET, C# надає великий набір бібліотек та фреймворків для розробки веб-додатків, десктопних додатків, мобільних додатків та ігор. Крім того, завдяки проекту .NET Core, C# тепер підтримує

розробку під Windows, Linux та macOS, що значно розширює його можливості [27].

C# використовується в різних сферах програмного забезпечення. Однією з ключових областей застосування є розробка веб-додатків за допомогою ASP.NET, який дозволяє створювати масштабовані та високопродуктивні веб-додатки та API. ASP.NET надає багатий набір інструментів і бібліотек для роботи з базами даних, автентифікацією користувачів, обробкою запитів та іншими важливими аспектами веб-розробки. Розробка десктопних додатків також є важливою сферою використання C#. Windows Forms та Windows Presentation Foundation (WPF) дозволяють створювати інтерактивні та зручні інтерфейси користувача для настільних додатків. Windows Forms є простішим у використанні і підходить для створення швидких прототипів, тоді як WPF надає більше можливостей для створення складних та естетично привабливих інтерфейсів [28].

У мобільній розробці C# використовується завдяки Xamarin, що дозволяє створювати кросплатформенні мобільні додатки для iOS та Android. Xamarin дозволяє використовувати єдиний кодовий базис для створення додатків для різних платформ, що значно зменшує час та витрати на розробку. В ігровій індустрії C# знайшов широке застосування завдяки Unity, одному з найпопулярніших ігрових двигунів, який використовує C# як основну мову для скриптів. Unity дозволяє створювати як прості, так і складні ігри для різних платформ, включаючи ПК, консолі та мобільні пристрої [29].

Хмарні сервіси також є важливою сферою застосування C#. Завдяки тісній інтеграції з Microsoft Azure, C# широко використовується для створення хмарних додатків і сервісів. Azure надає широкий спектр інструментів та сервісів для роботи з даними, штучним інтелектом, безпекою та іншими аспектами хмарних обчислень, що робить його ідеальною платформою для розвитку хмарних рішень. Крім того, можливості для автоматизації та DevOps-підходів значно спрощують роботу над складними проектами [30].

Серед переваг C# можна виділити зручність і простоту. Синтаксис C# є інтуїтивно зрозумілим і схожим на інші популярні мови, такі як Java та C++, що спрощує процес навчання. Потужність та продуктивність C# дозволяє створювати високопродуктивні додатки, а широка екосистема надає велику кількість бібліотек, фреймворків та інструментів, що робить розробку ефективною та продуктивною. Постійні оновлення та покращення від Microsoft забезпечують актуальність та передові можливості мови. Крім того, C# має велику спільноту розробників, що забезпечує підтримку та доступ до численних ресурсів, таких як документація, форуми, навчальні матеріали та приклади коду.

Однак є й недоліки: залежність від платформи .NET може бути обмежуючим фактором, особливо для розробників, які працюють в гетерогенних середовищах або з іншими технологіями. Деякі аспекти C# можуть вимагати більше ресурсів порівняно з іншими мовами, особливо в умовах обмежених ресурсів. Складність у великих проектах може бути значною без належного управління кодом та архітектурою, що може призводити до важких у підтримці та розвитку кодових баз. У великих проектах необхідно застосовувати суворі практики управління проектами, такі як модульне тестування, безперервна інтеграція та розгортання (CI/CD), а також ефективна документація та комунікація між членами команди [31].

Незважаючи на ці недоліки, C# залишається потужним інструментом для розробників завдяки своїм численним перевагам та широкому спектру застосувань. C# є потужною, сучасною та універсальною мовою програмування, яка підходить для широкого спектру завдань – від веб-розробки до ігрової індустрії. Її підтримка об’єктно-орієнтованого програмування, інтеграція з платформою .NET, багатофункціональність та постійне оновлення роблять C# відмінним вибором для розробників, які прагнуть створювати надійні та масштабовані програмні рішення. Незважаючи на деякі недоліки, переваги C# значно переважають, що забезпечує її популярність та затребуваність у сучасному світі програмування. Розробка на C# продовжує розвиватися завдяки інноваціям і підтримці спільноти, що робить її одним з ключових інструментів у

арсеналі сучасного програміста. З розширенням екосистеми .NET і постійними вдосконаленням у сфері продуктивності та безпеки, C# залишається актуальною і перспективною мовою для розробників будь-якого рівня. Вона дозволяє реалізувати як прості, так і високонавантажені системи, забезпечуючи при цьому високу продуктивність і надійність роботи додатків.

Одним із найважливіших аспектів, що сприяє популярності C#, є активна підтримка та розвиток з боку Microsoft. Компанія постійно вдосконалює мову, додаючи нові функції та можливості, що робить її актуальною в умовах швидко змінюваних технологій. Зокрема, останні версії C# включають такі можливості, як розширені вирази, нові типи даних, поліпшення в області обробки помилок і продуктивності, а також нові інструменти для розробників.

Інша важлива перевага C# – це велика кількість доступних інструментів та середовищ розробки. Серед них найпопулярнішим є Visual Studio – потужне інтегроване середовище розробки (IDE), яке надає розробникам всі необхідні інструменти для написання, налагодження та розгортання додатків на C#. Visual Studio підтримує широкий спектр мов програмування та платформ, що робить його універсальним інструментом для розробників.

Крім того, C# забезпечує високу продуктивність роботи додатків завдяки використанню сучасних технологій і методів оптимізації. Мова підтримує багато потоковість, що дозволяє ефективно використовувати ресурси сучасних багатоядерних процесорів. Також, завдяки підтримці асинхронного програмування, C# дозволяє створювати високопродуктивні додатки, які можуть обробляти великий обсяг запитів та працювати з великою кількістю користувачів.

Оскільки технології постійно змінюються, розробникам важливо постійно вдосконалювати свої знання та навички. Мова програмування C# відкриває широкі можливості для розробників, надаючи інструменти та ресурси для створення інноваційних та високопродуктивних програмних продуктів.

2.3 Вибір програмних засобів для створення проекту та аналіз Unity

Для створення проекту було обрано Unity як ігровий двигун і мову програмування C#. Весь код буде компілюватися в інтегрованому середовищі розробки Microsoft Visual Studio. Це рішення має кілька переваг:

- Спільнота розробників: Unity має велику спільноту, де можна знайти багато інформації та документації, а також отримати допомогу у вирішенні проблем.
- Кроссплатформеність: Unity дозволяє розробляти ігри для різних платформ, що забезпечує можливість портів проекту, наприклад, на мобільні пристрої.
- Широкий функціонал: Unity має безліч вбудованих функцій, які скорочують час розробки ігор.
- Інтерфейс: Unity пропонує дружній та інтуїтивно зрозумілий інтерфейс, що спрощує розробку ігор навіть для початківців. Він також забезпечує зручні інструменти для створення графіки, фізики та інших аспектів гри.
- Можливості розширення: Unity має власний магазин ассетів, де можна придбати готові рішення від інших розробників або додати готові моделі. Крім того, Unity підтримує модифікацію самого двигуна, що дозволяє досвідченим розробникам створювати інструменти для своїх потреб, які будуть виконуватися безпосередньо в середовищі Unity без необхідності компіляції коду.

Unity, окрім C#, раніше використовувала такі мови, як Boo (діалект Python) і Unity Script (модифікація JavaScript) [32-33]. Важливо зазначити, що кожен скрипт за замовчуванням успадковує клас MonoBehavior. Це базовий клас, який дозволяє додавати до будь-якого об'єкта на сцені класи, що від нього успадковані. Завдяки MonoBehavior можна використовувати базові типи, такі як Transform, у своїх скриптах, а також викликати основні методи, такі як Update (викликається на кожному кадрі), Start (викликається один раз перед Update), OnCollisionEnter (викликається при зіткненні об'єкта з іншим об'єктом) та інші [34]. Однак, незважаючи на зручність цього підходу, він має свої недоліки. Використання такого рішення не забезпечує належної архітектури, не надає

інструментів для управління залежностями, а пошук об'єктів на сцені значно впливає на продуктивність.

Проект в Unity ділиться на сцени, кожна з яких є файлом зі своїм набором об'єктів. Ці об'єкти можуть бути як звичайними, так і "порожніми". Кожен об'єкт має свій набір компонентів, з якими можуть взаємодіяти скрипти. Як видно з рисунку 2.1, у кожного об'єкта, незалежно від того, порожній він чи ні, є компонент Transform, який визначає координати об'єкта на сцені, його поворот та розмір. Редактор також підтримує успадкування об'єктів: усі дочірні об'єкти повторюють зміни повороту, розміру та переміщення батьківського об'єкта. Двигун підтримує більшість популярних форматів музики, звуків, текстур тощо. Крім того, в Unity є Unity Asset Server – інструмент для спільної розробки, який додає контроль версій та інші серверні функції.

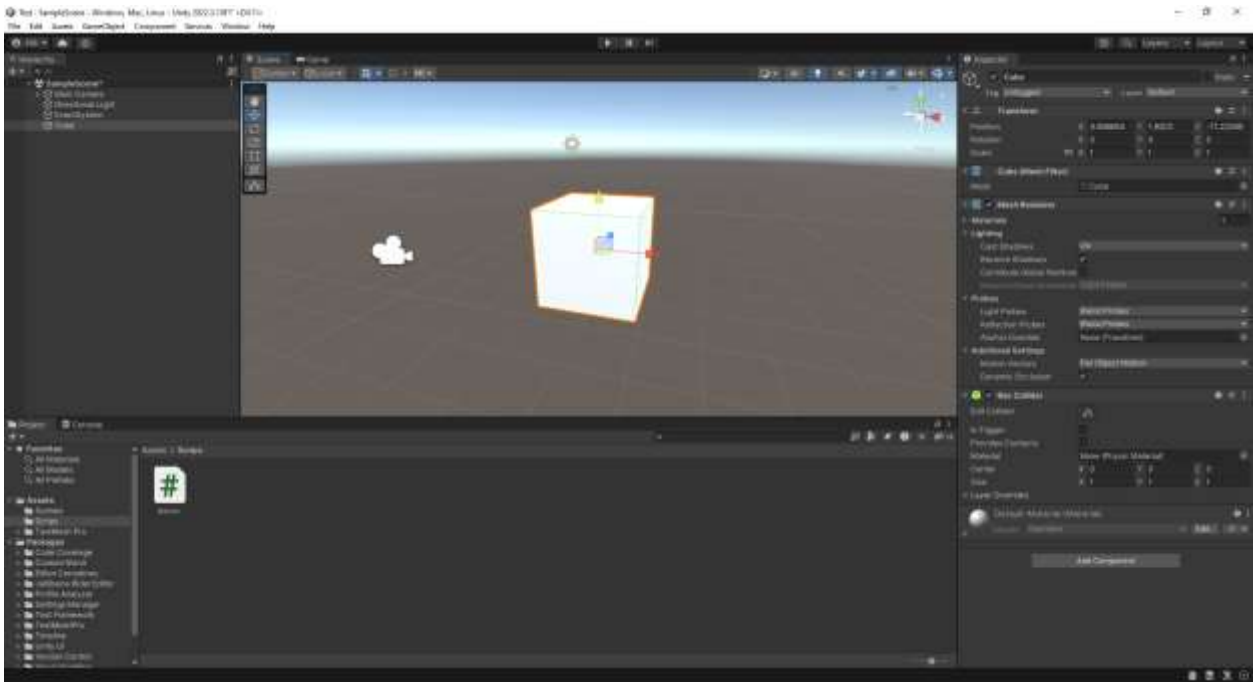

Рисунок 2.1 – Інтерфейс Unity

Однак, крім усіх переваг, Unity має і свої недоліки. По-перше, інтерфейс програми обмежує можливості візуального редактора при роботі з багатокомпонентними схемами. По-друге, відсутня вбудована підтримка зовнішніх бібліотек, тому розробникам доводиться налаштовувати їхню

інтеграцію самостійно. Також існує WebGL-версія, яка має низку невирішених проблем з оптимізацією, споживанням пам'яті та роботою на мобільних пристроях через особливості архітектури [35]. В цій версії трансляція коду з C# спочатку перетворюється на C++, а потім на JavaScript.

Ще одним недоліком можна назвати створення інтерфейсу користувача. Для цього потрібно використовувати компонент Canvas, що часто буває незручним. Альтернативою є пакет UIElements (UIToolkit) з підтримкою CSS, проте він не має функціоналу для стилів. Це означає, що в ньому відсутні анімації, маски, поля для редагування фону, шрифтів та тіней [36].

2.4 Порівняння альтернатив для розробки ігор

Як альтернативу Unity, можна згадати Unreal Engine, який на сьогоднішній день є однією з провідних платформ для розробки відеоігор. Є кілька причин, чому обрати його серед конкурентів:

- Графічний потенціал: Unreal Engine відомий своїм потужним графічним двигуном, який дозволяє створювати захоплюючі ігрові світи. У ньому доступний великий набір інструментів для реалістичної графіки, освітлення та візуальних ефектів.
- Фізичний двигун: Unreal Engine має потужний фізичний двигун, який дозволяє реалізувати реалістичну фізику об'єктів у грі. Це дозволяє створювати динамічні та інтерактивні середовища, а також реалістичні рухи персонажів.
- Blueprints та розширені інструменти програмування: Unreal Engine має інтуїтивну систему візуального програмування під назвою Blueprints, яка дозволяє розробникам без знань програмування виконувати складні інтерактивні дії. Крім того, можна використовувати мову програмування C++ для створення більш складних функцій та розширень.

Ураховуючи ці переваги, Unreal Engine стає конкурентом для Unity та приваблює розробників з різних сфер геймдеву.

Рисунок 2.2 – Інтерфейс Unreal Engine 5

Unreal Engine також забезпечує доступ до широкого спектру інформаційних ресурсів в Інтернеті для вирішення різних проблем, а також, як і Unity, є кросплатформеним [37]. В ігровому середовищі ієрархія об'єктів представлена у вигляді окремих елементів, кожен з яких володіє своїми властивостями та класом, який визначає ці властивості (рисунок 2.2).

Ще одною альтернативою може стати інший популярний двигун – GameMaker [38]. Ця платформа відома своєю простотою використання, що робить її популярним вибором серед початківців у галузі геймдеву, а також серед досвідчених розробників, які шукають швидкі та ефективні інструменти для створення відеоігор.

GameMaker також відомий своєю кросплатформеністю, що дозволяє розробникам створювати ігри для різних платформ, включаючи Windows, macOS, iOS, Android та інші. Це робить платформу привабливим вибором для тих, хто хоче розробляти ігри для різних пристроїв та операційних систем.

Ще однією перевагою GameMaker є широкий функціонал, що включає в себе інструменти для роботи з графікою, звуком, фізикою, штучним інтелектом та іншими аспектами гри. Це дозволяє розробникам створювати різноманітні та захоплюючі ігри без необхідності використання сторонніх інструментів.

Загалом, GameMaker є потужним інструментом для розробки відеоігор, який відзначається своєю простотою використання, кросплатформеністю та широким функціоналом. Ця платформа дозволяє розробникам будь-якого рівня досвіду створювати власні ігри швидко та ефективно.

Також однією з ключових особливостей GameMaker є його простий інтерфейс (рисунок 2.3), який дозволяє розробникам швидко освоїти платформу та почати створювати свої ігри. Крім того, платформа підтримує різні мови програмування, включаючи GML (GameMaker Language) та візуальне програмування за допомогою блок-схем, що робить її дружньою для користувачів з різним рівнем технічних знань.

Рисунок 2.3 – інтерфейс GameMaker

Альтернативою двох вищеописаних двигунів може стати Cocos Creator (рисунок 2.4). Це платформа для розробки ігор та мультимедійних додатків, яка надає розробникам широкий набір інструментів і можливостей для створення високоякісних ігрових проектів. Cocos Creator підтримує кілька мов програмування, включаючи Objective-C, Swift, C++, JavaScript та Lua, що дає розробникам можливість вибрати найбільш зручну мову для їхнього проекту

[39]. Інтегровані інструменти для створення графіки, анімації, фізики та інших аспектів гри дозволяють швидко та ефективно реалізовувати ідеї розробників.

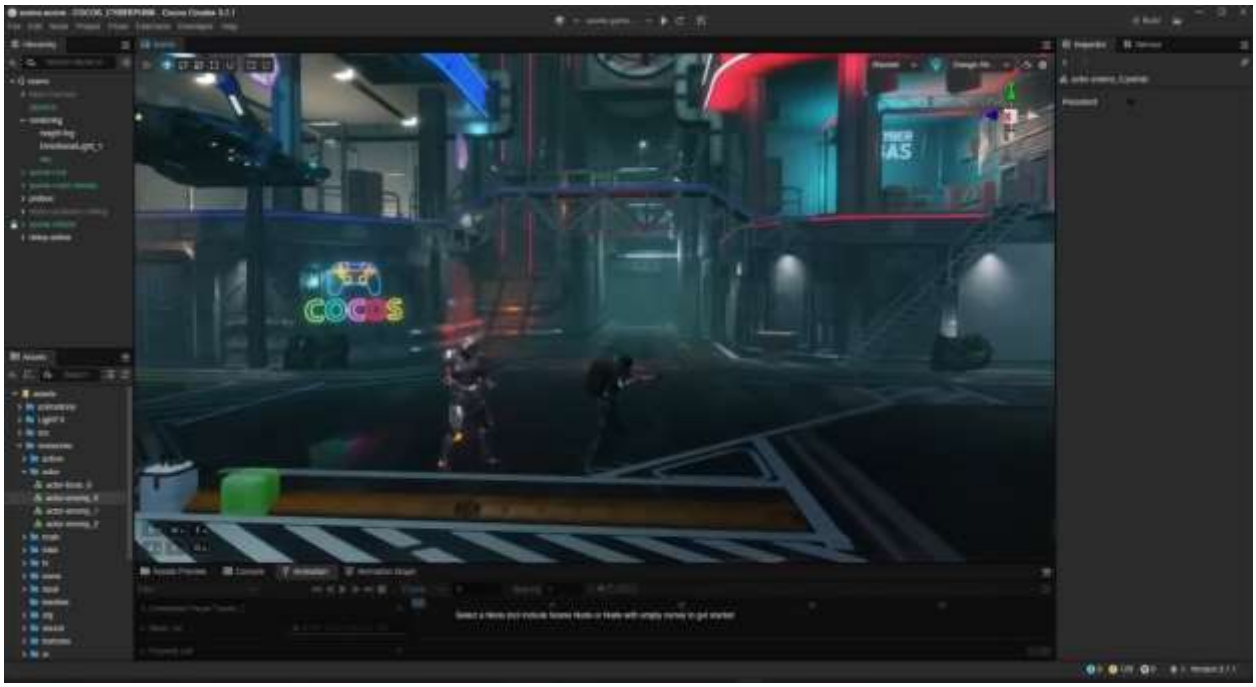

Рисунок 2.4 – Інтерфейс Cocos Creator

Основні відмінності між Cocos Creator і Unity можна виділити наступним чином:

- Unity підтримує C#, JavaScript та Boo (діалект Python), тоді як Cocos Creator підтримує Objective-C, Swift, C++, JavaScript та Lua.
- Unity дозволяє розробку ігор для різних платформ, включаючи Windows, macOS, iOS, Android, Xbox, PlayStation, тоді як Cocos Creator зазвичай використовується для розробки ігор для мобільних платформ, таких як iOS та Android.
- Unity відомий своїм потужним графічним двигуном, який дозволяє створювати відмінну графіку та реалістичні ігрові світи. Cocos Creator також має добрий графічний потенціал, але не настільки потужний, як у Unity.
- Unity має безкоштовну версію, але для доступу до деяких розширених функцій може знадобитися платна підписка. Cocos Creator є безкоштовною платформою з відкритим вихідним кодом.

- Unity має велику та активну спільноту користувачів, яка ділиться досвідом та ресурсами. Спільнота користувачів Cocos Creator менша, але також дуже активна.
- Unity має дружній та інтуїтивний інтерфейс, що полегшує розробку навіть для початківців. Інтерфейс розробки Cocos Creator може бути менш інтуїтивним для новачків, але пропонує більше гнучкості для досвідчених користувачів.

Менш популярною альтернативою є Godot Engine (рисунок 2.5). Це відкритий кросплатформовий двигун, який розробляється спільнотою Godot Engine Community [40]. Його основним принципом є повна відкритість коду та максимальна інтеграція, створюючи самодостатнє середовище розробки.

Рисунок 2.5 – інтерфейс Godot Engine

Цей двигун призначений для створення ігор різного жанру та складності і відзначається своєю потужністю та гнучкістю. Однією з основних особливостей Godot Engine є його безкоштовність та наявність відкритого вихідного коду, що дозволяє розробникам вільно користуватися та модифікувати його за своїми потребами.

Основні переваги Godot Engine включають кросплатформовість, що дозволяє розробникам створювати ігри для різних платформ, таких як Windows, macOS, Linux, iOS, Android, WebGL та інші. Крім того, розробники можуть використовувати різні мови програмування, такі як GDScript (спеціально розроблена мова для Godot), C#, C++, Python та VisualScript, для створення ігор та додатків.

Godot Engine надає інтегроване середовище розробки з гнучким інтерфейсом, що дозволяє легко створювати графіку, анімацію, фізику та логіку гри.

Порівнюючи інтерфейси Unity та Godot Engine, можна зазначити, що обидва двигуни мають простий та інтуїтивний інтерфейс та використовують ієрархічну структуру проекту. Однак, у відмінність від Unity, Godot Engine має ієрархічну структуру з вкладеними вкладками, що дозволяє зручніше перемикатися між сценами, редактором коду та ресурсами. Обидва двигуни також пропонують редактор WYSIWYG (What You See Is What You Get), який дозволяє переглядати зміни у реальному часі та сприяє швидкій настройці та візуалізації гри. Проте в Godot Engine цей редактор більш гнучкий та дозволяє більше налаштовувати інтерфейс порівняно з Unity.

2.5 Висновки до другого розділу

Сучасні екшн-ігри постійно вдосконалюються за допомогою нових механік та оригінальних підходів до геймплею, щоб залишати гравців зацікавленими, виходити вперед перед конкурентами та привертати нових учасників. Крім того, різноманітність двигунів, які застосовуються у створенні цих ігор, збільшує різноманіття ігрової індустрії. З цього випливає, що не існує одного універсального двигуна для створення екшн-ігор, і кожен розробник обирає той, який найкраще відповідає концепції його гри.

У таких іграх інтерфейс завжди схожий інтерфейс, але різний стиль графіки. Крім того, основною функцією інтерфейсу є відображення основних механік: здоров'я гравця, зброя гравця, артефакти гравця, противники та інше.

Для створення був обраний двигун Unity - кросплатформенний інструмент, який використовує мову програмування C# через її можливості та зручність. Проте, як і будь-який інший двигун, у нього є свої недоліки, які були описані в розділі.

Unity має ряд переваг перед іншими двигунами, такими як Unreal Engine 5, GameMaker, Cocos Creator та Godot Engine. По-перше, Unity підтримує широкий спектр платформ, що дозволяє створювати ігри для різних пристроїв, включаючи ПК, консолі, мобільні телефони та веб-браузери. По-друге, Unity має велику та активну спільноту користувачів і широкі ресурси для розробки, що включає в себе документацію, плагіни та різноманітні онлайн-ресурси. По-третє, Unity відзначається простотою використання, інтуїтивним інтерфейсом та легкістю освоєння, що робить його доступним навіть для початківців. По-четверте, для програмування в Unity використовується мова C#, яка є потужною та популярною серед розробників, полегшуючи процес створення ігор. Нарешті, Unity надає широкі можливості для розробки як 2D, так і 3D ігор, що дозволяє реалізувати різноманітні ідеї та концепції. Враховуючи ці переваги, Unity може бути вигідним вибором для розробки ігор у порівнянні з іншими платформами.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ГРИ «CASTLE TRANSYLVANIA»

3.1 Постановка завдання, правила та концепція гри

Перед початком розробки екшн-гри «Castle Transylvania» необхідно вирішити такі завдання:

- обрати та проаналізувати парадигму програмування, за допомогою якої буде створюватися проект;
- проаналізувати готові проекти по заданому жанру та їх геймплей;
- розробити дизайн для подальшої гри;
- розробити засоби для подальшого удосконалення проекту;
- дослідити потреби к інтерфейсу користувача по заданому жанру гри;
- проаналізувати та обрати спосіб розробки гри – інді-розробка або розробка у великій компанії;
- обрати ігровий двигун під потреби проекту та порівняти його з альтернативами.

Після створення проекту також важливим буде:

- розробити рекомендації щодо удосконалення проекту для подальшого розвитку;
- проаналізувати оцінки користувачів або тестерів;
- оцінити переваги та недоліки свого проекту.

Основні правила гри. Головною метою гри є виживання гравця якнайдовше при знищенні якомога більше ворогів. При цьому гравцю нараховуються очок та ресурси для покращення персонажа. Гравцю дається персонаж, який володіє магією посохів. Гра починається на карті з різними елементами декору. Персонаж керується за допомогою клавіш WASD, а атаки персонажа є автоматичними.

Вороги з'являються хвилями, їх кількість та сила збільшуються з часом. Кожен ворог має унікальні властивості, деякі з них можуть бути швидкими, інші — стійкими до певних типів атак. Персонаж має шкалу здоров'я, яка

зменшується при атаці ворогами, і коли здоров'я досягає нуля, гра закінчується. Вбиті вороги залишають кристали досвіду. За зібрані ресурси гравець може обрати покращення для персонажа, такі як нові види зброї, збільшення здоров'я, швидкість руху тощо.

Гра закінчується, коли персонаж вмирає або коли гравець вирішує завершити сесію. На основі часу виживання, кількості знищених ворогів та зібраних ресурсів гравець отримує фінальний результат. Castle Transylvania є захоплюючою грою, яка поєднує простоту управління з глибиною стратегічних рішень. Вона вимагає від гравця швидкої реакції, планування і постійного вдосконалення персонажа для виживання в умовах постійного зростання загрози.

3.2 Інтерфейс та розробка програмної частини

При запуску гри користувач бачить головне меню (рисунок 3.1). У ньому є дві основні кнопки: «Play» та «Exit». Натиснувши на кнопку «Play», гравець може почати гру. Натиснувши на кнопку «Exit», гравець може вийти з гри.

Рисунок 3.1 – Головне меню

При запуску гри на екрані відображається ігровий інтерфейс (рисунок 3.2), який включає: шкалу досвіду (вгорі по центру), рівень гравця (вгорі зліва), місце для посохів та артефактів (також вгорі зліва), кількість переможених супротивників (вгорі справа), ігрового персонажа (в центрі) та шкалу здоров’я (під ігровим персонажем).

Інтерфейс був розроблений з урахуванням доступності сприйняття для нових гравців, він також забезпечує необхідну інформативність.

Дизайн персонажів, супротивників, посохів, артефактів та декорацій розроблений для підтримки загального стилістики гри.

Рисунок 3.2 – Інтерфейс користувача

Після появи на рівні та отримання контролю над ігровим персонажем, на гравця починають нападати вороги (рисунок 3.3). Вороги миттєво виявляють місцезнаходження гравця і прямують до нього. Коли противник наближається до ігрового персонажа, він починає атакувати.

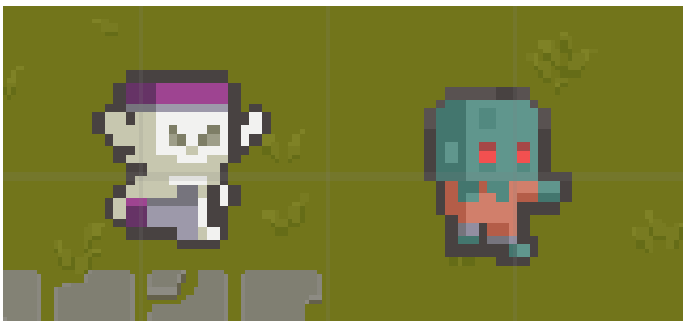

Рисунок 3.3 – Моделі ворогів в грі

Знаходження гравця реалізовано таким алгоритмом:

```
void Start()
{
    direction = new Vector2();
    rb = GetComponent<Rigidbody2D>();
    sr = GetComponent<SpriteRenderer>();
    target=FindObjectOfType<Player>().transform;
    Hp = MaxHp;
}
public void MoveToPlayer()
{
    Vector2 direction = (target.position - transform.position).normalized ;
    rb.velocity = direction * Movespeed;
}
```

У методі Start() спочатку створюється новий об'єкт типу Vector2 для зберігання напрямку руху. Потім отримуються посилання на компоненти Rigidbody2D та SpriteRenderer, які використовуються для керування фізикою об'єкту та відображення його на екрані відповідно. Далі знаходиться гравець у грі за допомогою методу FindObjectOfType<Player>() та зберігається посилання на його трансформацію для подальшого відстеження. Також, ініціалізується здоров'я об'єкту значенням максимального здоров'я. У методі MoveToPlayer() визначається напрямок до гравця як відстань між позиціями гравця та цього об'єкту, а потім нормалізується, щоб отримати одиничний вектор напрямку. Далі

встановлюється швидкість руху об'єкту у визначеному напрямку з використанням Rigidbody2D. Швидкість руху визначається змінною Movespeed. Таким чином, цей код забезпечує рух об'єкту у напрямку гравця з певною швидкістю.

Вороги з’являються випадково у визначених точках, які були попередньо встановлені. Вороги з’являються лише після знищення попередніх. Це реалізовано наступним кодом:

```
IEnumerator EnemySpawn()
{
   for (int i = 0; i < volna; i++)
   {
      randomPosition = UnityEngine.Random.Range(0, spawnPosition.Length);
      randomEnemy=UnityEngine.Random.Range(0, enemy.Length);
      Instantiate(enemy[randomEnemy],
spawnPosition[randomPosition].transform.position, Quaternion.identity);
      yield return new WaitForSeconds(1.5f);
   }
   volna++;
}
void Update()
{
   enemyList = GameObject.FindGameObjectsWithTag("Enemy");
   if (enemyList.Length > 0)
   {
   }
   else
   {
      StartCoroutine(EnemySpawn());
   }
}
```

Метод EnemySpawn() є IEnumerator, що дозволяє використовувати його для поетапного виконання його інструкцій. У цьому методі використовується цикл for для створення ворогів в заданій кількості хвиль. Для кожної хвилі генеруються випадкові позиції для створення ворогів та випадковий тип ворога з використанням методу UnityEngine.Random.Range(). Потім ворог створюється за допомогою функції Instantiate() з використанням випадкової позиції та типу ворога, після чого настає затримка на 1.5 секунди для наступного ворога. У методі Update() отримується масив всіх об'єктів з тегом "Enemy". Якщо на екрані є ще живі вороги, нічого не відбувається. Якщо ж всі вороги вбиті, запускається корутин EnemySpawn() для початку нової хвилі ворогів.

У грі зброя відіграє ключову роль у виживанні гравця та боротьбі проти орд ворогів. Основною зброєю є магічні посохи, які використовують магічні заряди для атаки ворогів. Існують три види посохів: звичайний, вогняний, зірковий, а також одна магічна сокира. Кожна зброя має унікальну поведінку.

Звичайний посох (рисунок 3.4) стріляє синім магічним снарядом, який летить до найближчого ворога від гравця. Пошук найближчого ворога реалізовано в на мові програмування C# наступним чином:

```
GameObject MinDistanc()
{
   float distanc;
   float minDistanc=float.MaxValue;
   foreach(GameObject go in enemy)
   {
      distanc = (transform.position - go.transform.position).magnitude;
      if (distanc < minDistanc)
      {
         minDistanc = distanc;
         closerEnemy = go;
      }
   }
```

```
    return closerEnemy;
}
```

Основний алгоритм складається з циклу foreach, який перебирає всі об'єкти ворогів з колекції enemy. Для кожного ворога обчислюється відстань до поточного об'єкта за допомогою виразу (transform.position – go.transform.position).magnitude, який використовує метод .magnitude для отримання довжини вектора, що представляє собою відстань між двома точками в просторі. Якщо поточна відстань менша за мінімальну знайдену до цього моменту відстань, змінні minDistanc та closerEnemy оновлюються відповідно. Після завершення циклу метод повертає найближчий знайдений об'єкт ворога. Цей метод ефективно обчислює і повертає найближчий ворог до поточного об'єкта, використовуючи просту лінійну перевірку всіх ворогів у колекції.

Рисунок 3.4 – Модель звичайного посоху

Вогняний посох (рисунок 3.5) стріляє вогняними кулями, які спрямовуються на найбільш віддаленого ворога від ігрового персонажа. Цей посох, на відміну від першого посоху, завдає значно більше шкоди ворогам, а його снаряди набагато більші, що полегшує влучання по ворогах.

Рисунок 3.5 – Модель вогняного посоху

Зірковий посох (рисунок 3.6) є унікальною зброєю, яка випускає маленькі зірочки, що автоматично переслідують найближчого ворога до ігрового персонажа. Ці зірочки мають вбудовану систему наведення, яка дозволяє їм самостійно знаходити ціль і точно вражати її, забезпечуючи максимальну ефективність у бою. Завдяки цій особливості, зірковий посох суттєво відрізняється від першого посоху, з яким гравець міг легко промахнутися. Зірковий посох гарантує влучання по ворогу, що робить його надзвичайно корисним для боротьби з швидкими та маневреними противниками, адже кожен випущений снаряд самостійно коригує свій курс, досягаючи цілі незалежно від її руху. Це робить зірковий посох одним з найнадійніших та найточніших видів зброї в арсеналі ігрового персонажа.

Рисунок 3.6 – Модель зіркового посоху

Магічна сокира (рисунок 3.7) у свою чергу є найсильнішим предметом у грі і завдає найбільшої шкоди супротивникам, проте вимагає від гравця виконувати певні умови щоб реалізувати цю шкоду, адже на відміну від посохів, сокира завдає шкоди тільки у певних напрямках від положення гравця, відповідно від гравця потрібно правильно позиціонувати себе щодо ворога, щоб завдавати найбільш можливої шкоди супротивникам. Підкидання вгору двох сокир, які потім падають на випадкового ворога, реалізоване алгоритмом:

```
void Start()
{
   rb = GetComponent<Rigidbody2D>();
   StartCoroutine(DestroyObject());
   Vector2 deflection = new Vector2(Random.Range(-2, 3), Random.Range(-2, 3));
   rb.AddForce(deflection+ Vector2.up *10f, ForceMode2D.Impulse);
}
```

Метод Start() викликається при активації об'єкта. Перший рядок коду отримує компонент Rigidbody2D, який додає фізичні властивості до об'єкта, і зберігає його в змінну rb. Далі, метод запускає корутину DestroyObject(), яка, відповідає за знищення об'єкта після певного періоду часу. Потім створюється вектор deflection з випадковими значеннями від -2 до 2 по обох осях. До компонента Rigidbody2D додається сила. Ця сила є сумою випадкового вектору deflection і вектору, спрямованого вгору з величиною 10. Використовується режим Impulse, який додає миттєву силу до об'єкта, змушуючи його рухатися. Метод Start() виконує початкову ініціалізацію об'єкта, зокрема: отримує компонент Rigidbody2D, щоб забезпечити фізичну взаємодію об'єкта з навколишнім середовищем, запускає корутину для майбутнього знищення об'єкта, та додає випадкову силу до об'єкта, щоб надати йому початковий імпульс і направити його вгору з випадковим відхиленням. Цей метод забезпечує динамічний початок руху об'єкта в грі, що робить його поведінку більш непередбачуваною та цікавою для гравця.

Рисунок 3.7 – Модель зіркового посоху

Артефакти в Castle Transylvania надають гравцю додаткові можливості та покращення, підвищуючи їхні шанси на виживання та ефективність у бою. Артефакти можуть збільшують показники здоров’я, регенерацію, шкоду по противникам та швидкість пересування дозволяючи гравцю адаптувати свої стратегії та стилі гри. Вибір правильного набору зброї та артефактів є ключовим для досягнення успіху та виживання у суворому світі гри.

Для отримання нової зброї або артефакту необхідно підвищити рівень гравця. Щоб підвищити рівень, гравець повинен збирати сфери досвіду, які випадають із переможених ворогів. Коли гравець накопичує достатню кількість досвіду, його рівень підвищується, і йому пропонується випадковий артефакт або випадкова зброя (рисунок 3.8).

Рисунок 3.8 – Вибір зброї або артефактів

Ключовим елементом фіналу є збирання якомога більшої кількості очок і ресурсів до моменту неминучої поразки. Протягом гри гравці вбивають ворогів, збирають корисні предмети, які допомагають їм покращувати свої здібності і досягати більш високих результатів. Чим довше гравець витримує атаки ворогів і збирає ресурси, тим вищі його підсумкові бали та тим більше можливостей відкривається для покращення персонажа в грі.

Після завершення рівня гравець отримує підсумковий екран з результатами, де вказано кількість зібраних ресурсів, вбитих ворогів, здобутих досягнень та іншу статистику. Ці дані можуть бути використані для покращення персонажа або розблокування нових можливостей для наступних ігор.

3.3 Висновки до третього розділу

У третьому розділі дипломної роботи були розглянуті етапи, складові та практичні аспекти розробки гри Castle Transylvania на платформі Unity із використанням мови програмування C#.

У процесі виконання розділу отримано такі результати:

- було реалізовано систему прогресії персонажа, що включає набір різноманітних навичок та зброї, які гравець може здобувати та розвивати в процесі гри;
- проведено оптимізацію гри для зниження затримок та підвищення якості гри;
- розроблено інтуїтивно зрозумілий інтерфейс, що забезпечує легкий доступ до основних функцій гри та покращує взаємодію гравців з ігровим середовищем;
- проведено тестування гри із залученням представників цільової аудиторії, що дозволило виявити сильні та слабкі сторони проекту.

На основі отриманих відгуків було внесені корективи, що покращили загальну якість гри та задовольнили потреби користувачів. Зокрема, було доопрацьовано баланс між різними типами зброї та навичок, а також поліпшено механіку адаптації складності, що дозволило зробити гру більш динамічною та цікавою для гравців з різним рівнем підготовки.

Сформульовано рекомендації щодо розширення ігрового контенту, вдосконалення існуючих механік та активної підтримки спільноти гравців. Зокрема, рекомендовано додати нові рівні, персонажів, зброю та навички, що забезпечить ще більшу реіграбельність та залучення нових користувачів. Також запропоновано впровадити систему регулярних оновлень та івентів, що стимулюватиме гравців повертатися до гри.

Визначено напрями для впровадження нових інновацій та технологій, що забезпечать подальший розвиток та комерційний успіх гри. У грі був розроблений унікальний стиль, концепція та зручний інтерфейс.

Таким чином, третій розділ роботи демонструє практичну реалізацію розробки гри Castle Transylvania, яка відповідає сучасним вимогам індустрії та враховує потреби цільової аудиторії. Досягнуті результати підтверджують ефективність обраних технологій та підходів, що є основою для подальшого вдосконалення проекту.

ВИСНОВКИ

У даній кваліфікаційній роботі були досліджені особливостей технологій програмної реалізації екшн гри під назвою Castle Transylvania, з унікальними ігровими механіками. В процесі проведеного дослідження були обрані програмні засоби Unity та C# для реалізації гри.

Гра Castle Transylvania, досягла поставлених цілей та підтвердила ефективність обраних технологій і підходів. У процесі роботи було здійснено детальний аналіз сучасних технологій і мов програмування, що дозволило зробити обґрунтований вибір на користь мови C# та платформи Unity. Це рішення забезпечило високу якість розробки та можливість реалізації складних ігрових механік.

Особлива увага була приділена розробці унікальних ігрових механік, які вирізняють гру серед інших ігор. Вони включають безкінечну прокачку зброї та артефактів, де гравець з кожним новим рівнем отримує можливість відкривати нові артефакти та зброю. Це не тільки стимулює інтерес до гри, але й забезпечує високу реіграбельність.

Аналіз ігрових механік та їх впливу на ігровий процес показав, що гра успішно поєднує простоту управління з глибокою та захоплюючою геймплейною системою. Гравцям пропонується широкий вибір стратегій для виживання, що вимагає від них адаптації та прийняття швидких рішень в умовах постійно зростаючої складності. Практичною цінністю гри є розважальні цілі. На основі проведеного дослідження можна зробити кілька висновків та надати рекомендації щодо подальшого розвитку та вдосконалення гри. Серед них:

1. Додавання нових персонажів, монстрів, локацій та карт розширило б геймплейні можливості та збільшило б варіативність гри, забезпечуючи більш довготривалий і захоплюючий досвід для гравців.
2. Вдосконалення існуючих механік та постійне оновлення та поліпшення існуючих ігрових механік, таких як баланс гравців та монстрів, адаптація складності гри, оптимізація інтерфейсу та управління, дозволить

підтримувати високий рівень задоволення від гри та зберігати інтерес гравців на тривалий термін.

3. Впровадження нових інновацій за допомогою новаторських ідей та технологій.
4. Додавання унікальних звуків для кожної зброї або взаємодії, та музики (це наситить ігровий процес та позитивно позначиться на сприйняття гравцями атмосфери).
5. Доповнення гри різноманітними тематичними анімаціями.
6. Переклад гри на інші мови (в тому ж порядку і на українську).
7. Розроблення бізнес-моделі для подальшого залучення гравців.
8. Додавання еволюції для зброї (змінює базову зброю на сильнішу покращену зброю, за наявності певного артефакту).
9. Додавання мультиплеєру до гри.
10. Додавання сюжетної лінії.

На момент напису кваліфікаційної роботи гра страждає на проблему відсутності звуків, музики та малу кількість персонажів. Були перешкоди при створенні унікальних механік, таких як створення випадкових карт для покращення персонажу та її програмна реалізація, але ці проблеми було вирішено.

На підставі відгуків гравців гри можна зробити висновок, що гра отримала позитивний прийом від цільової аудиторії та була оцінена ними як успішний продукт. Це свідчить про ефективність розробки та задоволення потреб користувачів. Із аналізу відгуків стає зрозумілим, що при подальшому розвитку гри шляхом додавання нового контенту та виправлення недоліків можна очікувати комерційного успіху, особливо при вдалих маркетингових заходах. З фінальними результатами роботи можна ознайомитись за посиланням [41].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Retro Diary. Retro Gamer. Bournemouth: Imagine Publishing (104): 2012.
2. Okita, Alex. Learning C# Programming with Unity 3D, second edition, 2019. — 690 с.
3. История разработки Sonic the Hedgeho
   URL: https://drukarnia.com.ua/articles/super-mario-istoriya-naividomishoyi-v-sviti-videogri-XGGxB (дата звернення 11.04.2024)
4. История разработки Sonic the Hedgehog
   URL:https://sonic.fandom.com/ua (дата звернення 11.04.2024)
5. Історія розвитку комп’ютерної графіки
   URL:https://wiki.cusu.edu.ua (дата звернення 11.04.2024)
6. Соціальні мережі та мобільні платформи
   URL:https://inproject.org (дата звернення 11.04.2024)
7. Розвитком стрімінгових сервісів
   URL:https://ua.hive-mind.community/blog/442,comu-striming-staje-dedali-populyarnisim (дата звернення 11.04.2024)
8. Сучасні ігри
   URL:https://www.microsoft.com/uk-ua/store/top-free/games/pc (дата звернення 11.04.2024)
9. Мобільні додатки
   URL:https://marketer.ua/ua/the-most-popular-apps-downloaded-by-ukrainians/ (дата звернення 15.04.2024)
10. Соціальні мережі та мобільні платформи
    URL:https://inproject.org (дата звернення 15.04.2024)
11. Інформація про RuneScape
    URL:https://play.runescape.com/ (дата звернення 17.04.2024)
12. Дані компанії UNIT.Cito
    URL:https://unit.city/ (дата звернення 17.04.2024)
13. Дані про команди студій

URL:https://newlms.magtu.ua/course/view.php?id=29522 (дата звернення 18.04.2024)

14. Компанії в Україні
URL:https://speka.media/100-naipributkovisix-it-kompanii-ukrayini-98grkp?utm_source=google&utm_medium=cpc&utm_campaign=21107681931&gad_source=1 (дата звернення 18.04.2024)
15. Інформація про Steam
URL:https://store.steampowered.com/about/index.html?l=ukrainian (дата звернення 20.04.2024)
16. Інформація про Microsoft
URL:https://news.microsoft.com/ua-ua/ (дата звернення 20.04.2024)
17. Роль у розвитку інді-індустрії в Україні
URL:https://dou.ua/forums/topic/32078/ (дата звернення 25.04.2024)
18. Devolver Digital
URL:https://www.devolverdigital.com/ (дата звернення 26.04.2024)
19. Станом на 2023 рік game developer
URL:https://gamedev.dou.ua/articles/top-gamedev-companies-winter-2022/ (дата звернення 26.04.2024)
20. Інформація про Dead Cells
URL:https://store.steampowered.com/app/588650/Dead_Cells/ (дата звернення 30.04.2024)
21. Сучасні roguelike-екшни
URL:https://mmo13.ua/games/feature/action-roguelike (дата звернення 30.04.2024)
22. Інформація про Vampire Survivors
URL:https://store.steampowered.com/app/1794680/Vampire_Survivors/ (дата звернення 30.04.2024)
23. Інформація про Brotato
URL:https://store.steampowered.com/app/1942280/Brotato/ (дата звернення 2.05.2024)

24. Інформація про Children of Morta
URL:https://store.steampowered.com/app/330020/Children_of_Morta/ (дата звернення 2.05.2024)
25. Інформація про Death Must Die
URL:https://store.steampowered.com/app/2334730/Death_Must_Die/ (дата звернення 2.05.2024)
26. Jez Humble, David Farley. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Addison-Wesley Signature Series (Fowler)). 2010. — 512 с.
27. Інформація про .NET Core
URL:https://learn.microsoft.com/ua-ua/dotnet/core/introduction (дата звернення 5.05.2024)
28. Інформація про Windows Presentation Foundation
URL:https://learn.microsoft.com/ua-ua/dotnet/desktop/wpf/overview/?view=netdesktop-8.0 (дата звернення 7.05.2024)
29. UnityScript’s long ride off into the sunset – Unity – Blog.
URL:https://blog.unity.com/community/unityscripts-long-ride-off-into-the-sunset (дата звернення 7.05.2024)
30. Хмарні сервіси C#
URL:https://training.epam.ua/ua/blog/491(дата звернення 10.05.2024)
31. Костриба О.В. Основи програмування. Частина 1. Visual C# Express Edition / О.В. Костриба. – Білогір’я, 2008. – 74 с.
32. Unity indie game solutions – Unity.
URL:https://unity.com/solutions/indie (дата звернення 10.05.2024)
33. Мови на яких можна писати в Unity
URL:https://itproger.com/ua/ (дата звернення 13.05.2024)
34. Scripting API: MonoBehaviour
URL:https://docs.unity3d.com/ScriptReference/MonoBehaviour.html (дата звернення 13.05.2024)

35. Build your WebGL application
URL:https://docs.unity3d.com/Manual/webgl-building.html (дата звернення 16.05.2024)
36. UI Toolkit
URL:https://docs.unity3d.com/Manual/UIElements.html (дата звернення 16.05.2024)
37. Introduction to Blueprints | Unreal Engine 4.27 Documentation – Unreal Engine.
URL:https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/ (дата звернення 17.05.2024)
38. Make 2D Games With GameMaker
URL:https://gamemaker.io/ua/ (дата звернення 20.05.2024)
39. Cocos Creator
URL:https://www.cocos.com/en/creator (дата звернення 20.05.2024)
40. Godot Engine
URL:https://godotengine.org/ (дата звернення 20.05.2024)
41. Google drive
URL:https://drive.google.com/drive/folders/1EYTZZZnMHh7XAAYyA0p4DRGVwtk1kpOZ?usp=sharing (дата звернення 27.05.2024)

ДОДАТОК

Лістинг програми

```
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using TMPro;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;

public class Player : Characteristic
{
    bool faceRight = true;
    public Animator animator;
    [SerializeField] GameObject[] allWeapon;
    public GameObject[] weaponPrefab;
    [SerializeField] GameObject[] cards;
    [SerializeField] GameObject canvasDied;
    private Rigidbody2D rb;
    Vector2 moveVector;
    public Image expBarImage;
    public Image hpBarImage;
    public TextMeshProUGUI lvlInterface;
    private int maxExp = 10;
    public int exp = 0;
    [SerializeField] int lvl=1;
    [SerializeField] float cooldawnBulet;
    public int regenerationInSecond;
    private bool isCooldawnBulet=true;
    private bool isCooldawnAxe=true;
    private bool isCooldawnStar=true;
    private bool isCooldawnFire=true;
    private bool isRegeneration=true;
    //private Transform Enemy;
    //public GameObject[] enemys;
    //public GameObject enemyzxc;

    // Start is called before the first frame update
    void Start()
    {
        animator = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
        moveVector = new Vector2();
```

```
        Hp = MaxHp;
    }

    // Update is called once per frame
    void Update()
    {
        //enemys = GameObject.FindGameObjectsWithTag("Enemy");
        Move();
        animator.SetFloat("MoveAnimator", Mathf.Abs(Mathf.Abs( moveVector.x)+
Mathf.Abs( moveVector.y)));
        Shoot();
        if(Hp<MaxHp)
        {
            if (isRegeneration)
            {
                Hp += regenerationInSecond;
                StartCoroutine(Regeneration());
                CheckHpBar();
            }
        }
        CheckHpBar();
        Reflect();
    }
    void Reflect()
    {
        if((moveVector.x>0 &&!faceRight) || moveVector.x < 0 && faceRight)
        {
            transform.localScale *= new Vector2(-1,1);
            faceRight=!faceRight;
        }
    }
    public void MoveSpeed()
    {
        Movespeed++;
    }
    IEnumerator Regeneration()
    {
        isRegeneration = false;
        yield return new WaitForSeconds(1);
        isRegeneration = true;
    }
    public void MaxHPBoost()
    {
        MaxHp += 10;
```

```
    CheckHpBar();
}

public void ExpBoost()
{
    exp += 10;
}
private void CheckHpBar()
{
    float newScale=(float)Hp/(float)MaxHp;
    hpBarImage.fillAmount = newScale;
}
private void CheckExpBar()
{
    float newScale = (float)exp / (float)maxExp;
    expBarImage.fillAmount = newScale;
}
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == ("Exp"))
    {
        Destroy(collision.gameObject);
        exp += 1;
        if(exp >= maxExp)
        {
            lvl++;
            maxExp += 10;
            exp = 0;
            lvlInterface.text = "LVl " + lvl;
            NewLvl();
        }
        CheckExpBar();
    }
}

public void TakeDamage(float damagePlayer)
{
    Hp-=damagePlayer;
    if(Hp <= 0)
    {
        Time.timeScale = 0;
        canvasDied.SetActive(true);
        Debug.Log("you dead");
    }
```

```
}

private void NewLvl()
{

    for (int i = 0; i<cards.Length; i++)
    {
        cards[i].SetActive(true);
    }
    GlobalEventManager.SendUpLvl();
    Time.timeScale=0;
}
private void Move()
{
    moveVector.x = Input.GetAxis("Horizontal");
    moveVector.y = Input.GetAxis("Vertical");
    moveVector*=Movespeed;
    rb.velocity = moveVector;
}
IEnumerator ShotBulet()
{
    isCooldawnBulet = false;
    yield return new WaitForSeconds(cooldawnBulet);
    isCooldawnBulet= true;
}
IEnumerator ShotAxe()
{
    isCooldawnAxe = false;
    yield return new WaitForSeconds(3f);
    isCooldawnAxe= true;
}
IEnumerator ShotStar()
{
    isCooldawnStar = false;
    yield return new WaitForSeconds(2f);
    isCooldawnStar = true;
}
IEnumerator ShotFire()
{
    isCooldawnFire = false;
    yield return new WaitForSeconds(5f);
    isCooldawnFire= true;
}
private void Shoot()
```

```
    {
        if (isCooldawnBulet)
        {
            GameObject bullet = Instantiate(weaponPrefab[0], transform.position,
Quaternion.identity);
            StartCoroutine(ShotBulet());
        }
        for (int i = 0; i < weaponPrefab.Length; i++)
        {
            if (weaponPrefab[i] == allWeapon[0] && isCooldawnAxe)
            {
                for(int j=0; j<2; j++)
                {
                    GameObject axe = Instantiate(weaponPrefab[i], transform.position,
Quaternion.identity);
                }
                StartCoroutine(ShotAxe());
            }
            else if (weaponPrefab[i] == allWeapon[1] && isCooldawnStar)
            {
                GameObject star = Instantiate(weaponPrefab[i], transform.position,
Quaternion.identity);
                StartCoroutine(ShotStar());
            }
            else if (weaponPrefab[i]== allWeapon[2] && isCooldawnFire)
            {
                GameObject fire = Instantiate(weaponPrefab[i],transform.position,
Quaternion.identity);
                StartCoroutine (ShotFire());
            }
        }


    }


}

using System.Collections;
using System.Collections.Generic;
using System.Data;
using System.Globalization;
using System.Reflection;
using Unity.VisualScripting;
```

```
using UnityEngine;

public class Enemy : Characteristic
{
    private Rigidbody2D rb;
    [SerializeField] private Transform target;
    [SerializeField] GameObject expPrefab;
    [SerializeField] float damagePlayer;
    public static float BoostDamage = 0;
    public Animator animator;
    private SpriteRenderer sr;
    private Vector2 direction;
    bool faceRight = true;

    // Start is called before the first frame update
    void Start()
    {
        direction = new Vector2();
        rb = GetComponent<Rigidbody2D>();
        sr = GetComponent<SpriteRenderer>();
        target=FindObjectOfType<Player>().transform;
        Hp = MaxHp;
    }
    public void MoveToPlayer()
    {
        Vector2 direction = (target.position - transform.position).normalized ;
        rb.velocity = direction * Movespeed;

    }

    public void TakeDamege(float damageBulletEnemy)
    {
        Hp -= damageBulletEnemy+BoostDamage;
        if (Hp <= 0)
        {
            Destroy(gameObject);
            GlobalEventManager.SendKillEnemy();
            GameObject expSpawn = Instantiate(expPrefab, transform.position,
Quaternion.identity);
        }
    }

    private void OnTriggerStay2D(Collider2D collision)
    {
```

```
        Player player = collision.GetComponent<Player>();
        if (collision.tag == ("Player"))
        {
            player.TakeDamage(damagePlayer);
            Debug.Log("Player take damage");
        }
    }

    // Update is called once per frame
    void Update()
    {
        MoveToPlayer();
        animator.SetFloat("Horizontal", Mathf.Abs(rb.velocity.x));
        Reflect();
    }
    void Reflect()
    {
        if ((rb.velocity.x > 0 && !faceRight) || rb.velocity.x < 0 && faceRight)
        {
            transform.localScale *= new Vector2(-1, 1);
            faceRight = !faceRight;
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using TMPro;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;

public class Cards : MonoBehaviour
{
    [SerializeField] Sprite[] image;
    [SerializeField] TextMeshProUGUI[] description;
    [SerializeField] Image[] icon;
    [SerializeField] GameObject[] cards;
    [SerializeField] Image[] weaponSlot;
    [SerializeField] Sprite[] spriteWeapon;
    [SerializeField] Image[] artifactSlot;
    [SerializeField] Sprite[] spriteArtifact;
    [SerializeField] string[] descriptionString;
    [SerializeField] GameObject PlayerGameObjeckt;
    private Player player;
```

```
[SerializeField] GameObject buletPrefab;
[SerializeField] GameObject axePrefab;
[SerializeField] GameObject starPrefab;
[SerializeField] GameObject firePrefab;
private TrigerBullet trigerBullet;
private AxeBullet axeBullet;
private ClosestBullet closestBullet;
private LastEnemyBullet lastEnemyBullet;

// Start is called before the first frame update
void Start()
{
    player = PlayerGameObjeckt.GetComponent<Player>();
    trigerBullet=buletPrefab.GetComponent<TrigerBullet>();
    axeBullet=axePrefab.GetComponent<AxeBullet>();
    closestBullet=starPrefab.GetComponent<ClosestBullet>();
    lastEnemyBullet=firePrefab.GetComponent<LastEnemyBullet>();
}

// Update is called once per frame
void Update()
{

}
public void OnClic(int i)
{
    switch (description[i].text)
    {
        case "Axe level up or take":
            for(i=0; i< player.weaponPrefab.Length; i++)
            {
                if (player.weaponPrefab[i] == axePrefab)
                {
                    axeBullet.UpDamage();
                    break;
                }
                else if (player.weaponPrefab[i] == null)
                {
                    Debug.Log("Take");
                    player.weaponPrefab[i] = axePrefab;
                    weaponSlot[i].sprite = spriteWeapon[1];
                    weaponSlot[i].gameObject.SetActive(true);
                    break;
                }
```

```
        }
        GameObjectFals();
        break;
    case "Staff level up or take":
        Debug.Log("Staff Level Up");
        for(i=0; i<player.weaponPrefab.Length; i++)
        {
            if (player.weaponPrefab[i] == buletPrefab)
            {
                trigerBullet.UpDamage();
                break;
            }
            else if (player.weaponPrefab[i] == null)
            {
                player.weaponPrefab[i] = buletPrefab;
                break;
            }
        }
        GameObjectFals();
        break;
    case "Star staff level up or take":
        for (i = 0; i < player.weaponPrefab.Length; i++)
        {
            if (player.weaponPrefab[i] == starPrefab)
            {
                closestBullet.UpDamage();
                break;
            }
            else if (player.weaponPrefab[i] == null)
            {
                Debug.Log("Take");
                player.weaponPrefab[i] = starPrefab;
                weaponSlot[i].sprite = spriteWeapon[3];
                weaponSlot[i].gameObject.SetActive(true);
                break;
            }
        }
        GameObjectFals();
        break;
    case "Fire staff level up or take":
        for (i = 0; i < player.weaponPrefab.Length; i++)
        {
            if (player.weaponPrefab[i] == firePrefab)
            {
```

```
                lastEnemyBullet.UpDamage();
                break;
            }
            else if (player.weaponPrefab[i] == null)
            {
                Debug.Log("Take");
                player.weaponPrefab[i] = firePrefab;
                weaponSlot[i].sprite = spriteWeapon[2];
                weaponSlot[i].gameObject.SetActive(true);
                break;
            }
        }
        GameObjectFals();
        break;
    case "Increased damage to enemies":
        for(i=0; i<artifactSlot.Length; i++)
        {
            if (artifactSlot[i].sprite == spriteArtifact[0])
            {
                break;
            }
            else if (artifactSlot[i].gameObject.activeSelf == false)
            {
                artifactSlot[i].sprite = spriteArtifact[0];
                artifactSlot[i].gameObject.SetActive(true);
                break;
            }
        }
        Enemy.BoostDamage++;
        GameObjectFals();
        break;
    case "Increases player regeneration":
        Debug.Log("Increases player regeneration");
        for (i = 0; i < artifactSlot.Length; i++)
        {
            if (artifactSlot[i].sprite == spriteArtifact[1])
            {
                break;
            }
            else if (artifactSlot[i].gameObject.activeSelf == false)
            {
                artifactSlot[i].sprite = spriteArtifact[1];
                artifactSlot[i].gameObject.SetActive(true);
                break;
```

```
            }
        }
        player.regenerationInSecond++;
        GameObjectFals();
        break;
    case "Increases the player's speed":
        Debug.Log("increases the player's speed");
        for(i=0 ; i<artifactSlot.Length ; i++)
        {
            if (artifactSlot[i].sprite == spriteArtifact[2])
            {
                break;
            }
            else if (artifactSlot[i].gameObject.activeSelf == false)
            {
                artifactSlot[i].sprite = spriteArtifact[2];
                artifactSlot[i].gameObject.SetActive(true);
                break;
            }
        }
        player.MoveSpeed();
        GameObjectFals();
        break;
    case "Increases the player's maximum health":
        Debug.Log("Increases the player's maximum health");
        for(i=0; i<artifactSlot.Length; i++)
        {
            if (artifactSlot[i].sprite == spriteArtifact[3])
            {
                break;
            }
            else if (artifactSlot[i].gameObject.activeSelf == false)
            {
                artifactSlot[i].sprite = spriteArtifact[3];
                artifactSlot[i].gameObject.SetActive(true);
                break;
            }

        }
        player.MaxHPBoost();
        GameObjectFals();
        break;

}
```

```
    }
    public void GameObjectFals()
    {
        for (int i = 0; i < cards.Length; i++)
        {
            cards[i].SetActive(false);
            Time.timeScale = 1;
        }
    }
    public void RandomCards(int i)
    {
        int random = randomMethod();
        TextCard(i,random);
        if (i == 1)
        {
            while (description[0].text == description[1].text)
            {
                random = randomMethod();
                TextCard(i,random);

            }

        }
        else if(i==2)
        {
            while (description[2].text == description[0].text || description[2].text
== description[1].text)
            {
                random= randomMethod();
                TextCard(i,random);
            }
        }
    }
    public void TextCard(int i, int random)
    {
        description[i].text = descriptionString[random];
        icon[i].sprite = image[random];
    }
    public int randomMethod()
    {
        int random;
        return random=Random.Range(0, descriptionString.Length);
    }
    private void Card()
```

```
    {
        for (int i = 0; i < cards.Length; i++)
        {
            RandomCards(i);
        }
    }
    private void OnDisable()
    {
        GlobalEventManager.UpLvl -= Card;
    }
    private void OnEnable()
    {
        GlobalEventManager.UpLvl += Card;
    }

}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpawnEnemy : MonoBehaviour
{
    [SerializeField] private Transform[] spawnPosition;
    [SerializeField] private GameObject[] enemy;
    [SerializeField] private GameObject[] enemyList;

    private int volna = 1;
    private int randomPosition;
    private int randomEnemy;
    // Start is called before the first frame update
    void Start()
    {

    }
    IEnumerator EnemySpawn()
    {
        for (int i = 0; i < volna; i++)
        {
            randomPosition = UnityEngine.Random.Range(0, spawnPosition.Length);
            randomEnemy=UnityEngine.Random.Range(0, enemy.Length);
            Instantiate(enemy[randomEnemy],
spawnPosition[randomPosition].transform.position, Quaternion.identity);
            yield return new WaitForSeconds(1.5f);
        }
```

```
            volna++;
        }

        // Update is called once per frame
        void Update()
        {
            enemyList = GameObject.FindGameObjectsWithTag("Enemy");
            if (enemyList.Length > 0)
            {

            }
            else
            {
                StartCoroutine(EnemySpawn());
            }
        }
}
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class TrigerBullet : MonoBehaviour
{
    public GameObject[] enemy;
    public GameObject closerEnemy;
    public GameObject bulletPrefab;
    public static int damageBullet=4;

    private Rigidbody2D rb;


    void Start()
    {
        enemy=GameObject.FindGameObjectsWithTag("Enemy");
        if(enemy == null)
        {
            Destroy(gameObject);
        }
        rb=GetComponent<Rigidbody2D>();
        MinDistanc();
        EnemyCloser();
        StartCoroutine(DestroyObject());
    }
```

```
    void Update()
    {


    }
    public void UpDamage()
    {
        damageBullet++;
    }
    private void EnemyCloser()
    {
        Vector2 direction = (closerEnemy.transform.position -
transform.position).normalized;
        rb.velocity = direction * 10f;
    }
    IEnumerator DestroyObject()
    {
        yield return new WaitForSeconds(10f);
        Destroy(gameObject);
    }

    GameObject MinDistanc()
    {
        float distanc;
        float minDistanc=float.MaxValue;
        foreach(GameObject go in enemy)
        {
            distanc = (transform.position - go.transform.position).magnitude;
            if (distanc < minDistanc)
            {
                minDistanc = distanc;
                closerEnemy = go;
            }
        }
        return closerEnemy;
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Enemy")
        {
            Enemy enemy = collision.GetComponent<Enemy>();
            enemy.TakeDamege(damageBullet);
            GameObject.Destroy(gameObject);
```

```
        }

    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ClosestBullet : MonoBehaviour
{
    public GameObject[] enemy;
    public GameObject closerEnemy;
    public GameObject bulletPrefab;
    public static int damageBullet = 3;

    private Rigidbody2D rb;


    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        StartCoroutine(DestroyObject());
    }

    void Update()
    {
        enemy = GameObject.FindGameObjectsWithTag("Enemy");
        MinDistanc();
        EnemyCloser();

    }
    public void UpDamage()
    {
        damageBullet++;
    }
    private void EnemyCloser()
    {
        Vector2 direction = (closerEnemy.transform.position -
transform.position).normalized;
        rb.velocity = direction * 10f;
    }
    IEnumerator DestroyObject()
    {
        yield return new WaitForSeconds(10f);
```

```
        Destroy(gameObject);
    }

    GameObject MinDistanc()
    {
        float distanc;
        float minDistanc = float.MaxValue;
        foreach (GameObject go in enemy)
        {
            distanc = (transform.position - go.transform.position).magnitude;
            if (distanc < minDistanc)
            {
                minDistanc = distanc;
                closerEnemy = go;
            }
        }
        return closerEnemy;
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Enemy")
        {
            Enemy enemy = collision.GetComponent<Enemy>();
            enemy.TakeDamege(damageBullet);
            GameObject.Destroy(gameObject);
        }

    }
}

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

public class LastEnemyBullet : MonoBehaviour
{
    public GameObject[] enemy;
    public GameObject closerEnemy;
    public GameObject bulletPrefab;
    public static int damageBullet = 8;

    private Rigidbody2D rb;
```

```
    void Start()
    {
        enemy = GameObject.FindGameObjectsWithTag("Enemy");
        if (enemy == null)
        {
            Destroy(gameObject);
        }
        rb = GetComponent<Rigidbody2D>();
        MinDistanc();
        EnemyCloser();
        StartCoroutine(DestroyObject());
    }

    void Update()
    {



    }
    public void UpDamage()
    {
        damageBullet++;
    }
    private void EnemyCloser()
    {
        Vector2 direction = (closerEnemy.transform.position -
transform.position).normalized;
        rb.velocity = direction * 10f;
    }
    IEnumerator DestroyObject()
    {
        yield return new WaitForSeconds(10f);
        Destroy(gameObject);
    }

    GameObject MinDistanc()
    {
        float distanc;
        float minDistanc = float.MinValue;
        foreach (GameObject go in enemy)
        {
            distanc = (transform.position - go.transform.position).magnitude;
            if (distanc > minDistanc)
            {
                minDistanc = distanc;
```

```
                closerEnemy = go;
            }
        }
        return closerEnemy;
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Enemy")
        {
            Enemy enemy = collision.GetComponent<Enemy>();
            enemy.TakeDamege(damageBullet);
            GameObject.Destroy(gameObject);
        }

    }
}

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

public class AxeBullet : MonoBehaviour
{
    public static int damageBullet = 7;
    public GameObject bulletPrefab;

    private Rigidbody2D rb;


    // Start is called before the first frame update
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        StartCoroutine(DestroyObject());
        Vector2 deflection = new Vector2(Random.Range(-2, 3), Random.Range(-2, 3));
        rb.AddForce(deflection+ Vector2.up *10f, ForceMode2D.Impulse);
    }

    // Update is called once per frame
    void Update()
    {

    }
    public void UpDamage()
```

```
    {
        damageBullet++;
    }
    IEnumerator DestroyObject()
    {
        yield return new WaitForSeconds(10f);
        Destroy(gameObject);
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Enemy")
        {
            Enemy enemy = collision.GetComponent<Enemy>();
            enemy.TakeDamege(damageBullet);
            GameObject.Destroy(gameObject);
        }

    }
}
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

public class StartsGame : MonoBehaviour
{
    // Start is called before the first frame update
    [SerializeField] GameObject menu;

    void Start()
    {
        Time.timeScale = 0f;
    }

    // Update is called once per frame
    void Update()
    {

    }
    public void GamePlaye()
    {
        Time.timeScale = 1f;
        menu.SetActive(false);
    }
```

```
    public void LeaveGame()
    {
        Application.Quit();
    }
}
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GlobalEventManager : MonoBehaviour
{
    public static Action killEnemy;
    public static Action UpLvl;
    public static void SendKillEnemy()
    {
        if(killEnemy != null) killEnemy.Invoke();
    }
    public static void SendUpLvl()
    {
        if(UpLvl != null) UpLvl.Invoke();
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Characteristic : MonoBehaviour
{
    [SerializeField] protected float Movespeed;
    [SerializeField] protected float MaxHp;
    [SerializeField] protected float Hp;

    // Start is called before the first frame update
    void Start()
    {

    }



    // Update is called once per frame
    void Update()
    {
```

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

public class RestartScene : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }
    public void RestartLvl()
    {
        string currentSceneName = SceneManager.GetActiveScene().name;
        SceneManager.LoadScene(currentSceneName);
        Time.timeScale = 1;
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```