

Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Інститут комп'ютерних систем
Кафедра інформаційних систем

Кравчишина Олена Олександрівна
студентка групи AI-214

ДИСЦИПЛІНА
Об'єктно-орієнтоване програмування

КУРСОВА РОБОТА
Розробка гри 2D

Спеціальність:
122 Комп'ютерні науки

Освітня програма:
Комп'ютерні науки

Керівник:
Годовиченко Микола Анатолійович,
кандидат технічних наук, доцент

Одеса – 2023

ЗМІСТ

Анотація	4
Вступ	5
1 Огляд різних систем-аналогів та технологій розробки мобільних додатків.....	7
1.1 Особливості використання мобільних технологій для розробки ігрових додатків в Unity	7
1.2 Огляд аналогічних ігрових додатків	8
1.2.1 Гра "Temple Run"	10
1.2.2 Гра "Monster Dash"	11
1.3 Формулювання вимог до основних функцій ігрового додатка	11
1.4 Огляд інформаційних технологій для розробки ігрового додатка в Unity	12
1.4.1 Використання рушія Unity для розробки ігор.....	12
1.4.2 Архітектурні особливості ігрового додатка в Unity	13
1.5 Висновки до першого розділу.....	13
2 Проектування ігрового додатка	15
2.1 Цілі та завдання ігрового додатка	15
2.2 Формулювання користувацьких сценаріїв та історій для ігрового додатка	16
2.3 Визначення нефункціональних вимог до ігрового додатка.....	16
2.4 Ідентифікація типового користувача ігрового додатка.....	16
2.5 Проектування користувацького інтерфейсу ігрового додатка.....	18
2.6 Висновки до другого розділу	19
3 Реалізація ігрового додатка в Unity.....	20
3.1 Структура ігрового проекту в Unity	21
3.2 Діаграма класів ігрового додатка	23
3.3 Керування вихідним кодом ігрового додатка	23
3.4 Функціональне тестування розробленого ігрового додатка.....	23
3.5 Інструкція користувача ігровим додатком	32
3.6 Висновки до третього розділу.....	39
ВИСНОВКИ.....	40

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ	41
-----------------------------------	----

АНОТАЦІЯ

Курсова робота присвячена розробці гри "Monster Mayhem: Escape and Shoot" в середовищі Unity з використанням C#. Гра є захоплюючою аркадою, де гравцю належить битися з монстрами, втікати та виживати в хаосі, наповненому небезпечними противниками. У грі будуть використані різноманітні елементи геймплею, включаючи рух персонажа, стрільбу, ворожих монстрів та різні рівні складності.

При розробці гри буде використано мову програмування C# для створення логіки гри, обробки взаємодії з об'єктами та контролю гравця. Unity надає потужні інструменти для розробки графічного інтерфейсу, анімації персонажів та налаштування фізики гри.

Метою цього проекту є розширення навичок розробки в Unity та поглиблення розуміння принципів роботи з C# для створення захоплюючих та розважальних ігрових досвідів. Курсова робота допоможе вам вдосконалити навички проектування іммерсивних ігор та реалізувати свої творчі ідеї.

ABSTRACT

The coursework is devoted to the development of the game "Monster Mayhem: Escape and Shoot" in the Unity environment using C#. The game is an exciting arcade where the player has to fight monsters, escape and survive in a chaos filled with dangerous opponents. The game will use a variety of gameplay elements, including character movement, shooting, enemy monsters, and different difficulty levels.

When developing the game, the C# programming language will be used to create game logic, handle interaction with objects, and control the player. Unity provides powerful tools for GUI development, character animation, and game physics customization.

The goal of this project is to expand your Unity development skills and deepen your understanding of C# principles to create exciting and entertaining gaming experiences. The coursework will help you improve your skills in designing immersive games and realize your creative ideas in the form of the game.

ВСТУП

У сучасному світі ігри стали невід'ємною частиною нашого життя. Вони надають нам можливість втекти в інші реальності, випробувати нові пригоди та

насолодитися захоплюючими віртуальними світами. Розробка власної гри є захоплюючим завданням, яке вимагає творчого мислення, програмування та використання сучасних інструментів.

Ця курсова робота присвячена розробці гри "Monster Mayhem: Escape and Shoot" в середовищі Unity з використанням мови програмування C#. Гра є захоплюючою аркадою, де гравцю належить битися з монстрами, втікати та виживати в хаосі, наповненому небезпечними противниками.

Метою цього проекту є розширення навичок розробки в Unity та поглиблення розуміння принципів роботи з C# для створення захоплюючих та розважальних ігрових досвідів. Під час розробки гри будуть використані різноманітні елементи геймплею, включаючи рух персонажа, стрільбу, ворожих монстрів та різні рівні складності. Гравець матиме можливість збирати бонуси та покращення, щоб полегшити свою місію та отримувати більше очок.

Під час розробки гри ви зможете поглибитися у вивченні мови програмування C# та його застосуванні у створенні ігор. Ви освоїте основні принципи роботи з Unity, яке надає потужні інструменти для розробки графічного інтерфейсу, анімації персонажів та фізики гри. Розробка гри "Monster Mayhem: Escape and Shoot" допоможе вам розширити свої знання в галузі геймдеву та реалізувати свої творчі ідеї у вигляді захопливої гри.

Таким чином, ця курсова робота пропонує вам відмінну можливість розгорнути свій потенціал у галузі розробки ігор, поглибити знання про Unity та мову програмування C#, а також створити власну захопливу гру "Monster Mayhem: Escape and Shoot".

1 ОГЛЯД РІЗНИХ СИСТЕМ-АНАЛОГІВ ТА ТЕХНОЛОГІЙ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

1.1 Особливості використання мобільних технологій для розробки ігрових додатків в Unity

Використання мобільних технологій для розробки ігрових додатків в Unity відкриває безліч можливостей для створення захоплюючих та інноваційних ігор, які можуть бути доступні на різних мобільних платформах, таких як iOS та Android. Особливості використання мобільних технологій включають:

Мобільні ресурси: Мобільні пристрої мають обмежені обчислювальні та графічні ресурси порівняно зі стаціонарними комп'ютерами. При розробці ігор для мобільних платформ важливо оптимізувати графіку, анімацію та ігрові механіки для ефективного використання обмежених ресурсів пристрою.

Екран та контроллери: Мобільні пристрої мають інші розміри екрану та специфічні методи введення, такі як сенсорні екрани, акселерометри та жести. При розробці ігор важливо враховувати ці особливості та створювати інтуїтивний ігровий інтерфейс, який легко користуватися на мобільних пристроях.

Оптимізація продуктивності: Мобільні пристрої мають різні характеристики та продуктивність, що може впливати на швидкість та стабільність гри. Розробники повинні оптимізувати гру, використовуючи методи, такі як рівні деталізації, управління пам'яттю та оптимізовані алгоритми для забезпечення плавної геймплею на різних пристроях.

Інтеграція з мобільними функціями: Мобільні пристрої мають широкий набір функцій, таких як камера, геолокація, акселерометр, мікрофон та інші. Розробники можуть використовувати ці функції для створення цікавих ігрових механік та взаємодії з користувачем.

Розповсюдження та монетизація: Мобільні платформи надають розробникам можливість легко розповсюджувати та монетизувати свої ігри через магазини додатків, такі як App Store та Google Play. Розробники можуть використовувати різні моделі монетизації, такі як покупки в програмі, реклама або підписки, для отримання прибутку від своїх ігор.

Загалом, використання мобільних технологій для розробки ігрових додатків в Unity дозволяє створювати захопливі та інноваційні ігри, які можуть бути

доступні широкому колу користувачів на мобільних пристроях. Важливо враховувати особливості мобільних платформ та оптимізувати гру для забезпечення найкращої геймплею та користувацького досвіду.

1.2 Огляд аналогічних ігрових додатків

Для покращення розуміння контексту та порівняння з іншими ігровими додатками, розглянемо огляд деяких аналогічних ігрових додатків, які пропонують схожий тип геймплею або концепцію:

- Гра "Temple Run" ,
- Гра "Monster Dash" ;

Ці приклади ігрових додатків демонструють різноманітність і можливості геймплею.

1.2.1 Гра "Temple Run"

"Temple Run" є однією з популярних ігор, яка була розроблена використовуючи Unity. Гра пропонує захоплюючий геймплей, де гравцю потрібно керувати персонажем, який втікає з храму, уникаючи перешкод, збираючи бонуси та досягаючи якомога більшої дистанції. Unity забезпечує потужний набір інструментів для розробки подібних ігор, включаючи рухову фізику, обробку колізій, анімацію персонажів та графіку.



Рисунки 1.1 Знімки екрану гри Temple Run

Основний функціонал гри "Temple Run" включає наступні елементи:

1. Безперервний біг: Гравець керує головним героєм, який безперервно біжить вперед. Головна мета полягає в тому, щоб уникати перешкод і триматися якомога довше.
2. Уникання перешкод: Гравець повинен швидко реагувати на різні перешкоди, такі як виїжджаючі вагонетки, дерева, скелі тощо. Він може переміщуватися вліво або вправо, а також стрибати або опускатися, щоб уникнути зіткнення.
3. Збирання бонусів: Під час бігу гравець може збирати різні бонуси, такі як монети, додаткові життя, посилення тощо. Ці бонуси можуть допомогти гравцю підвищити його результат і отримати більше можливостей для гри.
4. Поступове збільшення складності: Чим довше гравець виживає, тим швидше стає гра, і з'являються більш складні перешкоди. Це створює виклик та підвищує напругу гравця.
5. Результати та досягнення: Гра відстежує результати гравця, такі як відстань, пройдену ним, кількість зібраних монет і т.д. Також можуть бути наявні досягнення, які гравець може виконати для отримання нагород або відкриття додаткових можливостей.

Це лише загальний огляд основного функціоналу гри "Temple Run". Конкретні деталі і елементи гри можуть варіюватися в залежності від версії гри та розробника.

Платна версія гри має додатковий функціонал:

- Розширений набір персонажів: Платна версія може містити додаткових персонажів з унікальними властивостями або виглядом, яких немає у безкоштовній версії.
- Розблоковані рівні або локації: Платна версія може дозволяти гравцеві миттєво отримати доступ до всіх рівнів або додаткових локацій, які потрібно розблокувати або пройти в безкоштовній версії.

– Бонуси або підсилення: Платна версія може надавати гравцю додаткові бонуси, підсилення або спеціальні предмети, які полегшують геймплей або допомагають досягти кращих результатів.

– Видалення реклами: Платна версія може бути без реклами, що дозволяє гравцеві насолоджуватися грою без перебоїв.

1.2.2 Гра "Monster Dash"

Гра "Monster Dash" також є однією з ігор, розроблених з використанням Unity. У цій грі гравцю належить керувати головним героєм, який біжить по різних рівнях, уникаючи перешкод, стріляючи по монстрам та збираючи бонуси. Гра має швидкий темп і вимагає від гравця швидких реакцій та точних стрільб. Unity дозволяє розробникам створювати подібні ігри з використанням готових компонентів для обробки руху, стрільби, колізій та інших ігрових елементів.

Гра "Monster Dash" є екшн-платформером з елементами біжи та стрілянини, розробленим в Unity. Основна мета гри полягає в тому, щоб гравець перебігав якомога далі, уникавши перешкод та вбиваючи ворожих монстрів.

Основний функціонал гри "Monster Dash" може включати:

1. Різноманітні персонажі: Гравець може вибрати з різних героїв з унікальними характеристиками та навичками. Кожен персонаж може мати свою особливу здатність, яка допомагає в боротьбі з монстрами або подоланні перешкод.
2. Різні локації: Гра може містити різні локації, де гравець буде бігти та боротися з монстрами. Локації можуть мати унікальні дизайни та перешкоди.
3. Збір бонусів та покращень: Під час гри гравець може збирати бонуси, які надають додаткові переваги, такі як збільшення швидкості, збільшення сили атаки або тимчасова невразливість. Також можуть бути доступні покращення та зброя, які можна вдосконалювати з прогресом гри.
4. Рекорди та досягнення: Гра може відстежувати найкращі результати гравця і надавати можливість поділитися ними з друзями. Також можуть бути наявні

досягнення, які гравець може отримувати за досягнення певних цілей або виконання викликів.

5. Геймплейні режими: Крім основного режиму гри, можуть бути доступні інші режими, які додають різноманітність та виклики до геймплею. Наприклад, може бути режим безкінечного бігу, де гравець повинен вижити якомога довше.



Рисунок 2 – Знімки екрану гри Monster Dash

1.3 Формулювання вимог до основних функцій ігрового додатка

Формулювання вимог до основних функцій ігрового додатка є важливим кроком у процесі розробки гри. **Вимоги можуть бути наприклад такими:**

- **Аналіз жанру гри:** Розуміння жанру гри, яку ви хочете створити, допоможе вам визначити основні функції, які повинні бути присутні в додатку. Наприклад, якщо це гра-стратегія, можливі вимоги до основних функцій можуть включати систему управління ресурсами, будівництво та поліпшення споруд, битви з іншими гравцями тощо.
- **Визначення ігрового процесу:** Визначення основних кроків ігрового процесу та взаємодії гравця з грою допоможе вам сформулювати вимоги до функцій. Наприклад, якщо це гра-головоломка, можливі вимоги можуть

включати рівні складності, головоломки різного типу та можливість перегляду підказок.

- **Опис основних персонажів або об'єктів:** Якщо в грі є персонажі або основні об'єкти, формулювання вимог до їх функціональності допоможе у визначенні основних функцій гри. Наприклад, якщо є головний герой, можливі вимоги можуть включати рух, стрибки, атаку та використання спеціальних вмінь.
- **Встановлення цілей гри:** Визначення головної мети гри та підцілей допоможе вам у формулюванні вимог до функцій. Наприклад, якщо ціль гри - досягнення найвищого рівня або збір певної кількості балів, можливі вимоги можуть включати систему рівнів, систему нагород та систему підрахунку балів.
- **Врахування користувацького досвіду:** Розуміння потреб користувачів і їх очікувань допоможе вам сформулювати вимоги до функцій, що покращують користувацький досвід. Наприклад, можливі вимоги можуть включати інтуїтивний інтерфейс, можливість налаштування параметрів гри, зручну систему збереження прогресу тощо.

1.4.1 Огляд інформаційних технологій для розробки ігрового додатка в Unity

Для розробки ігрових додатків в Unity існує широкий спектр інформаційних технологій, які можна використовувати. **Ось деякі з них:**

1. Unity Engine: Unity є основною технологією для розробки ігрових додатків в Unity. Вона надає потужний набір інструментів для створення графіки, фізики, звуку та ігрової логіки.
2. C#: Unity використовує мову програмування C# для розробки функціональності ігрових додатків. C# є потужною мовою програмування з широким набором функцій і можливостей, що дозволяють реалізувати складну ігрову логіку.
3. Unity Asset Store: Unity Asset Store є онлайн-магазин, де можна знайти готові ассети, розширення та інструменти для розробки ігрових додатків в Unity. Це дає можливість ефективно використовувати готові ресурси та компоненти для прискорення процесу розробки.

4. 3D-модельовання та анімація: Для створення 3D-моделей та анімації можна використовувати програми, такі як Blender, Autodesk Maya, 3ds Max та інші. Unity підтримує імпорт цих моделей та анімаційних кліпів, що дозволяє створювати вражаючу графіку для ігрового додатка.
5. Фізика: Unity має вбудовану систему фізики, яка дозволяє моделювати реалістичні фізичні ефекти в ігрових додатках. Вона підтримує різні типи колізій, сили тяжіння, симуляцію руху та багато іншого.
6. Звук: Unity має можливості для роботи зі звуком, включаючи імпорт аудіофайлів, створення аудіо ефектів та змішування звуків. Це дозволяє створювати іммерсивний аудіо досвід для гравців.
7. Мобільні платформи: Unity підтримує розробку ігрових додатків для різних мобільних платформ, таких як iOS та Android. Вона надає можливість оптимізувати гру під конкретні мобільні пристрої та використовувати їх унікальні можливості, такі як сенсорний екран або акселерометр.

1.4.2 Архітектурні особливості ігрового додатка в Unity

Unity - це популярний двигун гри, який надає широкі можливості для розробки ігрових додатків на різних платформах. При розробці ігрового додатка в Unity, важливо враховувати кілька архітектурних особливостей, які допоможуть створити структурований і легко розширюваний код.

Архітектурні особливості, які враховуються при розробці ігрового додатка в Unity:

- **Компонентно-орієнтована архітектура:** Unity базується на парадигмі компонентно-орієнтованої розробки. Це означає, що функціональність гри будується навколо компонентів, які надають конкретні властивості та поведінку. Рекомендується розділяти логіку на окремі компоненти і використовувати компоненти для керування різними аспектами гри, такими як гравці, вороги, фізика, анімація і т.д.
- **Розділення обов'язків:** Рекомендується використовувати принцип розділення обов'язків (Separation of Concerns) для розмежування різних аспектів гри. Це означає, що логіку гри, графіку, управління, аудіо і т.д. слід

розміщувати в окремих модулях або скриптах. Це полегшує розуміння і підтримку коду, а також сприяє повторному використанню ігрових компонентів.

- **Модульність:** Рекомендується розробляти ігровий додаток з використанням модульної структури. Модулі дозволяють групувати схожі компоненти і функціональність, що полегшує розширення та управління кодом. Модульність також дозволяє розробляти різні частини гри паралельно, що покращує продуктивність розробника.
- **Використання шаблонів проектування:** Використання шаблонів проектування може сприяти покращенню архітектури гри. Наприклад, шаблон "Observer" може бути використаний для реалізації системи подій і сповіщень, а шаблон "Factory" може допомогти створити об'єкти з певними параметрами. Використання шаблонів допомагає уникнути дублювання коду і спрощує підтримку гри.
- **Управління станом:** В більш складних ігрових додатках важливо враховувати управління станом гри. Це означає, що ви повинні мати чітке уявлення про поточний стан гри і які переходи можуть відбутися між станами. Можна використовувати станові машини або інші підходи для реалізації управління станом.
- **Тестування:** Важливо розробляти гру з урахуванням можливостей тестування. Unity надає функціональність для автоматизованого тестування гри, що допомагає виявляти помилки і забезпечувати якість коду.
- **Оптимізація:** При розробці ігрового додатка важливо враховувати питання про оптимізацію продуктивності. Unity надає різні інструменти для профілювання і аналізу продуктивності гри, які допомагають знайти та виправити проблеми з продуктивністю.

1.5 Висновки до першого розділу

В першому розділі було проведено наступні кроки:

Представлено загальне визначення теми курсової роботи, а саме розробка ігрового додатка в середовищі Unity.

Було описано особливості використання мобільних технологій для розробки ігрових додатків в Unity.

Здійснено огляд аналогічних ігрових додатків, зокрема гри "Temple Run" і "Monster Dash", для показу схожих концепцій і можливостей.

Сформульовано вимоги до основних функцій мобільного додатку, що дозволяють організувати роботу над подальшим розробленням.

Проведено огляд інформаційних технологій, які використовуються для розробки ігрових додатків в Unity.

Зроблено висновки, які підсумовують отриману інформацію і надають загальну оцінку розглянутих аспектів.

Отже, в першому розділі було проведено аналіз, опис і огляд важливих елементів, що стосуються розробки ігрового додатка в Unity, що дало основу для подальшої розробки та дослідження.

2 ПРОЕКТУВАННЯ ІГРОВОГО ДОДАТКА

2.1 Цілі та завдання ігрового додатка

- Метою ігрового додатка "Monster Mayhem: Escape and Shoot" є створення захопливої аркадної гри, де гравець може насолоджуватися швидким і екшн-повним геймплеєм. Гра пропонує інтенсивні битви з монстрами, елементи виживання та викликів, які вимагають від гравця швидкості, точності та стратегічного мислення.
- Цей ігровий додаток розробляється з використанням Unity - потужного двигуна розробки ігор, який надає широкий набір інструментів для створення високоякісних ігрових досвідів. Unity дозволяє розробникам ефективно працювати з графікою, фізикою, анімацією, звуком та іншими аспектами гри, що сприяє створенню якісних ігрових продуктів.
- Використання Unity дозволяє покращити продуктивність розробки, забезпечити кросплатформову сумісність та досвід гри на різних пристроях, включаючи мобільні платформи. Крім того, Unity надає можливість використовувати мову програмування C# для реалізації функціональності

гри, що спрощує процес розробки та забезпечує потужний і гнучкий інструментарій для програмістів.

- Завдяки використанню Unity, ігровий додаток "Monster Mayhem: Escape and Shoot" має потенціал стати захоплюючим ігровим досвідом з високоякісною графікою, реалістичною фізикою та цікавим геймплеєм. Гравець матиме можливість насолоджуватися викликами та досягненнями, а також вдосконалювати свої навички у боротьбі з монстрами та досягненні нових рекордів.

2.2 Формулювання користувацьких сценаріїв та історій для ігрового додатка

З метою визначення користувацьких історій та нефункціональних вимог до веб-ресурсу, було розроблено діаграму сценаріїв використання гри (рис. 3). Діаграма сценаріїв(прецедентів) UML (Unified Modeling Language) - це графічний інструмент для опису функціональної взаємодії між користувачами та системою. Вона складається з акторів, сценаріїв та взаємодії між ними.

Ця діаграма містить основних акторів системи та описує сценарії їх взаємодії з системою. Вона допомагає проаналізувати залежності між акторами та можливими варіантами використання системи. Також ця діаграма швидко демонструє основні функції системи для розробників.



Рисунок 3 – Діаграма варіантів використання мобільного додатку

US1: Як користувач, я хочу почати нову гру, щоб випробувати свої навички та отримати задоволення.

Сценарій:

- Користувач запускає гру "Monster Mayhem: Escape and Shoot".
- Головне меню з'являється на екрані з опціями гри.
- Користувач натискає на кнопку "Нова гра".
- Гравець вибирає рівень складності (легкий, середній, складний).
- Гра починається з персонажем гравця розташованим в початковій позиції у світі, захопленому монстрами.
- Користувач керує персонажем, використовуючи рух, стрільбу та прижки для виживання та боротьби з монстрами.
- Гра триває, поки гравець не втрачає всі життя або не досягає мети рівня.
- Після закінчення гри, з'являється екран з результатами, що відображає кількість очків, досягнутий рівень та інші статистики.
- Користувач має можливість повторно почати гру або повернутися до головного меню.

US2: Як користувач, я хочу збирати предмети під час гри, щоб отримувати бонуси та покращувати свої можливості.

Сценарій:

- Під час гри, користувач помічає різні предмети, розташовані на рівні.
- Користувач керує персонажем і збирає предмети, натискаючи на них або проходячи повз них.
- Коли користувач збирає предмет, його кількість збільшується в ігровому інтерфейсі.
- Залежно від типу предмету, користувач може отримати бонуси, такі як збільшення життя, підвищення швидкості або покращення зброї.
- Користувач може використовувати накопичені предмети та бонуси для поліпшення персонажа або придбання нової зброї у магазині в головному меню гри.

US3: Як користувач, я хочу мати можливість зустрічати монстрів, які володіють стрільбою, щоб зробити гру більш викликаючою та захоплюючою.

- Під час гри, користувач зустрічає монстрів, які мають здатність стріляти в нього.
- Монстри можуть використовувати різні види зброї, такі як вогнепальна зброя, лазери або енергетичні снаряди.
- Користувач повинен використовувати свою вправність у ухиленні від пострілів, стріляти відповідно та знайти слабину у монстрів, щоб перемогти їх.
- Зі зростанням рівня гри, монстри збільшують складність та мають нові та потужніші види зброї, вимагаючи від користувача високого рівня навичок та стратегії, щоб подолати їх.

2.3 Визначення нефункціональних вимог до ігрового додатка

При визначенні нефункціональних вимог до ігрового додатка "Monster Mayhem: Escape and Shoot", можна звернути увагу на наступні аспекти:

1. Продуктивність: Додаток повинен працювати швидко і без затримок навіть при великій кількості монстрів, ворожих пострілів та бонусів на екрані.
2. Надійність: Гра повинна бути стабільною і не часто вішатися або зависати під час гри. Вона повинна мати мінімальну кількість помилок або збоїв.
3. Сумісність: Додаток повинен бути сумісним з різними платформами, такими як мобільні пристрої (Android, iOS) і комп'ютери (Windows, macOS), забезпечуючи однаковий функціонал і коректну роботу на всіх підтримуваних пристроях.
4. Графіка і звук: Вимоги до графічного оформлення і звукових ефектів можуть включати високу якість графіки, реалістичність монстрів і оточуючого світу, плавність анімацій, наявність звукових ефектів та музики, які доповнюють геймплей і створюють належну атмосферу гри.
5. Інтерфейс користувача: Гра повинна мати зручний і інтуїтивно зрозумілий інтерфейс користувача, де кнопки, меню та інші елементи керування розташовані зручно і легко доступні для гравця.

6. Безпека: Додаток повинен забезпечувати безпечне зберігання і обробку користувацьких даних, а також запобігати несанкціонованому доступу до гри або обману.
7. Скаляльність: Гра повинна бути гнучкою і скальованою, щоб забезпечувати задоволення ігрового процесу як на пристроях з низькими технічними характеристиками, так і на більш потужних пристроях.

2.4 Ідентифікація типового користувача ігрового додатка

Основні типові користувачі цієї гри можуть включати:

1. Геймери-любители: Це люди, які цінують ігри як засіб розваги і релаксації. Вони можуть бути зацікавлені в захопливому геймплеї, різноманітних викликах і досягненнях у грі. Такі користувачі можуть грати у гру у вільний час або під час перерв між робочими справами.
2. Фанати аркадних ігор: Ці користувачі насолоджуються швидкими, екшн-орієнтованими іграми з простим керуванням. Вони шукають адреналін та випробування своїх рефлексів. Гра "Monster Mayhem: Escape and Shoot" може приваблювати таких користувачів своїм швидким темпом, ворожими монстрами і потужними зброями.
3. Мобільні гравці: Оскільки гра розроблена для мобільних платформ, типовими користувачами можуть бути власники смартфонів і планшетів. Ці користувачі можуть грати у гру під час поїздок, вільних хвилин або просто насолоджуватися ігровим досвідом на мобільних пристроях.
4. Любителі викликів: Деякі користувачі люблять складні завдання та виклики, які вимагають від них стратегічного мислення, впевненості в рішеннях та навичок. Вони можуть бути зацікавлені в складних рівнях гри, розблокуванні досягнень та побитті рекордів.

2.5 Проектування користувацького інтерфейсу ігрового додатка

Проектування користувацького інтерфейсу (UI) ігрового додатка є важливою частиною розробки, оскільки воно визначає, як користувачі будуть взаємодіяти з грою та отримувати необхідну інформацію. Ефективний користувацький інтерфейс допомагає забезпечити зручність використання, легкість навігації та задоволення

від гри. Основні кроки при проектуванні користувацького інтерфейсу ігрового додатка включають:

1. Визначення функцій та вмісту: Встановлення основних функцій, які повинен виконувати інтерфейс, та визначення необхідного вмісту для його відображення. Це можуть бути кнопки, іконки, тексти, зображення тощо.
2. Розміщення елементів інтерфейсу: Встановлення правильного розташування елементів інтерфейсу на екрані, забезпечення їх зручного доступу та логічного розташування. Важливо враховувати ергономіку та простоту використання.
3. Вибір графічного стилю: Розробка графічного стилю, який відповідає загальному вигляду та настрою гри. Це може включати вибір кольорової палітри, шрифтів, стилів кнопок та інших елементів.
4. Використання анімацій та переходів: Додавання анімацій та переходів між елементами інтерфейсу для поліпшення візуального враження та покращення зрозумілості дій користувача.
5. Забезпечення легкості навігації: Розробка зручних методів навігації між екранами та функціями, таких як кнопки "назад" або меню навігації, щоб користувачі могли швидко й легко переміщатись у грі.
6. Тестування та оптимізація: Проведення тестування інтерфейсу з реальними користувачами для виявлення потенційних проблем та недоліків, а також внесення змін для поліпшення його функціональності та зручності використання.

Важливо також забезпечити сумісність інтерфейсу з різними пристроями та роздільними здатностями екрану, щоб гравці з різних пристроїв могли насолоджуватись грою без проблем.

В результаті правильного проектування користувацького інтерфейсу ігрового додатка, гравці зможуть легко орієнтуватись у грі, ефективно взаємодіяти з нею та насолоджуватись геймплеєм.

2.6 Висновки до другого розділу:

У другому розділі були сформульовані цілі та завдання ігрового додатка. Цілі визначають основні напрямки розробки додатка, а завдання визначають конкретні задачі, які необхідно виконати для досягнення поставлених цілей.

Були сформульовані користувацькі сценарії та історії для ігрового додатка. Користувацькі сценарії описують послідовність дій користувача в додатку, а історії розповідають про конкретні ситуації або завдання, які користувач може зустріти у грі.

Нефункціональні вимоги визначають властивості та характеристики додатка, які не відносяться безпосередньо до його функціональності. Вони охоплюють такі аспекти, як продуктивність, безпека, надійність, доступність та інші.

Був ідентифікований типовий користувач ігрового додатка. Це людина, яка має інтерес до гри, бажання провести час з задоволенням та досягти успіху у грі.

Ці етапи проектування допомагають визначити напрямок розробки ігрового додатка, описати користувацькі сценарії та нефункціональні вимоги, ідентифікувати цільових користувачів та розробити навігаційну структуру.

3 Реалізація ігрового додатка в Unity

Після запуску **Unity** вибираємо шаблон, який нам потрібен, в залежності яку гру хочемо зробити. В моєму випадку це стандартний шаблон 2D. Як це все виглядає зображено на рисунках 3-5.

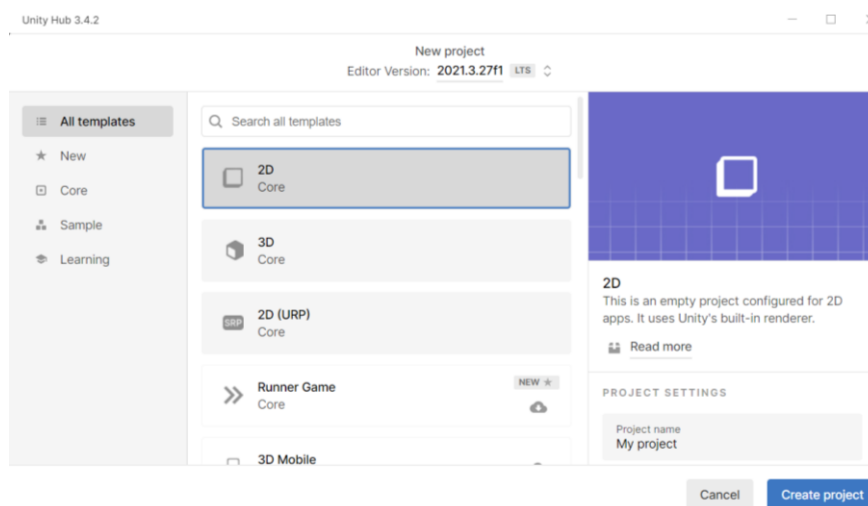


Рис.4 Продемонстровано вибір шаблону

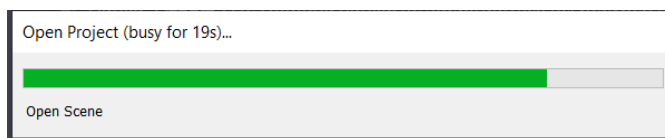


Рис.5 Зображено запуск проекту

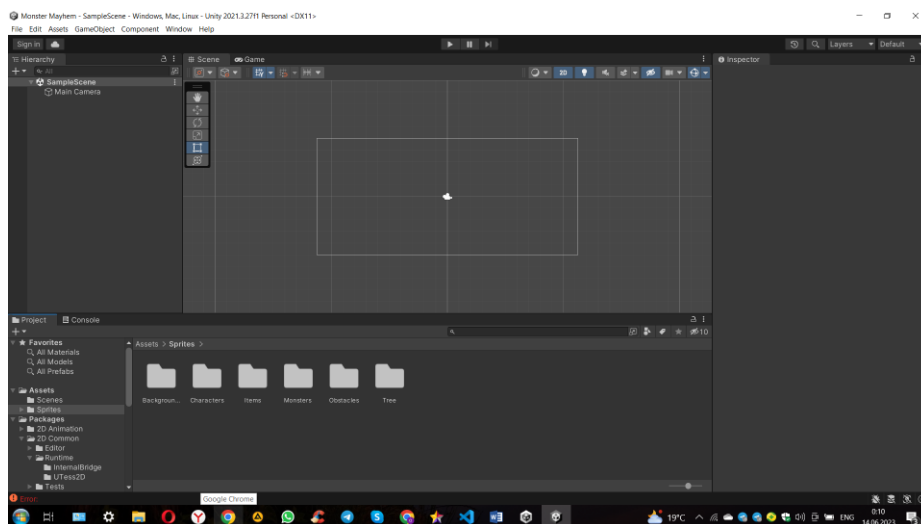


Рис.6 Вигляд проекту

3.1 Структура ігрового проекту в Unity

Загальна структура ігрового проекту в Unity:

1. **Папка "Assets":** Це головна папка, де зберігаються всі ресурси гри, такі як моделі персонажів, текстури, звуки, анімації, сцени та скрипти. Вона містить підпапки для кожного типу ресурсів. Як вона виглядає продемонстровано на рисунку 6.

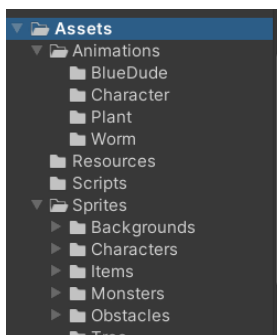


Рис. 7 Папка "Assets"

1. **Папка "Scripts" (Скрипти):** Ця папка містить всі скрипти, які використовуються в проекті. Скрипти відповідають за логіку гри, поведінку персонажів, управління інтерфейсом тощо. Як вона виглядає продемонстровано на рисунку 7.

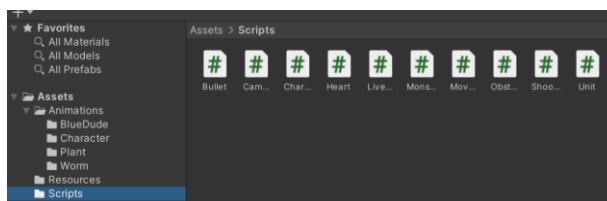


Рис.8 Папка "Scripts"

2. **Папка "Scenes" (Сцени):** В цій папці зберігаються сцени гри. Сцени визначають розташування та взаємодію об'єктів у грі. Кожна сцена може містити різні етапи гри, такі як головне меню, рівні геймплею, екрани досягнень тощо. Як вона виглядає продемонстровано на рисунку 8.

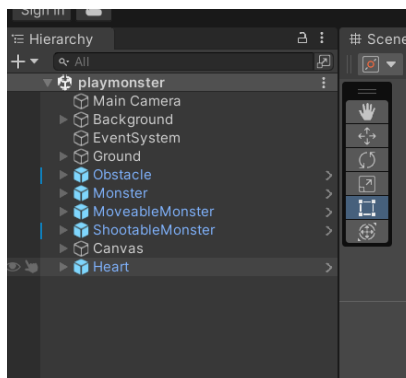


Рис. 9 Папка "Scenes"

3. **Папка "Resources":** В цій папці знаходяться фото, які використовуються для створення матеріалів (персонажів)об'єктів у грі. Як вона виглядає продемонстровано на рисунку 9.

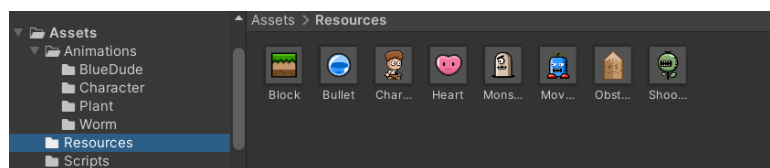


Рис. 10 Папка "Resources"

3.2 Діаграма класів ігрового додатка

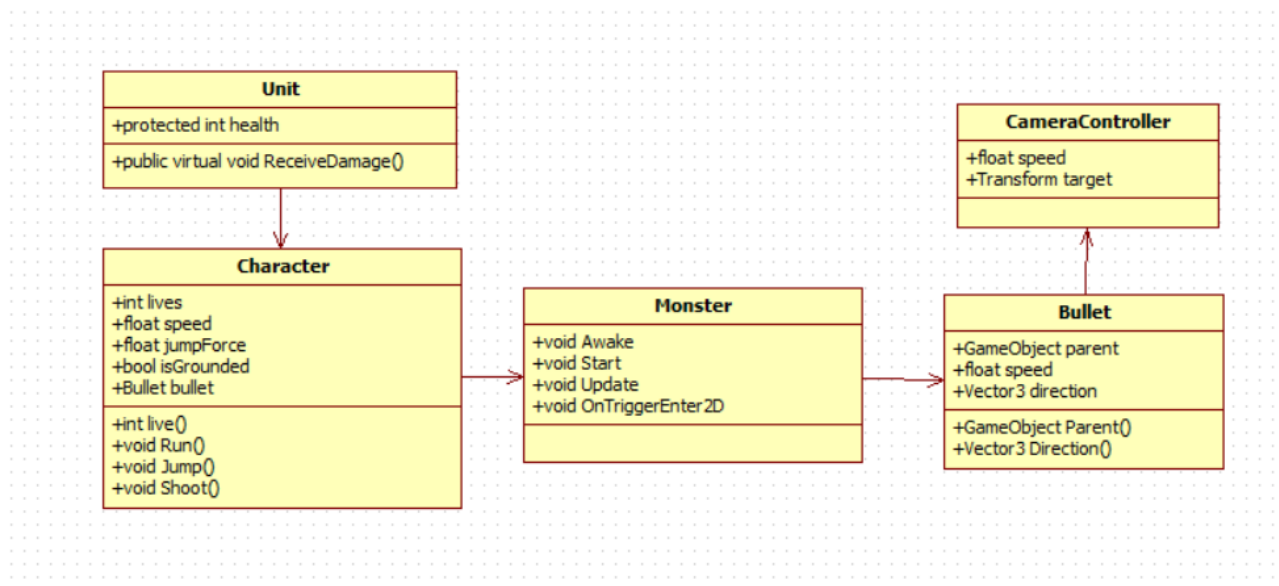


Рис.11 Діаграма класів

3.3 Керування вихідним кодом ігрового додатка

Використання інтегрованого середовища розробки (IDE): В моєму випадку Visual Studio для розробки ігрового додатка в Unity. Ці інструменти надають багато корисних функцій, таких як підказки, відлагодження коду, автоматичне завершення та багато іншого, що полегшує процес розробки.

3.4 Функціональне тестування розробленого ігрового додатка

Код, що наведений, описує клас *Bullet* (куля) у *gri*.

```

1 using UnityEngine;
2
3 public class Bullet : MonoBehaviour
4 {
5     // Посилання на батьківський об'єкт (примаркається базові)
6     private GameObject parent;
7     public GameObject Parent { set { parent = value; } get { return parent; } }
8
9     // Швидкість руху кулі
10    private float speed = 10.0f;
11
12    // Напрямок руху кулі (примаркається базові)
13    private Vector3 direction;
14    public Vector3 Direction { set { direction = value; } }
15
16    // Встановлення кольору кулі
17    public Color color
18    {
19        set { GetComponentInChildren().color = value; }
20    }
21
22    private void Start()
23    {
24        // Зникає куля через 1.4 секунди після створення
25        Destroy(gameObject, 1.4f);
26    }
27
28    private void Update()
29    {
30        // Переміщення кулі у напрямку direction зі швидкістю speed
31        transform.position += direction * speed * Time.deltaTime;
32    }
33
34    private void OnTriggerEnter2D(Collider2D collider)
35    {
36        // Отримання посилання на компонент Unit об'єкта, з яким зіткнулася куля
37        Unit unit = collider.GetComponent<Unit>();
38
39        // Перевірка чи це не є батьківський об'єкт кулі, інакше куля
40        if (unit != parent)
41        {
42            Destroy(gameObject);
43        }
44    }
45 }
  
```

Рис. 12 Зображений код клас *Bullet*

1. void Start()

- Ця функція викликається під час старту об'єкту.
- У даній функції встановлюється таймер, що знищує об'єкт (кулю) через 1.4 секунди після створення. Це забезпечує обмежену тривалість життя кулі.

2. **void Update()**

- Ця функція викликається на кожному кадрі гри.
- У даній функції здійснюється переміщення кулі у заданому напрямку (**direction**) з певною швидкістю (**speed**). Це забезпечує рух кулі по грі.

3. **void OnTriggerEnter2D(Collider2D collider)**

- Ця функція викликається, коли об'єкт, якому призначений колайдер, зіштовхується з іншим об'єктом, який також має колайдер.
- У даній функції перевіряється, чи зіштовхнулась куля з об'єктом, який є екземпляром класу **Unit** (наприклад, ворог).
- Якщо об'єкт, з яким зіткнулась куля, є юнітом і не є батьківським об'єктом кулі, то куля знищується (**Destroy(gameObject)**), щоб виключити подальше взаємодію з цим об'єктом.

4. Властивість **public Color Color**

- Ця властивість дозволяє встановити колір кулі.
- Вона приймає значення типу **Color** і задає цей колір для спрайту, який відображається на кулі.

5. Властивість **public GameObject Parent**

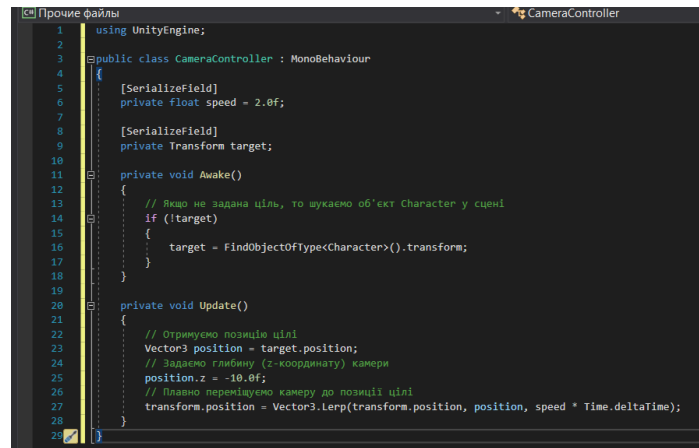
- Ця властивість дозволяє встановити посилання на батьківський об'єкт кулі.
- Вона приймає значення типу **GameObject** і зберігає це посилання.

6. Властивість **public Vector3 Direction**

- Ця властивість дозволяє встановити напрямок руху кулі.
- Вона приймає значення типу **Vector3** і задає цей напрямок для руху кулі.

Цей код демонструє, як керувати рухом і взаємодією кулі з іншими об'єктами у грі.

Даний код описує клас CameraController, який відповідає за керування камерою у грі.



```

1 using UnityEngine;
2
3 public class CameraController : MonoBehaviour
4 {
5     [SerializeField]
6     private float speed = 2.0f;
7
8     [SerializeField]
9     private Transform target;
10
11     private void Awake()
12     {
13         // Якщо не задана ціль, то шукаємо об'єкт Character у сцені
14         if (!target)
15         {
16             target = FindObjectOfType<Character>().transform;
17         }
18     }
19
20     private void Update()
21     {
22         // Отримуємо позицію цілі
23         Vector3 position = target.position;
24         // Задасмо глибину (z-координату) камери
25         position.z = -10.0f;
26         // Плавне переміщення камери до позиції цілі
27         transform.position = Vector3.Lerp(transform.position, position, speed * Time.deltaTime);
28     }
29 }

```

Рис.13 Зображений код класу CameraController

1. void Awake()

- Ця функція викликається під час запуску сцени.
- У даній функції перевіряється, чи задана ціль для камери (**target**). Якщо ціль не задана, то шукається об'єкт типу **Character** у сцені за допомогою методу **FindObjectOfType()**. Знайдений об'єкт стає ціллю для камери.

2. void Update()

- Ця функція викликається на кожному кадрі гри.
- У даній функції отримується поточна позиція цілі (**target.position**).
- Задасється глибина (z-координата) камери, щоб камера була віддалена від сцени.
- Використовуючи метод **Lerp()**, камера плавно переміщується до позиції цілі з заданою швидкістю (**speed * Time.deltaTime**).

Цей код демонструє просту реалізацію камерного контролера, який слідкує за цільовим об'єктом (наприклад, головним героєм) і плавно переміщує камеру до його позиції. Це дозволяє створити ефект слідування за головним героєм під час руху.

Даний код описує клас Character, який представляє головного героя в грі.

```

1 using UnityEngine;
2
3 public class Character : Unit
4 {
5     [SerializeField]
6     private int lives = 5;
7
8     public int Lives
9     {
10         get { return lives; }
11         set
12         {
13             // Перевіряємо, щоб значення життя було не більше 5
14             if (value < 5)
15                 lives = value;
16             livesBar.Refresh();
17         }
18     }
19     private LivesBar livesBar;
20
21     [SerializeField]
22     private float speed = 3.0f;
23     [SerializeField]
24     private float jumpForce = 15.0f;
25
26     private bool isGrounded = false;
27
28     private Bullet bullet;
29
30     private CharState State
31     {
32         get { return (CharState)animator.GetInteger("State"); }
33         set { animator.SetInteger("State", (int)value); }
34     }
35
36     new private Rigidbody2D rigidbody;
37     private Animator animator;
38     private SpriteRenderer sprite;
39
40     // Підключення компонентів та завантаження снаряду
41     private void Awake()
42     {
43         livesBar = FindObjectOfType<LivesBar>();
44         rigidbody = GetComponent<Rigidbody2D>();
45         animator = GetComponent<Animator>();
46         sprite = GetComponentInChildren<SpriteRenderer>();
47
48         bullet = Resources.Load<Bullet>("Bullet");
49     }
50
51     // Оновлення стану героя
52     private void FixedUpdate()
53     {
54         CheckGround();
55     }
56
57     // Обробка введення користувача
58     private void Update()
59     {
60         if (isGrounded)
61             State = CharState.Idle;
62
63         if (Input.GetButtonDown("Fire1"))
64             Shoot();
65         if (Input.GetButton("Horizontal"))
66             Run();
67         if (isGrounded && Input.GetButtonDown("Jump"))
68             Jump();
69     }
70
71     // Біг героя
72     private void Run()
73     {
74         Vector3 direction = transform.right * Input.GetAxis("Horizontal");
75         transform.position = Vector3.MoveTowards(transform.position, transform.position + direction, speed * Time.deltaTime);
76         sprite.flipX = direction.x < 0.0f;
77
78         if (isGrounded)
79             State = CharState.Run;
80     }
81
82     // Перебіг героя
83     private void Jump()
84     {
85         rigidbody.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
86     }
87
88     // Вистріл героя
89     private void Shoot()
90     {
91         Vector3 position = transform.position;
92         position.y += 0.8f;
93         Bullet newBullet = Instantiate(bullet, position, bullet.transform.rotation) as Bullet;
94         newBullet.Parent = gameObject;
95         newBullet.direction = newBullet.transform.right * (sprite.flipX ? -1.0f : 1.0f);
96     }
97
98     // Обробка отримання пошкодження
99     public override void ReceiveDamage()
100     {
101         Lives--;
102
103         rigidbody.velocity = Vector3.zero;
104         rigidbody.AddForce(transform.up * 8.0f, ForceMode2D.Impulse);
105
106         Debug.Log(lives);
107     }
108
109     // Перевірка снаряду на землю
110     private void CheckGround()
111     {
112         Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position, 0.3f);
113
114         isGrounded = colliders.Length > 1;
115
116         if (!isGrounded)
117             State = CharState.Jump;
118     }
119
120     // Обробка зіткнення зі снарядом
121     private void OnTriggerEnter2D(Collider2D collider)
122     {
123         Bullet bullet = collider.gameObject.GetComponent<Bullet>();
124         if (bullet && bullet.Parent != gameObject)
125         {
126             ReceiveDamage();
127         }
128     }
129
130     // Перевірка стану героя
131     public enum CharState
132     {
133         Idle,
134         Run,
135         Jump
136     }
137 }

```

Рис.14 код описує клас Character

1. **void Awake()**

- У даній функції виконується пошук та підключення компонентів, які пов'язані з героєм.
- Знаходиться компонент **LivesBar** для відображення панелі життя героя.
- Отримується посилання на компоненти **Rigidbody2D**, **Animator** та **SpriteRenderer** для керування фізикою, анімацією та відображенням спрайту героя.
- Завантажується префаб снаряду (**Bullet**) з ресурсів.

2. **void FixedUpdate()**

- Ця функція викликається з фіксованою частотою на кожному кадрі фізики.
- У даній функції перевіряється, чи герой знаходиться на землі за допомогою методу **CheckGround()**.

3. **void Update()**

- Ця функція викликається на кожному кадрі гри.
- У даній функції оброблюється введення користувача.
- Якщо герой знаходиться на землі, то встановлюється стан **CharState.Idle**.
- Якщо натиснута клавіша "Fire1" (зазвичай, ліва кнопка миші), то викликається функція **Shoot()**.
- Якщо клавіша "Horizontal" (зазвичай, стрілки вліво/вправо) утримується, то викликається функція **Run()**.
- Якщо герой знаходиться на землі і натиснута клавіша "Jump", то викликається функція **Jump()**.

4. **void Run()**

- Ця функція відповідає за рух героя.
- Визначається вектор напрямку руху залежно від натискання клавіші "Horizontal".

- Герой плавно переміщується в заданому напрямку з заданою швидкістю.
- В залежності від напрямку руху відбувається зміна орієнтації спрайту героя.
- Якщо герой знаходиться на землі, то встановлюється стан **CharState.Run**.

5. **void Jump()**

- Ця функція відповідає за стрибок героя.
- Герой отримує імпульс у напрямку вгору (**transform.up**) з заданою силою (**jumpForce**).

6. **void Shoot()**

- Ця функція відповідає за вистріл героя.
- Визначається позиція створення снаряду (**Bullet**) над героєм.
- Створюється екземпляр снаряду (**newBullet**) за допомогою методу **Instantiate()**.
- Встановлюються значення батьківського об'єкту та напрямку руху снаряду залежно від орієнтації героя.

7. **public override void ReceiveDamage()**

- Цей метод перевизначає метод з батьківського класу **Unit** і відповідає за обробку отримання пошкоджень героєм.
- Зменшується значення життів героя.
- Герой отримує імпульс у напрямку вгору, що дає ефект відкидання.
- Виводиться повідомлення про залишок життів героя в консоль.

8. **void CheckGround()**

- Ця функція перевіряє наявність землі під героєм.
- За допомогою методу **Physics2D.OverlapCircleAll()** перевіряється, чи є колайдери навколо героя в заданому радіусі.
- Якщо кількість колайдерів більше 1, то герой знаходиться на землі.
- Якщо герой не знаходиться на землі, то встановлюється стан **CharState.Jump**.

9. void OnTriggerEnter2D(Collider2D collider)

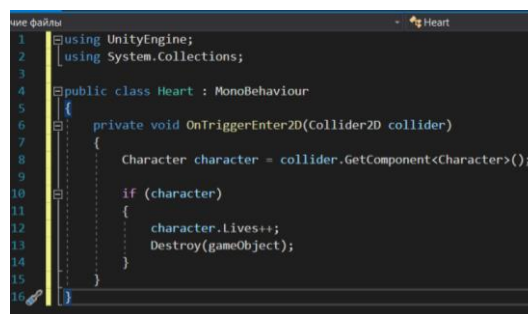
- Ця функція викликається при зіткненні героя з колайдером.
- Перевіряється, чи зіткнутий об'єкт є снарядом (**Bullet**) та чи не є герой батьківським об'єктом цього снаряду.
- Якщо умова виконується, то викликається метод **ReceiveDamage()**, який обробляє отримання пошкоджень героєм.

10.enum CharState

- Перерахування, що визначає стани героя (**Idle, Run, Jump**).

Це описує функціональність кожної функції в коді класу **Character**.

Цей код відповідає класу Heart, який визначає поведінку об'єкту "Heart" у грі.



```

1 using UnityEngine;
2 using System.Collections;
3
4 public class Heart : MonoBehaviour
5 {
6     private void OnTriggerEnter2D(Collider2D collider)
7     {
8         Character character = collider.GetComponent<Character>();
9
10        if (character)
11        {
12            character.Lives++;
13            Destroy(gameObject);
14        }
15    }
16 }

```

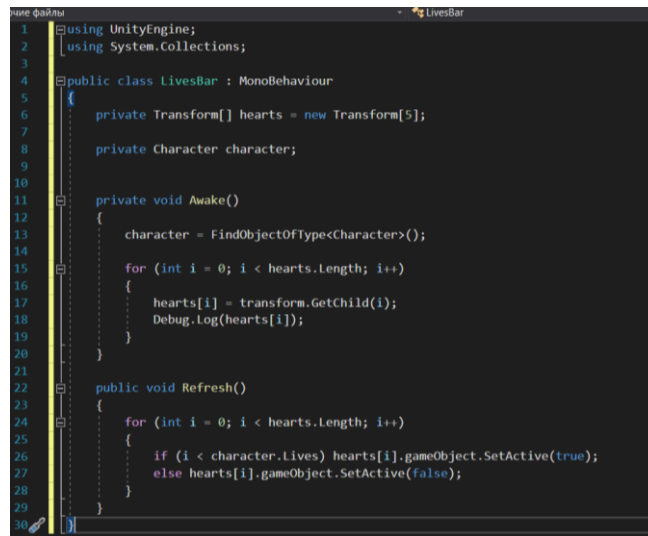
Рис.15 код відповідає класу Heart

Основна функція цього класу - **OnTriggerEnter2D(Collider2D collider)**, яка викликається при зіткненні об'єкту "Heart" з колайдером.

1. void OnTriggerEnter2D(Collider2D collider)

- Ця функція викликається при зіткненні об'єкту "Heart" з іншим колайдером.
- З отриманого колайдеру отримується посилання на компонент **Character** за допомогою методу **GetComponent<Character>()**.
- Якщо отриманий компонент **Character** не є **null**, то це означає, що об'єкт "Heart" зіткнувся з героєм.
- Збільшується значення життів героя за допомогою звернення до властивості **Lives** об'єкту **character**.
- Об'єкт "Heart" знищується за допомогою методу **Destroy(gameObject)**.

Цей код відповідає класу LivesBar, який відповідає за відображення і оновлення інтерфейсу зображення життів гравця.



```

1 using UnityEngine;
2 using System.Collections;
3
4 public class LivesBar : MonoBehaviour
5 {
6     private Transform[] hearts = new Transform[5];
7
8     private Character character;
9
10
11     private void Awake()
12     {
13         character = FindObjectOfType<Character>();
14
15         for (int i = 0; i < hearts.Length; i++)
16         {
17             hearts[i] = transform.GetChild(i);
18             Debug.Log(hearts[i]);
19         }
20     }
21
22     public void Refresh()
23     {
24         for (int i = 0; i < hearts.Length; i++)
25         {
26             if (i < character.Lives) hearts[i].gameObject.SetActive(true);
27             else hearts[i].gameObject.SetActive(false);
28         }
29     }
30 }

```

Рис.16 код відповідає класу LivesBar

Основні функції цього класу:

1. void Awake()

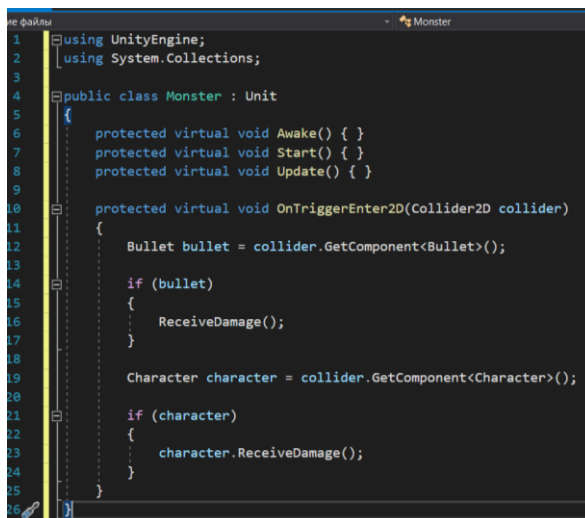
- Ця функція викликається при створенні об'єкту.
- Знаходить компонент **Character** у сцені за допомогою методу **FindObjectOfType<Character>()**.
- Заповнює масив **hearts** з посиланнями на дочірні об'єкти поточного об'єкта (тобто дочірні об'єкти, що відображають серця).
- Виводить кожне посилання на серце у консоль за допомогою **Debug.Log(hearts[i])**.

2. public void Refresh()

- Ця функція викликається для оновлення відображення життів гравця.
- Ітерується по масиву **hearts**.
- Якщо індекс менше значення життів гравця (**i < character.Lives**), вмикається графічний об'єкт серця за допомогою **hearts[i].gameObject.SetActive(true)**.
- В іншому випадку (якщо індекс більший або дорівнює значенню життів гравця), вимикається графічний об'єкт серця за допомогою **hearts[i].gameObject.SetActive(false)**.

Цей клас забезпечує відображення кількості життів гравця на ігровому інтерфейсі та оновлює його при зміні значення життів.

Цей код відповідає класу Monster, який є базовим класом для монстрів в грі. Він містить ряд віртуальних методів та логіку зіткнень з кулями та гравцем.



```

1  using UnityEngine;
2  using System.Collections;
3
4  public class Monster : Unit
5  {
6      protected virtual void Awake() { }
7      protected virtual void Start() { }
8      protected virtual void Update() { }
9
10     protected virtual void OnTriggerEnter2D(Collider2D collider)
11     {
12         Bullet bullet = collider.GetComponent<Bullet>();
13
14         if (bullet)
15         {
16             ReceiveDamage();
17         }
18
19         Character character = collider.GetComponent<Character>();
20
21         if (character)
22         {
23             character.ReceiveDamage();
24         }
25     }
26 }

```

Рис. 17 код відповідає класу Monster

Основні елементи цього класу:

1. **protected virtual void Awake() { }**

- Цей метод може бути перевизначений у похідних класах монстрів.
- Викликається при створенні об'єкта.
- Може використовуватись для ініціалізації даних або налаштування об'єкта монстра.

2. **protected virtual void Start() { }**

- Цей метод може бути перевизначений у похідних класах монстрів.
- Викликається перед першим оновленням кадру.
- Може використовуватись для початкових дій або налаштування об'єкта монстра перед грою.

3. **protected virtual void Update() { }**

- Цей метод може бути перевизначений у похідних класах монстрів.
- Викликається на кожному оновленні кадру гри.
- Може використовуватись для оновлення стану монстра або виконання додаткової логіки.

4. **protected virtual void OnTriggerEnter2D(Collider2D collider)**

- Цей метод може бути перевизначений у похідних класах монстрів.
- Викликається при зіткненні об'єкта монстра з іншим коллайдером.
- Перевіряє, чи зіткнувся об'єкт з кулею, отримує пошкодження відповідно до методу **ReceiveDamage()**.
- Також перевіряє, чи зіткнувся об'єкт з гравцем (**Character**), і наносить пошкодження гравцеві.

Цей клас надає базову структуру та логіку для монстрів у грі. Він може бути розширений і перевизначений в похідних класах для визначення конкретних властивостей та поведінки окремих типів монстрів.

3.5 Інструкція користувача ігровим додатком

Інструкція користувача для ігрового додатка:

1. Запустіть ігровий додаток на вашому пристрої.
 2. Ваша мета - вижити, борючись з монстрами та виконуючи завдання гри. Використовуйте свої навички, стратегію та реакцію, щоб перемогти ворогів і досягти успіху.
 3. Для переміщення свого персонажа використовуйте відповідні кнопки або джойстик. Дотримуйтесь правил гри, уникайте перешкод та збирайте сердечки, щоб вижити.
 4. Якщо ви зустрічаєте монстра, спробуйте атакувати його, використовуючи доступні зброю або спеціальні рухи. Зберіться і виконайте належні дії, щоб уникнути пошкоджень і нанести ворогові шкоду.
 5. У разі отримання пошкоджень зберігайте свої життя, використовуючи доступні об'єкти або інші можливості відновлення. Уважно стежте за показником своїх життів або енергії та дійте своєчасно.
 6. Продовжуйте грати, доки не досягнете кінцевої мети гри .
- Насолоджуйтесь грою та відчуйте задоволення від своїх досягнень у світі ігор.

Ігровий сценарій може включати наступні етапи:

1. Гравець керує персонажем, який може рухатись вправо та вліво.
2. Зелений монстр постійно рухається вгору-вниз або з боку на бік.

3. Зелений монстр випускає зелені пулі у напрямку гравця з певною частотою або за випадковим принципом.
4. Якщо зелена пуля торкається гравця, гравець втрачає одну життя.
5. Гравець може кидати сині пулі у напрямку монстра, намагаючись влучити в нього.
6. Якщо синя пуля торкається зеленого монстра, монстр отримує пошкодження.
7. Якщо монстр отримує певну кількість пошкоджень, він знищується.
8. Є ще один монстр, який рухається вправо-вліво з певною швидкістю.
9. Якщо гравець успішно пригинається на монстрів, вони отримують пошкодження та знищуються.
10. Гра триває, поки гравець не втратить всі життя або не знищить усіх монстрів.
11. Після закінчення гри гравець може перезапустити гру.

Ось детальна інформація про дії, які відбуваються в цій грі, з скріншотами.

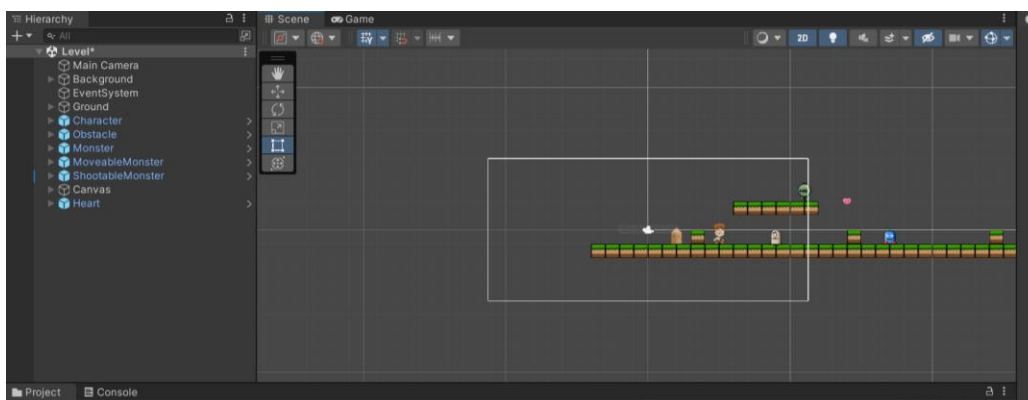


Рис.18 Зображена в цілому вся гра, без запуску

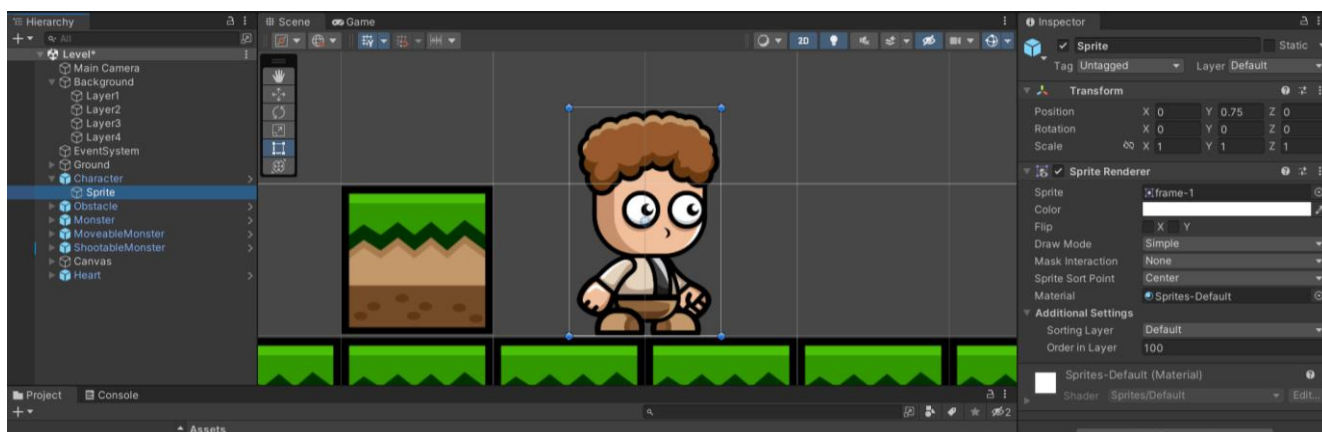


Рис.19 Зображений головний персонаж, який задіяний в цій грі

Він має змогу бігати вправо(вперед) , вліво(назад) та стрибати вгору(для того щоб знищити монстра).

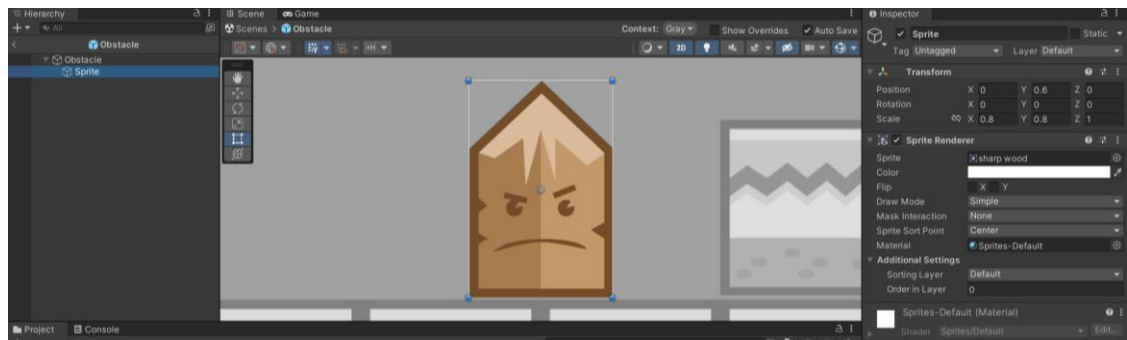


Рис.20

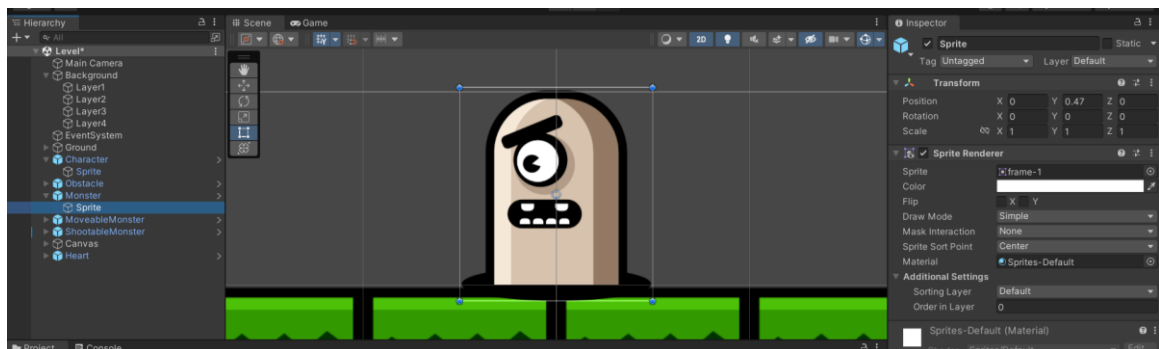


Рис.21

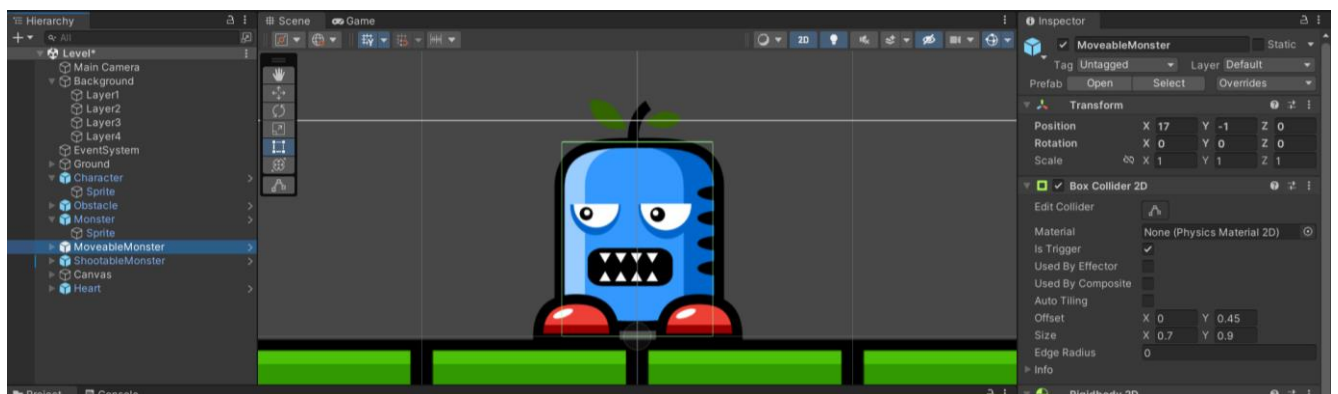


Рис.22

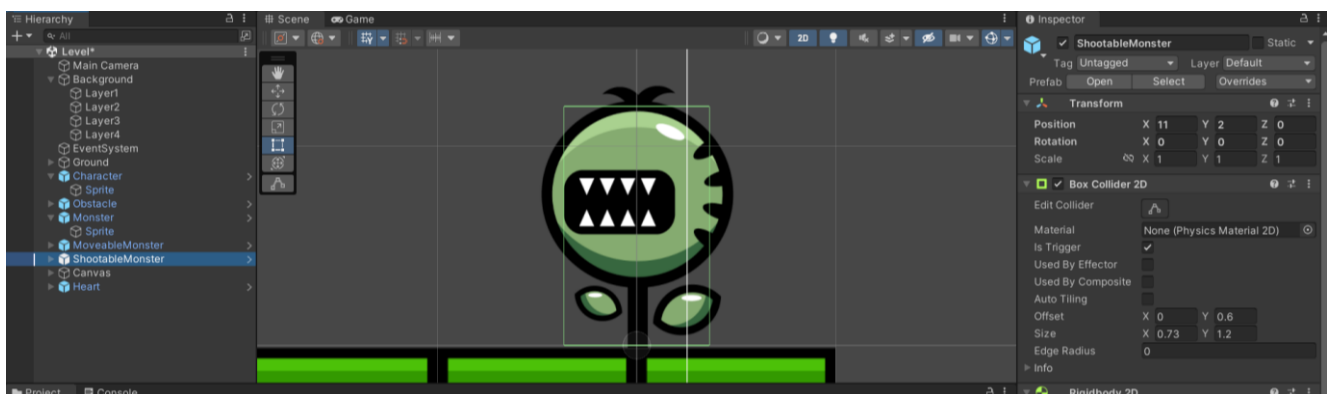


Рис.23

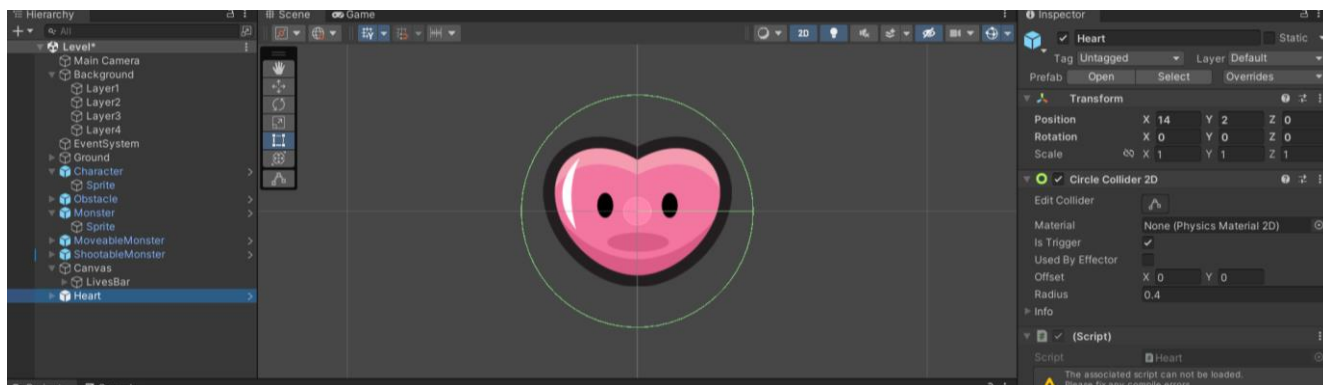


Рис.24 Зображено сердечко(в грі це життя).

Їх на гру дається 5, і коли в тебе пуляє монстр/ворог пулю, то одне життя забирається.

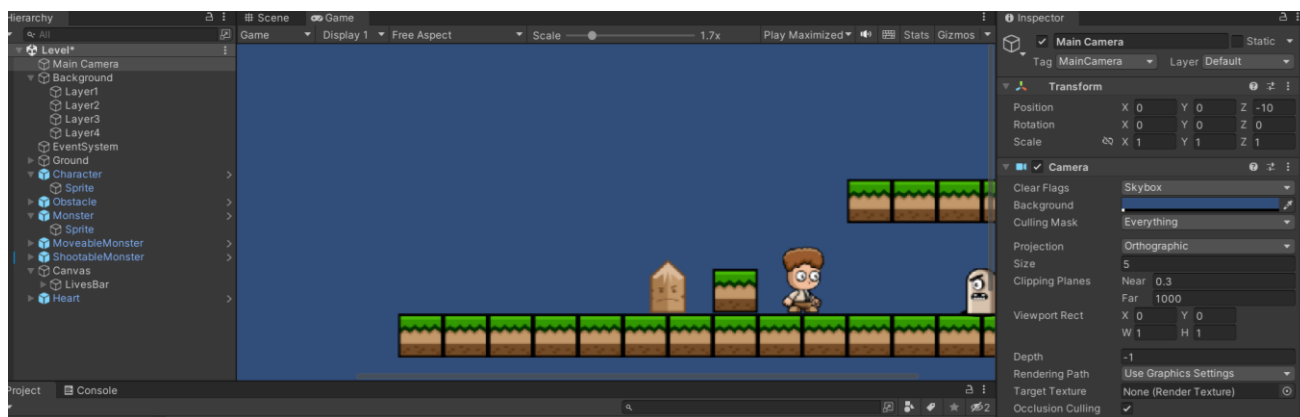


Рис.25

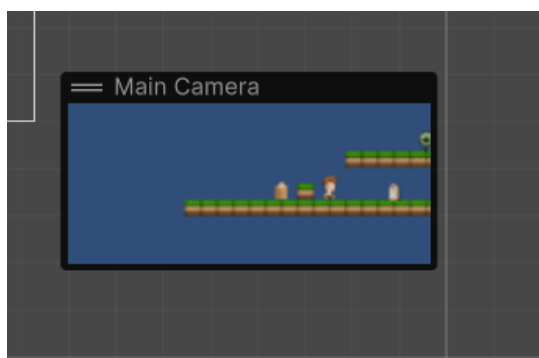


Рис.26



Рис.27 Зображена гра при запуску



Рис.28

Картина тут така, персонаж намагається підійти до монстра щоб пригнути зверху на нього, тим самим вбити його.



Рис.29

Але монстр пускає пулю в персонажа, та в нього мінус 1 життя.



Рис.30

Персонаж не здається підходить, та стрибає зверху, та вбиває його.



Рис.31








Character 	Obstacle 	Monster 	Moveable Monster 
Bullet 	Item 		Shootable Monster 

Рис.32

Сюди входить персонаж головний, монстри, враги, кулі, сердечки(життя).

Монстри або сам персонаж може кидати пулі, щоб знищити життя.

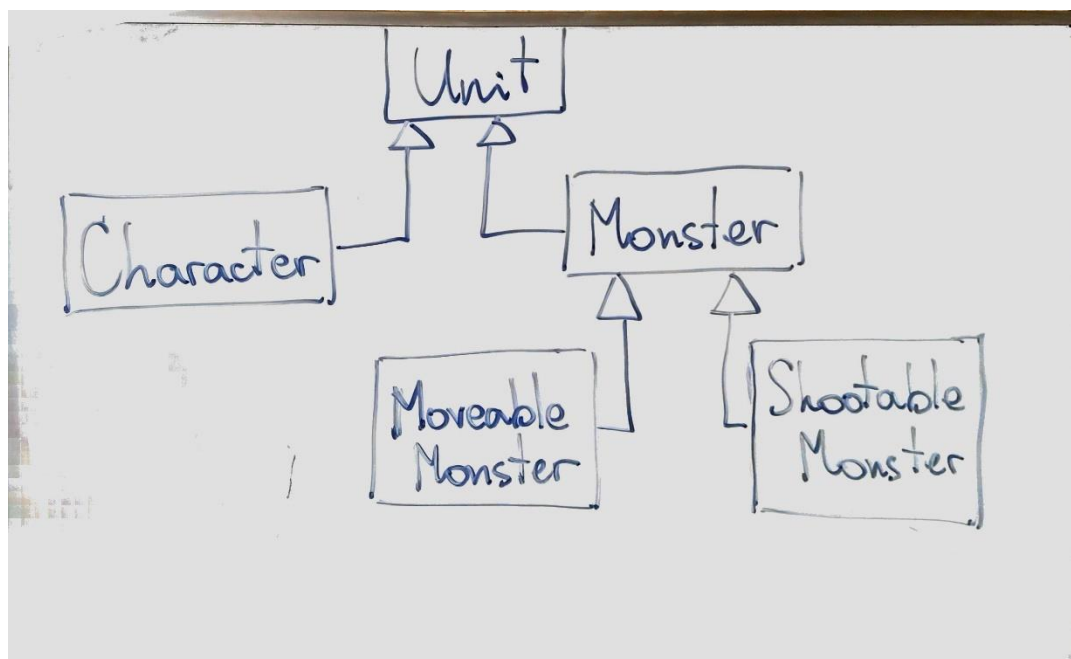


Рис.33

3.6 Висновки до третього розділу

У третьому розділі "Реалізація ігрового додатка в Unity" були розглянуті наступні аспекти:

3.1 Структура ігрового проекту в Unity:

- Була описана загальна структура ігрового проекту в середовищі Unity, включаючи розташування ресурсів, скриптів, ассетів та інших елементів.

3.2 Діаграма класів ігрового додатка:

- Була надана діаграма класів, яка відображає взаємозв'язки між різними класами у ігровому додатку. Це допомагає візуалізувати структуру програмного коду та взаємодію об'єктів у грі.

3.3 Керування вихідним кодом ігрового додатка:

- Було пояснено процес керування вихідним кодом ігрового додатка в середовищі Unity, включаючи створення скриптів, редагування коду, компіляцію та налагодження.

3.4 Функціональне тестування розробленого ігрового додатка:

- Була здійснена перевірка розробленого ігрового додатка на відповідність функціональним вимогам. Це допомагає виявити та виправити можливі помилки, проблеми з функціональністю та забезпечити коректну роботу гри.

3.5 Інструкція користувача ігровим додатком:

- Була надана інструкція користувача, яка описує основні кроки, правила та взаємодію з ігровим додатком. Це допомагає користувачам зрозуміти, як грати та насолоджуватись грою.

У результаті реалізації ігрового додатка в Unity була створена готова гра з належною структурою, функціональністю та інструкцією для користувачів.

ВИСНОВКИ

У даній курсовій роботі була розроблена гра за допомогою середовища Unity.

При розробці гри буде використано мову програмування C# для створення логіки гри, обробки взаємодії з об'єктами та контролю гравця. Unity надає потужні інструменти для розробки графічного інтерфейсу, анімації персонажів та налаштування фізики гри.

В першому розділі курсової роботи було виконано наступні кроки: Представлено загальне визначення теми курсової роботи - розробка ігрового додатка в середовищі Unity. Описано особливості використання мобільних технологій для розробки ігрових додатків в Unity. Проведено огляд аналогічних ігрових додатків, таких як "Temple Run" і "Monster Dash", для виявлення спільних концепцій і можливостей. Сформульовано вимоги до основних функцій мобільного додатка, які дозволять організувати подальшу розробку. Оглянуто інформаційні технології, які використовуються для розробки ігрових додатків в Unity. Зроблено висновки, що узагальнюють отриману інформацію і дають загальну оцінку розглянутих аспектів.

Отже, перший розділ надав важливу основу для подальшої розробки та дослідження ігрового додатка в Unity.

В другому розділі курсової роботи були сформульовані цілі, завдання та інші важливі аспекти проектування ігрового додатка. Було визначено, які напрямки потрібно розвивати, які конкретні задачі потрібно виконати для досягнення цілей. Крім того, були описані користувацькі сценарії та історії, які відображають послідовність дій користувача та можливі ситуації у грі. Також були визначені нефункціональні вимоги, які описують характеристики додатка, що не відносяться до його функціональності. Крім того, було ідентифіковано типового користувача, який має інтерес до гри та бажання досягнути успіху. Всі ці етапи проектування допомагають визначити напрямок розробки, описати сценарії та вимоги, ідентифікувати користувачів та створити навігаційну структуру для додатка.

У третьому розділі були розглянуті наступні аспекти: Описано структуру ігрового проекту в середовищі Unity, включаючи розташування ресурсів, скриптів та інших елементів. Надано діаграму класів, яка відображає взаємозв'язки між класами у грі. Пояснено процес керування вихідним кодом гри, включаючи створення скриптів, редагування коду та налагодження. Здійснено функціональне тестування гри для перевірки відповідності функціональним вимогам. Надано інструкцію користувача, яка описує основні кроки та правила гри.

В результаті реалізації ігрового додатка в Unity була створена гра з належною структурою, функціональністю та інструкцією для користувачів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація_по_Unity_[Електронний ресурс]: Unity 3D. – Режим доступу: <https://unity.com/ru>.(дата звернення 25.05.2023)
2. UNITY C# URL: <https://unity.com/ru/how-to/beginner-game-coding-resources> (дата звернення 26.05.2023).

3.Ігрові рушії UNITY: <https://dev.ua/news/game-engines-list-1657124713> (дата звернення 26.05.2023).

4. Мобільний платформер. URL:

<https://krs.chmnu.edu.ua/jspui/bitstream/123456789/2505/1/4.%20%D0%9A%D0%B8%D1%81%D1%81%D0%B0%20%D0%A4%D0%B0%D1%85%D0%BE%D0%B2%D0%B0%20%D1%87%D0%B0%D1%81%D1%82%D0%B8%D0%BD%D0%B0.pdf> (дата звернення 26.05.2023).

5. Класи та об'єкти URL: <https://metanit.com/sharp/tutorial/3.1.php> (дата звернення 28.05.2023).