Вищий навчальний заклад

«Університет економіки та права «КРОК»
Фаховий коледж

Циклова комісія з інформаційних технологій

Кваліфікаційна робота фахового молодшого бакалавра

на тему Створення гри на платформі Unity, з жанром аркада

Виконав_____
(Підпис)
_____Матвійчук Єгор Сергійович_____
(прізвище, ім'я, по батькові)

_____

Науковий керівник_____
_____Добришин Юрій Євгенович_____
(прізвище, ім'я, по батькові)

_____

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: "До захисту в екзаменаційній комісії ")

Голова циклової комісії _____ _____ _____
(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2025 року

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

Фаховий коледж

Циклова комісія з інформаційних технологій

Спеціальність 121 інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Голова циклової комісії _____ Леонід УВАРОВ

(підпис)

«____» _____ 2025 року

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач освіти Матвійчук Єгор Сергійович

1. Тема роботи Створення гри на платформі Unity, з жанром аркада затверджена наказом по університету

від «___» _____ 202__ р. № _____

2. Термін здачі закінченої роботи «30» травня 2025 року

3. Вихідні дані до роботи:

1. цільова аудиторія – гравці, що полюбляють аркадні ігри;
2. функціональність – запуск гри на різних платформах, управління персонажем, підрахунок балів, збереження прогресу, тощо;
3. технічні вимоги – ігровий рушій Unity, мова програмування C#, підтримка мобільних та десктопних платформ;
4. можливість розширення функціоналу – додавання рівнів, таблиця рекордів, онлайн-режим;
5. аналіз існуючих аркадних ігор і кращих практик їх розробки.

4. Зміст пояснювальної записки:

1. **Розділ 1. Теоретична частина.**
   Аналіз існуючих аркадних ігор, обґрунтування вибору жанру та платформи, огляд інструментів розробки (Unity, C#), сучасні тенденції у створенні 2D-ігор.
2. **Розділ 2. Проєктування та розробка.**
   Формування концепції гри, сценарію, ігрової механіки, інтерфейсу. Розробка структури проєкту в Unity, реалізація логіки гри, анімацій, управління персонажем, інтеграція графічних елементів.
3. **Розділ 3. Експериментальна частина.**
   Тестування гри на різних пристроях, виявлення і виправлення помилок, аналіз відповідності реалізації початковим вимогам, створення інструкції користувача.

Рекомендації щодо покращення продуктивності, UX/UI, можливості подальшого розвитку проєкту.

5. **Перелік графічного матеріалу:**

- Скріншоти існуючих аркадних ігор
- Скріншоти інтерфейсу розробленої гри
- Фрагменти коду та конфігураційні файли
- Інструкції для технічного персоналу та користувачів

Дата видачі завдання «12» лютого 2025 року

Науковий керівник _____ _Добришин Ю. Є_

(підпис) (прізвище, ім'я, по батькові)

Завдання прийняв до виконання _____ __Матвійчук Є. С._

(підпис) (прізвище, ім'я, по батькові)

РЕФЕРАТ

Пояснювальна записка: 52 сторінок, 28 рисунків, 1 таблиця, 6 додатки, 15 джерел.
Об'єкт дослідження – аркадна комп'ютерна гра, розроблена на ігровій платформі Unity.
Мета роботи – розробка комп'ютерної гри жанру аркада на платформі Unity з реалізацією основних ігрових механік, управління персонажем, системою підрахунку балів та підтримкою мультиплатформенності. Кваліфікаційна робота містить результати комплексної розробки ігрового проєкту. Проведено аналіз жанру аркада, досліджено кращі практики розробки ігор, обґрунтовано вибір платформи Unity та мови програмування C#. Розроблено структуру ігрового проєкту, реалізовано логіку гри, анімації, управління персонажем і взаємодію з користувачем. Проведено тестування гри на різних пристроях, виявлено та усунено помилки, проаналізовано відповідність результатів початковим вимогам. Описано організацію ігрових даних та систему збереження прогресу. Розроблені програмні модулі створені в середовищі Unity із застосуванням мови C#, що забезпечує кросплатформенність проєкту та можливість подальшого розширення функціоналу (додавання нових рівнів, таблиць рекордів, онлайн-режиму).Результати роботи можуть бути впроваджені для навчання, розваг та подальшого розвитку в сфері розробки ігор.

Ключові слова: Unity, аркадна гра, 2D-ігри, ігровий рушій, C#, мультиплатформенність, ігровий дизайн, програмування.

ABSTRACT

Explanatory note: 52 pages, 28 figures, 1 table, 6 appendices, 15 sources.
The object of the study is an arcade-style computer game developed on the Unity game engine platform.
The purpose of the work is to develop an arcade game using the Unity engine, implementing key gameplay mechanics, character control, score tracking, and cross-platform support. The explanatory note of the qualification work contains the results of the comprehensive development of a 2D arcade game project. An analysis of existing arcade games and best development practices was conducted, and the choice of Unity as the game engine and C# as the programming language was substantiated. The game concept, gameplay mechanics, interface, and scenario were designed and implemented. The structure of the project was developed in Unity, including logic, animations, character control, and graphical integration. The game was tested on multiple devices, errors were identified and fixed, and the final implementation was analyzed for compliance with the initial requirements. The organization of game data and a system for saving player progress were described. The software components were developed using Unity and the C# programming language, ensuring cross-platform functionality and the potential for further feature expansion (new levels, leaderboard, online mode). The results of the work can be applied in educational, entertainment, and demonstration contexts, and can serve as a foundation for further development in game design.

Keywords: Unity, arcade game, 2D games, game engine, C#, cross-platform, game design, programming.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

1. **AI** – *Artificial Intelligence* – штучний інтелект;
2. **API** – *Application Programming Interface* – програмний інтерфейс прикладного програмування;
3. **C#** – мова програмування C-Sharp;
4. **FPS** – *Frames Per Second* – кількість кадрів за секунду;
5. **HP** – *Health Points* – кількість очок здоров'я персонажа;
6. **HUD** – *Heads-Up Display* – графічний інтерфейс, що відображається поверх ігрового процесу;
7. **NPC** – *Non-Playable Character* – неігровий персонаж, керований системою;
8. **UI** – *User Interface* – інтерфейс користувача;
9. **UX** – *User Experience* – користувацький досвід;
10. **XP** – *Experience Points* – очки досвіду, які набирає персонаж у грі.

ВСТУП

У сучасному світі відеоігри стали однією з найдинамічніших і найвпливовіших галузей цифрової індустрії. Вони виконують не лише функцію розваги, а й слугують потужним інструментом соціалізації, розвитку креативного мислення, когнітивних навичок і навіть навчання. Зокрема, аркадні ігри завдяки простій механіці та динамічному геймплею залишаються популярними серед різних категорій користувачів.

Актуальність завдання. Актуальність теми зумовлена високим попитом на прості у реалізації, але захопливі аркадні ігри, які можуть бути ефективним тренажером для розробника-початківця. Крім того, вивчення процесу створення гри в Unity дозволяє на практиці застосувати знання з програмування, графіки, дизайну інтерфейсу та штучного інтелекту. Це також відповідає сучасним тенденціям інженерії програмного забезпечення та потребам гейм-індустрії.

Мета роботи. Розробити аркадну гру з елементами виживання на ігровому рушії Unity з використанням мови програмування C#, яка поєднує динамічний геймплей, поступове ускладнення та взаємодію гравця з оточенням (руйнування об’єктів, боротьба з ворогами).

Завдання роботи

1. Провести аналіз жанру аркадних ігор та ігрових рушіїв.
2. Обґрунтувати вибір технологій та програмних засобів (Unity, C#).
3. Розробити концепцію гри, описати сценарій і механіки.
4. Реалізувати в Unity основні функції гри: рух, атака, інтерфейс, анімація.
5. Створити візуальне і аудіооформлення гри.
6. Провести тестування гри, виявити помилки та оптимізувати.
7. Оформити інструкцію користувача.
8. Оцінити результат і надати рекомендації щодо подальшого розвитку проєкту.

Об’єкт дослідження. Процес розробки інтерактивних комп’ютерних ігор у середовищі Unity.

Предмет дослідження. Архітектура, ігрова логіка та технічна реалізація аркадної гри з жанром виживання.

Методи дослідження:

- аналіз та узагальнення наукових і технічних джерел;
- порівняльний аналіз рушіїв і підходів до розробки;
- методи програмної реалізації у C#;
- моделювання й тестування ігрових ситуацій;
- експериментальна перевірка працездатності програмного продукту.

Практичне значення одержаних результатів. Розроблена гра може бути використана як навчальний приклад для студентів IT-спеціальностей, демонстрація можливостей рушія Unity або як базова платформа для подальшого розширення в рамках комерційного чи навчального проєкту. Крім того, реалізація гри дозволяє сформувати комплексне розуміння повного циклу проєктування ігрового ПЗ — від ідеї до готового продукту.

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА

1.1. Аналіз існуючих ігор жанру аркада

Жанр аркадних ігор є одним із найдавніших у відеоігровій індустрії, а його еволюція тісно пов’язана з розвитком як апаратного забезпечення, так і програмних засобів. Починаючи з 1970-х років, аркади здобули популярність завдяки простому керуванню, коротким ігровим сесіям і високій динамічності. Вони стали основою для формування звичок гравців та базовим рівнем для вивчення ігрової логіки.

Однією з перших комерційно успішних аркадних ігор вважається **Pong (1972)** — симулятор настільного тенісу, що започаткував окрему нішу спортивних аркад. Іншою знаковою грою стала **Space Invaders (1978)**, яка увела у жанр елементи шутера та нарощувала темп гри відповідно до успіхів гравця.

Серед ігор, що мали визначальний вплив на розвиток жанру, варто відзначити:

- **Pac-Man (1980)** — класика жанру, де гравець керує персонажем, який збирає предмети в лабіринті та уникає ворогів. У грі реалізовано базову штучну поведінку супротивників і логіку прогресії;
- **Donkey Kong (1981)** — одна з перших ігор, яка започаткувала жанр платформерів, поєднуючи аркадну динаміку з елементами головоломки;
- **Tetris (1984)** — еталонна аркадна головоломка, заснована на принципах простоти та логічного мислення. Гра завоювала глобальну аудиторію та стала фундаментом жанру казуальних ігор.

У 2000–2020-х роках з’явилися нові представники аркадних ігор, які адаптували класичні механіки під мобільні платформи. До прикладу:

- **Angry Birds** — поєднання фізики, стрільби та стратегічного мислення;
- **Geometry Dash** — ритмічна аркада з високою складністю і миттєвою реакцією;
- **Super Mario Run** — мобільна адаптація класичного платформера з авто-рухом і простим керуванням одним дотиком.

Характерні риси жанру, які збереглися з часів перших аркад, включають:

- **Швидкий старт гри** — відсутність довгого навчання;
- **Чіткий зворотний зв'язок** — миттєва реакція на дії гравця;
- **Просте управління** — часто достатньо однієї чи двох кнопок;
- **Наростаюча складність** — гра ускладнюється відповідно до прогресу;
- **Мінімалістичний дизайн** — зосередженість на геймплеї, а не графічних деталях.

Для реалізації власного проєкту — гри **«Лісоруб»** — було вирішено використовувати саме аркадну структуру. Вона включає знайомі механіки: рух, атака, збирання ресурсів, протистояння хвилям ворогів. Водночас у грі інтегровано нові ідеї, зокрема — симуляція виживання у ворожому середовищі та стратегічне управління витривалістю персонажа.

Таким чином, аркадний жанр є ідеальним середовищем для створення динамічної гри з мінімальним порогом входу, але потенціалом для глибокого занурення та вдосконалення.

1.2. Кращі практики розробки аркадних ігор

Розробка аркадних ігор, незважаючи на їхню зовнішню простоту, вимагає точного дотримання ряду принципів, які забезпечують комфортний, динамічний та захопливий ігровий процес. Аркадні проєкти орієнтовані на швидке занурення гравця в ігровий світ, тому важливо забезпечити інтуїтивну взаємодію, поступове ускладнення і миттєвий зворотний зв'язок.

До найкращих практик розробки аркадних ігор належать:

1. **Простота та інтуїтивність**. З першої хвилини гри користувач має зрозуміти основні правила та управління. Інтерфейс повинен бути мінімалістичним і логічним. Навчання гравця повинно відбуватися поступово, за допомогою інтерактивних підказок або через сам геймплей.
2. **Прогресивне зростання складності**. Успішні аркадні ігри реалізують поступове підвищення складності. Це може проявлятися у збільшенні кількості ворогів, пришвидшенні їх руху, ускладненні рівнів або обмеженні ресурсів. Така механіка тримає гравця у стані напруження та зацікавленості.

3. **Постійний зворотний зв’язок**. Анімації, звукові ефекти, вібрація, ефекти часток — усе це дає гравцеві розуміння наслідків його дій. Навіть прості елементи, такі як миготіння екрана при отриманні шкоди або звук при зборі бонуса, суттєво впливають на залученість користувача.
4. **Мінімалістичний ігровий дизайн**. Візуальні елементи повинні бути простими й контрастними, аби не відволікати увагу від основних об’єктів взаємодії. Зайві декоративні деталі варто уникати, оскільки вони можуть знижувати продуктивність гри, особливо на мобільних пристроях.
5. **Короткі цикли гри**. Один сеанс гри зазвичай не перевищує кількох хвилин. Це дозволяє гравцеві легко повертатися до гри кілька разів на день, не втрачаючи інтересу. У багатьох аркадах реалізована система рекордів, що мотивує до повторного проходження.
6. **Мотиваційна система**. Використання рівнів, досягнень, балів, рейтингу або внутрішньоігрової валюти забезпечує довготривалу мотивацію гравця. Це дозволяє створити ефект залежності від прогресу, що важливо в мобільних іграх.

Unity та C# як технологічна основа дозволяють ефективно реалізувати всі вищезазначені принципи. Зокрема, Unity має:

- **Потужний редактор анімацій** — для створення плавних та динамічних ефектів;
- **Підтримку кросплатформеності** — розроблену гру можна запускати на ПК, Android, iOS тощо;
- **Гнучкий інтерфейс користувача (UI Toolkit / Canvas)** — що дозволяє створювати як прості, так і складні інтерфейси без потреби у зовнішніх бібліотеках;
- **Бібліотеки для фізики, колізій та візуальних ефектів** — які значно спрощують імплементацію ключових ігрових механік.

У контексті створення гри **«Лісоруб»**, ці принципи дозволили побудувати гру, яка є простою для освоєння, але водночас динамічною та захопливою. Поєднання ефективних ігрових механік, адаптованого інтерфейсу та поступової складності

створює відчуття повного занурення та заохочує гравця повертатися до гри знову і знову.

1.3. Вибір платформи та технологій для розробки (Unity, C#)

Одним із ключових етапів розробки програмного продукту є обґрунтування вибору платформи та технологічного стеку. Для реалізації гри жанру аркада обрано рушій **Unity** у поєднанні з мовою програмування **C#**. Такий вибір є зумовлений рядом технічних, практичних і методичних переваг.

Unity — це багатоплатформовий рушій, який активно використовується для створення 2D та 3D ігор, інтерактивних симуляцій і навчальних додатків. Він забезпечує інтеграцію графіки, фізики, анімації, звукових ефектів і логіки в єдиному середовищі розробки.

Основні переваги Unity:

1. **Підтримка 2D та 3D графіки**. Unity дозволяє створювати як прості 2D-ігри, так і складні тривимірні симуляції. Це робить його універсальним рішенням для реалізації аркад, у яких можна поєднувати обидва види графіки.
2. **Мультимедійні можливості**. Убудовані засоби для роботи з текстурами, анімацією, аудіофайлами, шейдерами та системами часток (Particle System) дають змогу створити насичене візуальне та звукове оформлення гри.
3. **Кросплатформеність**. Unity підтримує компіляцію проєкту для різних платформ — Windows, Android, iOS, macOS, WebGL, а також ігрових консолей (PS, Xbox). Це дозволяє у майбутньому масштабувати гру «Лісоруб» на інші пристрої без переписування основної логіки.
4. **Зручність середовища розробки**. Інтерфейс редактора Unity інтуїтивно зрозумілий і дозволяє швидко створювати сцени, налаштовувати компоненти, додавати фізику та взаємодію між об’єктами. Велика кількість документації, форумів і відеоуроків сприяє самостійному навчанню.

Вибір мови програмування C#

Unity використовує **C#** як основну мову для реалізації сценаріїв (scripts). Серед переваг C#:

- об'єктно-орієнтована структура;
- висока читабельність та легкість у підтримці коду;
- підтримка великим співтовариством;
- інтеграція з візуальними редакторами, такими як Visual Studio.

Мова також надає розробникам гнучкі інструменти для створення ігрової логіки, керування подіями, реалізації анімацій, AI, UI тощо.

Приклад коду в Unity на мові C#

Наведений нижче фрагмент демонструє базову реалізацію руху персонажа за допомогою клавіш керування:

```
void Update()
{
 float move = Input.GetAxis("Horizontal");
 transform.position += new Vector3(move, 0, 0) * speed * Time.deltaTime;
}
```

Цей код щосекунди оновлює позицію персонажа, зчитуючи введення користувача з клавіатури та переміщуючи об’єкт по горизонталі з урахуванням швидкості та часу кадру.

Висновок. Застосування Unity та C# є оптимальним рішенням для розробки аркадної гри «Лісоруб», оскільки забезпечує гнучкість, масштабованість і підтримку всіх необхідних функцій. Таке поєднання дає можливість створити якісний, кросплатформовий ігровий продукт із багатим функціоналом та високою продуктивністю.

1.4 Створення гри

Розробка гри «Лісоруб» відбувалася в декілька етапів: підготовка графічних ресурсів, імпорт моделей, створення ігрової сцени, налаштування камери, реалізація керування персонажем, бойової механіки, візуалізація

об’єктів та обробка взаємодій. Кожен етап вимагав як програмної, так і дизайнерської роботи.

Підготовка моделей і сцени

На початковому етапі було створено чотири 3D-моделі персонажів та ігрових об’єктів за допомогою Blender. Ці моделі було імпортовано до Unity та організовано у відповідні папки з унікальними матеріалами для кожної частини тіла персонажів (рис. 1, 2).

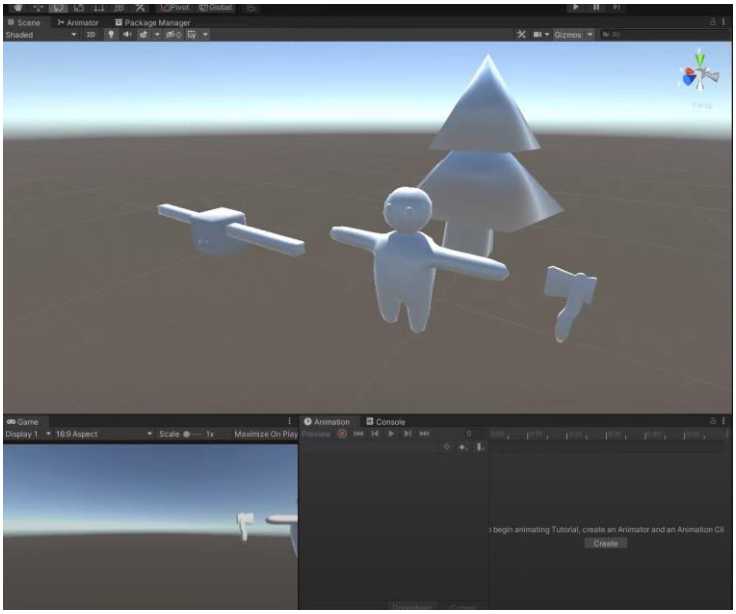


Рисунок 1. Модель сцени гри

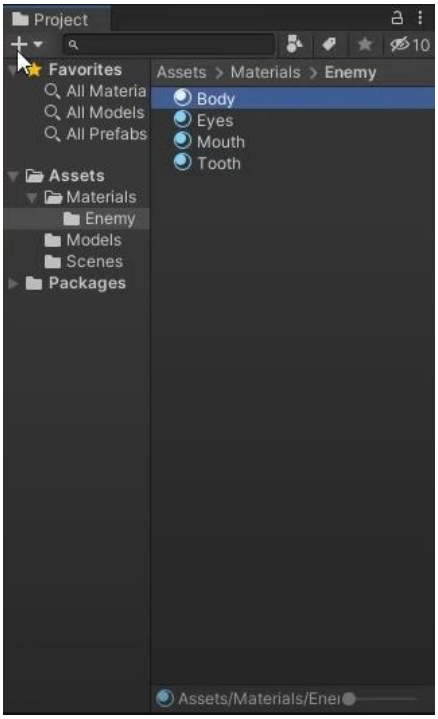


Рисунок 2. Матеріал для кожної частини тіла

Для реалізації генерації дерев на сцені було використано готовий скрипт із відкритого доступу під назвою **Enviro Spawn**. Він дозволив автоматизувати процес розміщення об’єктів дерев у межах певної території, яка налаштовується через інспектор Unity. Було створено порожній об’єкт “environment spawn”, до якого прикріплено скрипт, та визначено зону генерації (рис. 3).

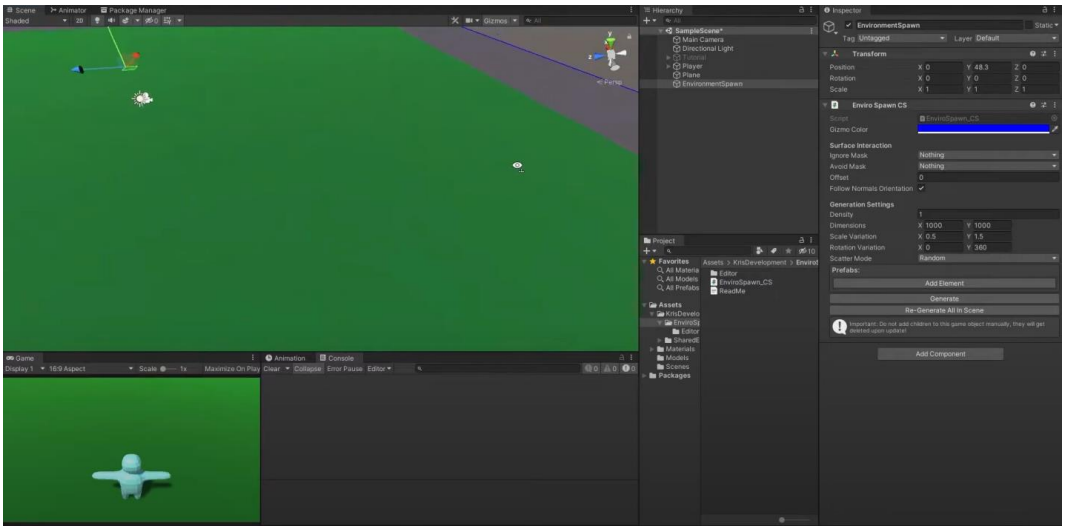


Рисунок 3. Розширення меж спавнера

Для дерева було задано матеріал, який імітує натуральну текстуру кори та листя (рис. 4).

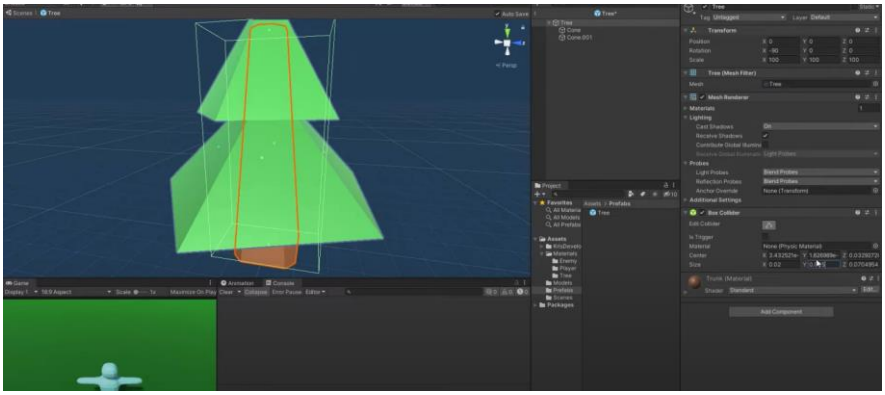


Рисунок 4. Створене дерево

Камера та інтерфейс

Важливою складовою зручності гри є правильне налаштування камери. У проєкті використовується вид зверху (top-down view), що дозволяє гравцеві краще орієнтуватися в просторі та контролювати ситуацію на полі гри (рис. 5).

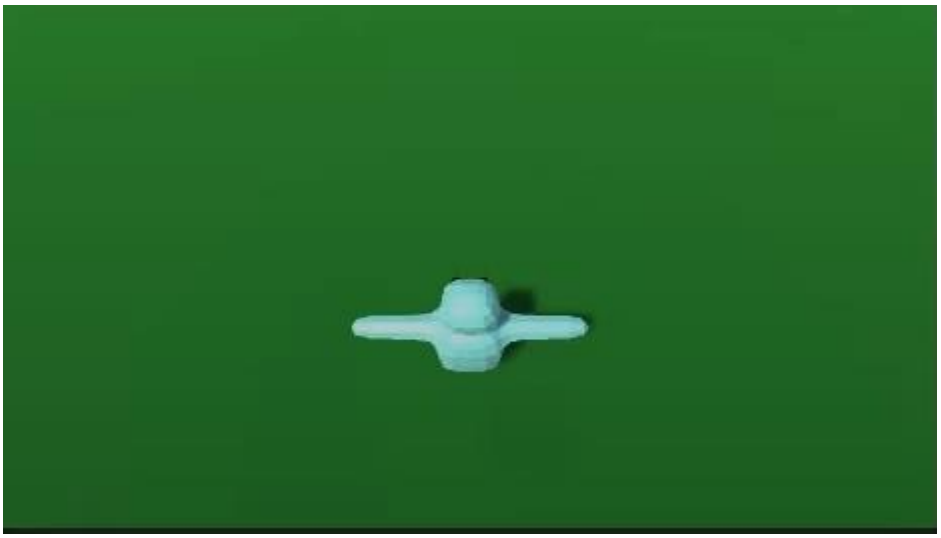


Рисунок 5. Камера з видом зверху

Система керування персонажем

Ключовий скрипт гравця реалізує обробку таких дій, як рух, поворот, атака та отримання шкоди. Нижче подано структуру основних змінних:

- speed — швидкість руху персонажа;
- rotationSpeed — швидкість повороту;
- reloadTime — затримка між атаками;
- startHealth — початковий запас здоров’я;
- ui — посилання на об’єкт UI;
- hitPoint — точка атаки;
- hitRadius — радіус дії атаки;
- _rb, _anim, _canHit, _health — службові змінні (Rigidbody, Animator, логіка удару, поточне здоров’я).

Скрипт включає директиви простору імен:

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

```
{
    [SerializeField] private float speed;
    [SerializeField] private float rotationSpeed;
    [SerializeField] private float reloadTime;
    [SerializeField] private int startHealth;
    [SerializeField] private PlayerUI ui;
    [SerializeField] private Transform hitPoint;
    [SerializeField] private float hitRadius;

    private Rigidbody _rb;
    private Animator _anim;
    private bool _canHit = true;
    private int _health;
```

Рисунок 6. Директиви using

У методі Start() встановлюється початковий стан гравця (рис. 7).

```
private void Start()
{
    _rb = GetComponent<Rigidbody>();
    _anim = GetComponent<Animator>();

    _health = startHealth;
}
```

Рисунок 7. Метод Start

Основний функціонал реалізовано в методі Update():

- обробка вводу з клавіатури;
- обчислення вектору руху;
- зміна орієнтації;
- запуск анімацій;
- перевірка натискання кнопки атаки;
- активація таймера перезарядки (рис. 8).

```
void Update()
{
 float move = Input.GetAxis("Horizontal");
 transform.position += new Vector3(move, 0, 0) * speed * Time.deltaTime;
}
```

```
private void Update()
{
    Vector3 moveInput = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));

    if (moveInput.magnitude > 0.1f)
    {
        Quaternion rotation = Quaternion.LookRotation(moveInput);
        rotation.x = 0;
        rotation.z = 0;
        transform.rotation = Quaternion.Lerp(transform.rotation, rotation, rotationSpeed * Time.deltaTime);
    }

    _anim.SetBool("isWalk", moveInput.magnitude > 0.1f);

    _rb.velocity = moveInput * speed;


    if (Input.GetMouseButton(0) && _canHit == true)
    {
        _anim.SetTrigger("axeHit");
        StartCoroutine(Reload());
    }
}
```

Рисунок 8. Метод Update

Метод Reload() тимчасово вимикає можливість атакувати, запускаючи затримку:

```
IEnumerator Reload()
{
_canHit = false;
yield return new WaitForSeconds(reloadTime);
_canHit = true;
}
```

```
IEnumerator Reload()
{
    _canHit = false;
    yield return new WaitForSeconds(reloadTime);
    _canHit = true;
}
```

Рисунок 9. Функція Reload

Метод GetDamage(int damage) відповідає за зменшення очок здоров'я та перезапуск сцени у випадку смерті персонажа:

```
public void GetDamage(int damage)
{
_health -= damage;
if (_health <= 0)
SceneManager.LoadScene(SceneManager.GetActiveScene().name);
```

```
}
```

```
public void GetDamage(int damage)
{
    _health -= damage; // _health = _health - damage
    ui.SetHealth(_health);

    if (_health <= 0)
    {
        SceneManager.LoadScene(0);
    }
}
```

Рисунок 10. Функція GetDamage

Атака реалізована у методі Hit(), який використовує фізичну перевірку зони атаки (Physics.OverlapSphere) і застосовує логіку ураження дерева або ворога:

```
void Hit()
{
Collider[] hitColliders = Physics.OverlapSphere(hitPoint.position, hitRadius);
foreach (var hitCollider in hitColliders)
{
if (hitCollider.CompareTag("Enemy")) Destroy(hitCollider.gameObject);
if (hitCollider.CompareTag("Tree")) Destroy(hitCollider.gameObject);
}
}
```

```
    private void Hit()
    {
        Collider[] colliders = Physics.OverlapSphere(hitPoint.position, hitRadius);
        for (int i = 0; i < colliders.Length; i++)
        {
            if (colliders[i].TryGetComponent(out Tree tree) && tree.isRespawned)
            {
                ui.TreeCount++;
                tree.Destroy();
            }

            if (colliders[i].TryGetComponent<Enemy>(out _))
            {
                Destroy(colliders[i].gameObject);
            }
        }
    }
}
```

Рисунок 11. Функція Hit

Завдяки реалізованим скриптам, а також правильно налаштованому середовищу Unity, вдалося забезпечити плавну анімацію, чіткий зворотний зв’язок і базову бойову систему, що є типовими елементами аркадних ігор.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ГРИ

2.1. Вимоги до інтерфейсу гри "Лісоруб"

Інтерфейс гри є важливим елементом, що забезпечує зручність зрозумілість гри для користувачів. У нашій грі **Лісоруб** інтерфейс повинен бути інтуїтивно зрозумілим, мінімалістичним, при цьому виконувати роль організації ігрових елементів, таких як відображення здоров’я, витривалості, часу та прогресу гравця. Важливо, щоб інтерфейс не відволікав гравця від основного процесу гри, а лише доповнював його.

Основні вимоги до інтерфейсу:

1. **Відображення здоров'я (HP)**: Гравець повинен бачити поточне значення здоров’я. Для цього можна використати горизонтальну панель, що показує кількість здоров’я візуально, в залежності від кількості HP. Панель повинна змінювати колір залежно від рівня здоров'я (наприклад, зелений при високому рівні здоров'я, жовтий при середньому і червоний при низькому).
2. **Відображення витривалості (Stamina)**: Гравець витрачатиме енергію при виконанні дій (наприклад, рубці дерев та боротьбі з монстрами). Витривалість відображатиметься на окремій панелі, що дозволить користувачеві контролювати рівень енергії. Панель витривалості також повинна змінювати колір у міру зниження рівня енергії.
3. **Індикатор хвилі**: Гравець повинен мати можливість побачити, скільки хвиль монстрів ще залишилось до завершення поточного етапу. Індикатор хвилі має бути чітким і легко зрозумілим, з кількістю хвиль, що залишились, або прогресом візуально відображеним через лінійку чи шкалу.
4. **Панель статистики**: Враховуючи жанр гри, корисно мати відображення таких елементів, як кількість вбитих монстрів, час, що пройшов від початку гри, а також інші досягнення, наприклад, кількість зрубаних дерев. Ці елементи повинні бути доступними для швидкого перегляду, не заважаючи гравцеві під час гри.

5. **Інтерактивні елементи**: Меню гри, кнопки для паузи, налаштувань та виходу з гри повинні бути чітко видимими, але не заважати основному процесу. Кнопки мають бути адаптовані до різних платформ, зокрема до мобільних пристроїв, де вони повинні бути зручними для натискання, з чіткими іконками та текстами.
6. **Анімації та ефекти**: Інтерфейс має містити динамічні елементи, такі як анімації для зміни стану здоров’я, витривалості та інші візуальні ефекти, що підвищують емоційний досвід гравця. Наприклад, анімація зменшення здоров'я при атаці монстрів або ефекти, що з’являються під час досягнення важливих моментів (наприклад, завершення хвилі монстрів).

Приклад структури інтерфейсу:

- Вгорі екрану буде розміщена панель здоров’я та витривалості, з чітким відображенням їх рівня.
- Знизу екрану буде знаходитися індикатор прогресу, що показує кількість хвиль монстрів, які залишилися до кінця поточного етапу, а також додаткові елементи статистики.
- Зліва буде знаходитися кнопка для паузи та доступ до налаштувань. Важливо, щоб ці елементи не перекривали основний ігровий процес і не заважали огляду сцени.
- Всі інші елементи інтерфейсу мають бути мінімалістичними, щоб не забирати зайвого простору на екрані, але залишати достатньо місця для відображення основної гри.

Інтерфейс гри буде адаптований для різних платформ, зокрема для мобільних пристроїв, де елементи інтерфейсу повинні бути зручними для натискання. На мобільних пристроях інтерфейс може бути компактнішим, а кнопки — більшими та зручнішими для управління однією рукою.

2.2. Розробка концепції та сценарію гри

Концепція гри **Лісоруб** полягає в тому, щоб створити динамічну ігрову атмосферу з елементами виживання. Гравець виступає в ролі лісоруба, який бореться за своє життя в небезпечному лісі, де на нього чекають монстри та інші

загрози. Головною метою є вижити, рубаючи дерева для отримання ресурсів і захищаючись від хвиль монстрів, що з’являються з кожним рівнем.

Сюжет:

Гравець потрапляє в ліс, де навколо повно магічних чудовиськ. Ліс періодично атакує хвилями монстрів, і гравець має вигнати їх, використовуючи сокиру та інші ресурси, при цьому забезпечуючи своє виживання до появи нової хвилі монстрів. Рубаючи дерева для отримання матеріалів, гравець також збирає ресурси для зміцнення своєї оборони та підготовки до наступних атак.

Основні етапи гри:

1. **Початковий рівень**: Гравець починає гру з базовим набором можливостей: сокира для рубки дерев і базове здоров’я. Перші хвилі монстрів будуть легкими, щоб дозволити гравцеві освоїти основні механіки гри, як використання сокири для збору ресурсів та боротьбу з початковими монстрами. У цей період важливо навчитися правильно балансувати між рубкою дерев і захистом від ворогів.
2. **Середні рівні**: З кожною новою хвилею монстри стають сильнішими, а кількість їх типів збільшується. Вони можуть мати особливі здатності, такі як дальнобійні атаки або швидкість, що змушує гравця використовувати різні стратегії. Крім того, на цих етапах важливо уважно стежити за станом здоров’я та витривалості, щоб вчасно відновлювати сили. Рубання дерев стає також важливим для забезпечення ресурсами для створення нових інструментів або захисту.
3. **Фінальні рівні**: На фінальних рівнях гравець стикається з найскладнішими хвилями монстрів, які вимагатимуть повного використання всіх ресурсів, накопичених протягом гри, а також стратегічного підходу. Важливо правильно управляти запасами енергії та здоров’я, а також своєчасно застосовувати всі здобуті навички для перемоги.

Сюжетний розвиток:

У ході гри ліс стане різноманітнішим: з’являтимуться нові локації, які додають не лише естетичної привабливості, але й нові можливості та загрози для гравця. У грі буде ряд NPC (неігрових персонажів), з якими гравець може взаємодіяти. Вони можуть дати корисні поради, поділитися знаннями чи продати ресурси, що допомагають пройти важкі етапи гри. Ці NPC можуть мати свою історію і допомагати або ставати перепоною для подальшого просування в сюжеті.

2.3. Розробка алгоритму роботи гри (механіка, логіка гри)

Алгоритм гри має бути побудований таким чином, щоб гра була динамічною та не передбачуваною для гравця. Механіка гри ґрунтується на наступних принципах:

1. **Рубка дерев:**

Гравець використовує сокиру для рубки дерев. Кожен удар по дереву зменшує його здоров’я. Після того, як дерево зрубається, гравець отримує ресурси, які можна використати для укріплення оборони або відновлення здоров'я.

Рисунок 1. Процес рубки дерева

2. **Боротьба з монстрами:**

Монстри з’являються на певних етапах гри. Гравець повинен за допомогою сокири або інших засобів боротьби знищити їх. Кожен знищений монстр приносить досвід і дає ресурси.

3. **Витривалість:**

Виконання дій, таких як рубка дерев і боротьба з монстрами, забирає витривалість. Коли витривалість вичерпується, гравець не може активно атакувати і змушений чекати на відновлення енергії.

4. **Рівень складності:**

З кожною новою хвилею монстрів складність гри збільшується. Для цього буде використано алгоритм, який визначає кількість монстрів на кожному рівні в залежності від прогресу гравця.

2.4. Анімація в Unity

Анімація в Unity – це процес створення і налаштування руху та поведінки об'єктів у грі. Unity надає різні інструменти та системи для роботи з анімаціями, що дозволяє розробникам створювати реалістичні та складні анімаційні ефекти.

1. **Animator та Animator Controller:**
    - **Animator** — компонент, який додається до об'єкта і керує його анімаціями.
    - **Animator Controller** — файл, який визначає, які анімації повинні виконуватись і за яких умов.

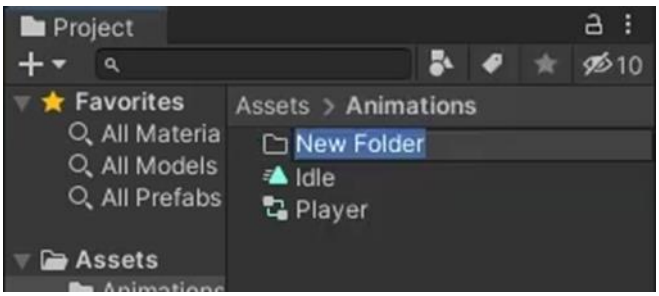


Рисунок 2. Папка з анімаціями в Unity

2. **Animation Clips:**
3. Це окремі файли, які містять дані про анімації. Вони можуть бути створені в Unity або імпортовані з зовнішніх програм.
4. **Animation Window:**
5. Інтерфейс для створення та редагування анімацій. Тут можна додавати ключові кадри для різних параметрів.
6. **Mecanim:**

Потужна система анімації, яка дозволяє створювати анімаційні дерева для персонажів і плавно змішувати анімації.

7. **Root Motion:**

Техніка, що дозволяє анімаціям персонажів контролювати їх пересування у просторі.

2.5. Рух персонажа та налаштування анімацій

Після налаштування скриптів руху персонажа потрібно створити анімацію рухів.

1. **Створення анімації стояння (Idle):**
2. Для того, щоб персонаж стояв реалістично, налаштували позицію рук.

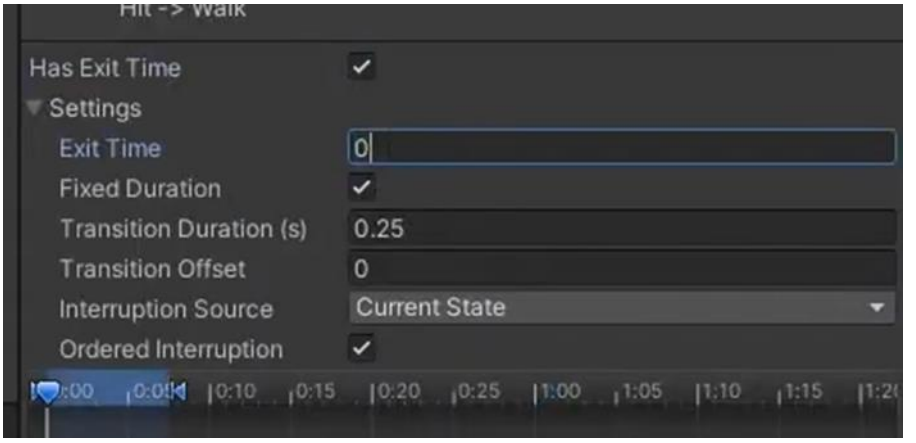


Рисунок 3. Налаштування анімації "idle"

3. **Налаштування анімації ходьби:**
4. Для анімації ходьби потрібно налаштувати рухи ніг, рук, спини та тулуба.

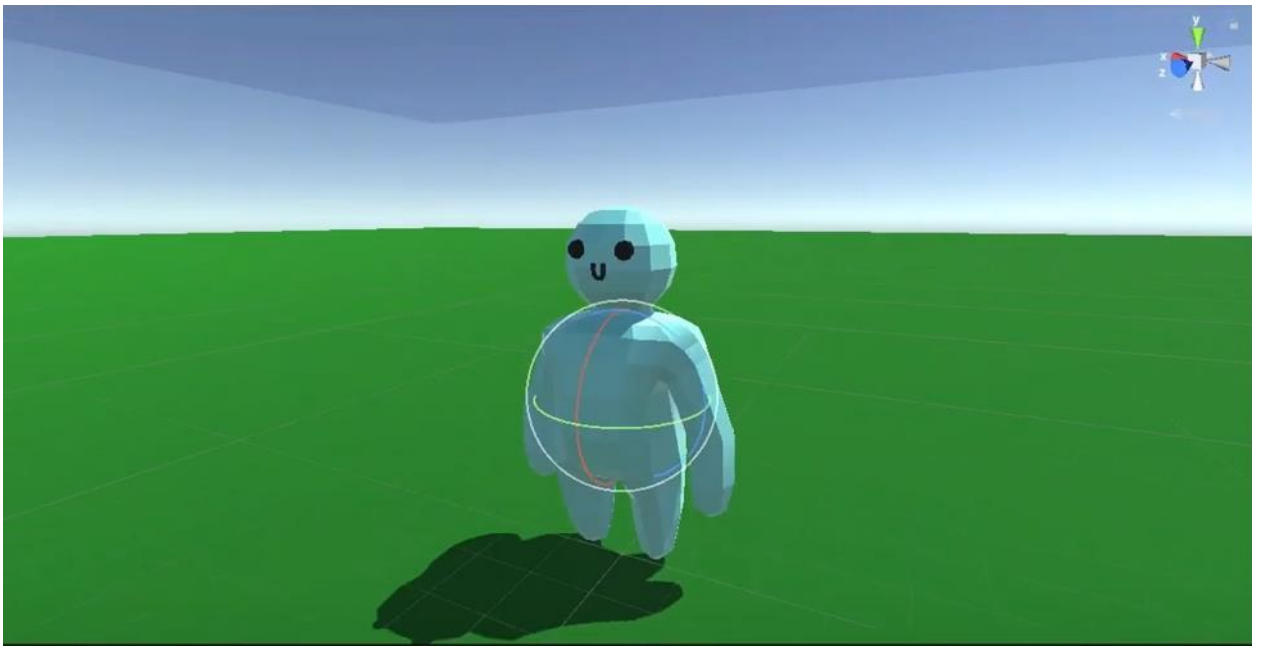


Рисунок 4. Налаштування анімації ходьби

5. **Налаштування анімації удару сокирою:**
6. Зміна напрямку руки для імітації удару.

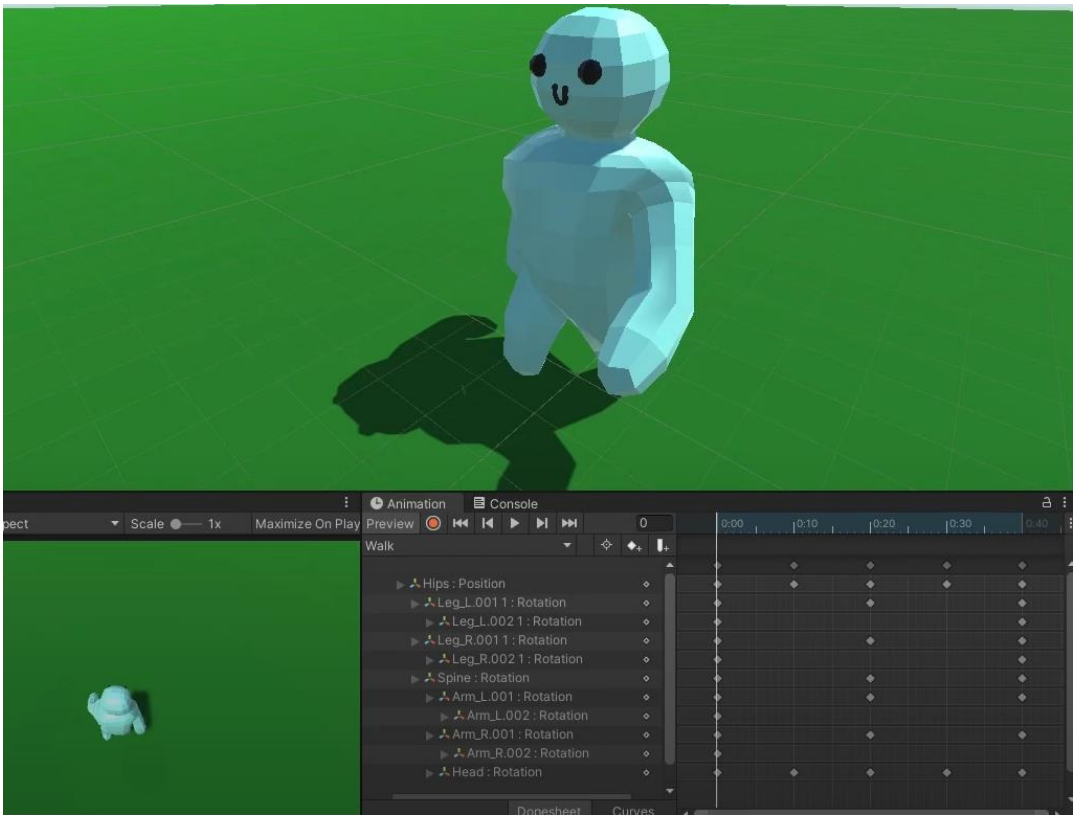


Рисунок 5. Налаштування анімації удару сокирою

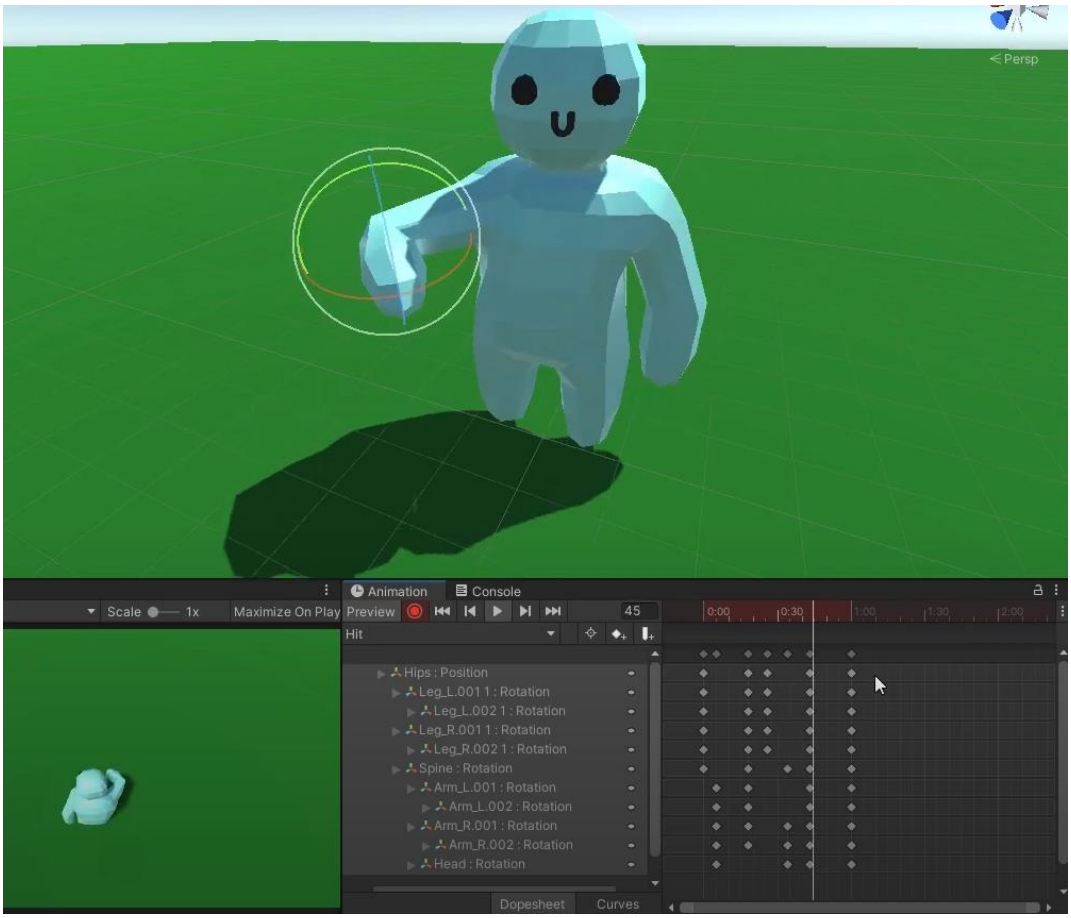


Рисунок 5.1. Налаштування анімації удару сокирою

2.6. Інтеграція графіки та анімацій

Інтеграція графіки та анімацій є важливою частиною створення візуально привабливої гри. Для цього використовуються такі елементи:

1. **2D графіка:**
2. Гра буде розроблена в 2D стилі, що є типовим для аркадних ігор. Основні елементи – персонажі (лісоруб та монстри), дерева, фон і ефекти.
3. **Анімації:**
4. Анімації персонажа будуть включати рухи при рубці дерев, атаки монстрів, а також інші дії.
5. **Інтеграція в Unity:**
6. Всі графічні елементи будуть імпортовані як спрайти в Unity. Використовуючи анімаційну систему Unity, ці спрайти будуть анімовані для створення динамічних рухів персонажа.

РОЗДІЛ 3. ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

3.1. Тестування гри на різних платформах (ПК, мобільні пристрої)

Тестування гри є важливим етапом розробки, оскільки дозволяє оцінити її продуктивність, стабільність та відповідність поставленим вимогам. Гра *Лісоруб* була протестована на двох основних платформах: персональні комп'ютери (Windows) та мобільні пристрої (Android).

Тестування проводилось у кілька етапів:

- **Функціональне тестування** — перевірка основних механік гри (рух, атака, взаємодія з об'єктами).
- **Тестування продуктивності** — оцінка FPS (кадрів за секунду), часу завантаження рівнів та загальної стабільності.
- **Юзабіліті-тестування** — оцінка зручності керування та взаємодії з інтерфейсом.

Результати тестування:

1. **Тестування на ПК:**
    - Гра стабільно працює при роздільній здатності Full HD (1920x1080).
    - Частота кадрів (FPS) залишається на рівні 60+ навіть при великій кількості об'єктів на екрані.
    - Відсутні критичні баги, за винятком рідкісних проблем із відображенням анімацій персонажів.
2. **Тестування на мобільних пристроях:**
    - На потужних смартфонах (Snapdragon 865+) гра працює без лагів із FPS ~50-60.
    - На слабших пристроях (Snapdragon 450) помітне зниження продуктивності (~30 FPS) при великій кількості монстрів.
    - Довший час завантаження рівнів (~5-7 секунд) у порівнянні з ПК.
    - Виявлені проблеми з адаптацією управління під сенсорний екран (недостатньо чутливий джойстик).

Висновки:

Тестування показало, що гра добре працює на ПК та потужних мобільних пристроях. Однак для оптимізації мобільної версії необхідно:

- Зменшити кількість анімаційних ефектів на слабких пристроях.
- Оптимізувати завантаження ресурсів гри.
- Доопрацювати сенсорне управління.

3.2. Аналіз помилок та їх усунення

Основні помилки та способи їх виправлення:

1. **Лаги при великій кількості монстрів**
    - **Причина**: надмірне використання ресурсів (велика кількість об’єктів одночасно на сцені).
    - **Вирішення**: застосування об’єктного пулінгу (Object Pooling) для зменшення навантаження. Об’єктний пулінг дозволяє повторно використовувати вже існуючі об’єкти замість їхнього створення та знищення, що суттєво зменшує витрати ресурсів.
2. **Некоректне відображення інтерфейсу на мобільних пристроях**
    - **Причина**: невірне масштабування UI-елементів у Canvas Unity.
    - **Вирішення**: використання адаптивного дизайну з налаштуванням *UI Scale Mode* на "Scale with Screen Size", що дозволяє елементам інтерфейсу коректно масштабується залежно від розміру екрану різних пристроїв.
3. **Зависання гри при переході між рівнями**
    - **Причина**: завантаження всіх ресурсів одночасно.
    - **Вирішення**: використання асинхронного завантаження сцен за допомогою *SceneManager.LoadSceneAsync*, що дозволяє завантажувати ресурси поступово, покращуючи загальну стабільність гри та скорочуючи час завантаження.
4. **Проблеми з сенсорним управлінням на мобільних пристроях**
    - **Причина**: погана чутливість віртуального джойстика.

- **Вирішення**: налаштування області торкання та підвищення чутливості віртуального джойстика. Це дозволяє поліпшити точність і комфорт управління, особливо на мобільних пристроях з різними розмірами екрану.

Ці виправлення допомогли значно покращити стабільність і зручність гри, особливо на мобільних пристроях, де оптимізація є критично важливою для забезпечення хорошого користувацького досвіду.

3.3. Оцінка відповідності функціоналу поставленим завданням

Функціональні вимоги та їх реалізація:

Вимога	Виконання
Рубка дерев	Реалізовано, механіка працює стабільно
Атака монстрів	Реалізовано, анімації атак відображаються правильно
Відображення здоров’я та витривалості	Панель HP та Stamina працює коректно
Система хвиль монстрів	Монстри спавняться згідно з рівнями складності
Оптимізація для мобільних пристроїв	Потребує покращення продуктивності на слабких мобільних пристроях

Висновок. Гра **Лісоруб** успішно реалізує основні заявлені функції, такі як механіка рубки дерев, атака монстрів, відображення здоров'я та витривалості, а також система хвиль монстрів. Однак, для досягнення оптимального користувацького досвіду на мобільних пристроях, необхідно покращити продуктивність, особливо для слабших пристроїв, шляхом додаткової оптимізації ресурсів та управління.

3.4. Інструкція для користувачів гри "Лісоруб"

1. **Запуск гри:**

- **На ПК**: Запустіть файл **Lisorub.exe**.
- **На мобільному пристрої**: Відкрийте додаток **Lisorub** після встановлення.

2. **Основне управління:**

- **Переміщення**:
  - **ПК**: Використовуйте стрілки на клавіатурі.
  - **Мобільні пристрої**: Використовуйте віртуальний джойстик.
- **Атака**:
  - **ПК**: Ліва кнопка миші.
  - **Мобільні пристрої**: Кнопка «Атака».
- **Рубка дерев**: Наведіть персонажа на дерево і натисніть «Атака».

3. **Основні цілі гри:**

- Вижити, захищаючись від хвиль монстрів.
- Рубати дерева для отримання ресурсів.
- Відстежувати стан здоров'я та витривалості.

3.5. Рекомендації щодо оптимізації продуктивності та юзабіліті

Оптимізація продуктивності:

1. **Використання Object Pooling**:
   - Для покращення продуктивності була реалізована система Object Pooling для управління ворогами та іншими об'єктами на сцені. Це дозволяє ефективно повторно використовувати об'єкти, замість їх постійного створення і знищення, що зменшує навантаження на систему.
2. **Зменшення кількості частинок**:
   - Зменшення кількості частинок в ефектах вогню, ударів та вибухів дозволило значно оптимізувати продуктивність гри, особливо на мобільних пристроях із обмеженими ресурсами. Вибір менш складних ефектів також покращив стабільність.
3. **Використання Sprite Atlas**:
   - Всі спрайти були згруповані в один або кілька **Sprite Atlas**, що дозволило знизити кількість текстур, які завантажуються окремо, і зменшити навантаження на пам'ять пристрою.

Покращення юзабіліті:

1. **Оптимізація інтерфейсу гри для мобільних пристроїв**:
    - Кнопки та інші інтерфейсні елементи були збільшені, щоб зробити їх більш зручними для натискання на мобільних пристроях, забезпечуючи комфортне управління навіть для користувачів з великими пальцями.
2. **Покращення сенсорного керування**:
    - Внесені корективи в розміщення елементів інтерфейсу для різних роздільних здатностей екранів, що покращило відчуття контролю та зручність використання на різних мобільних пристроях.
3. **Візуальна індикація втоми персонажа**:
    - Додано ефект зміни кольору екрану при низькому рівні витривалості персонажа, що дозволяє гравцю наочно відслідковувати стан персонажа, не зважаючи на відсутність явного числового індикатора.

Ці оптимізації не тільки покращують продуктивність гри, але й роблять її більш зручною та доступною для користувачів на різних пристроях.

ВИСНОВКИ

У процесі виконання дипломного проєкту було створено аркадну гру **«Лісоруб»**, яка поєднує елементи виживання, динамічного екшену та стратегічного планування. Протягом роботи було проведено детальний аналіз існуючих ігор жанру, визначено кращі практики розробки та обґрунтовано вибір технологій для створення гри, таких як **Unity** та **C#**.

Основні досягнення включають:

1. **Розробка концепції та сценарію гри:**
    - Визначено ігрову механіку, головні цілі та завдання, що забезпечили чітке спрямування розвитку проєкту.
    - Створена система хвиль монстрів, що додає динамічності та постійний виклик для гравця.
2. **Реалізація основної механіки гри:**
    - Забезпечено механіку рубки дерев і отримання ресурсів, що є ключовою частиною ігрового процесу.
    - Реалізовано захист від хвиль монстрів із використанням зброї, що розширює можливості гравця в битвах.
    - Відображення показників здоров’я та витривалості дозволяє гравцеві контролювати стан персонажа.
3. **Розробка графіки та анімацій:**
    - Інтеграція візуальних ефектів дозволила покращити атмосферу гри, зробивши її більш захоплюючою.
    - Налаштовані плавні анімації для персонажів та ворогів, що сприяло природному руху та бойовим взаємодіям.
4. **Оптимізація гри для ПК та мобільних платформ:**
    - Використано метод **Object Pooling** для ефективного управління об'єктами в грі, що покращує продуктивність.
    - Внесені зміни для підвищення продуктивності на слабших пристроях, зокрема оптимізація графічних ефектів та використання Sprite Atlas.

5. **Тестування та виправлення основних помилок:**
    - Виявлені проблеми з FPS на мобільних пристроях були частково оптимізовані, що дозволило досягти стабільної роботи на більшості пристроїв.
    - Виправлено помилки в відображенні інтерфейсу на різних роздільних здатностях екрану.
    - Поліпшено чутливість сенсорного управління для мобільних пристроїв, що покращило взаємодію з грою.

Перспективи розвитку проєкту

Гра «Лісоруб» має значний потенціал для подальшого розвитку та покращення. Рекомендується:

- Додати нові рівні та противників для збільшення різноманітності ігрового процесу.
- Розширити систему ресурсів, зокрема додавши можливість будівництва укріплень або створення нових інструментів.
- Оптимізувати мобільну версію для кращої продуктивності на бюджетних пристроях.
- Реалізувати мультиплеєрний режим для можливості гравцям співпрацювати та виживати разом.

Висновок. Розробка гри «Лісоруб» стала успішним застосуванням теоретичних знань та практичних навичок у галузі розробки ігор. Завдяки ефективному використанню обраних технологій було досягнуто поставлених цілей, ігра забезпечує захоплюючий та стабільний ігровий досвід. Попри вже досягнуті результати, проект має великий потенціал для подальшого вдосконалення, що дозволить розширити його можливості та підвищити якість гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

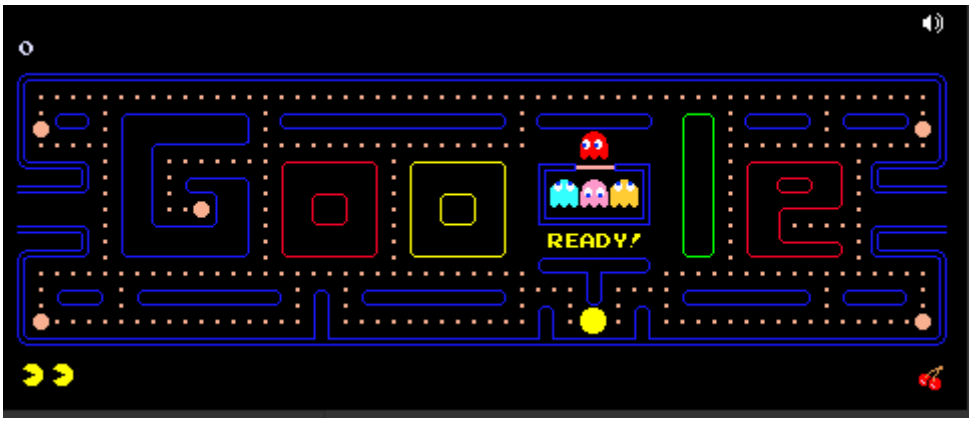
1. Unity Technologies. Unity User Manual [Електронний ресурс]. URL: https://docs.unity3d.com/ (дата звернення: 07.05.2025).
2. Microsoft. C# Programming Guide [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 07.05.2025).
3. Rabin, S. Game AI Pro: Collected Wisdom of Game AI Professionals. Boca Raton: A K Peters/CRC Press, 2013. 480 с.
4. Nystrom, R. Game Programming Patterns. Genever Benning, 2014. 354 с.
5. Gregory, J. Game Engine Architecture. Boca Raton: CRC Press, 2018. 1024 с.
6. McShaffry, M. Game Coding Complete. Boston: Cengage Learning PTR, 2012. 872 с.
7. Stroustrup, B. The C++ Programming Language. 4th ed. Boston: Addison-Wesley, 2013. 1366 с.
8. Unity Technologies. Game Development Best Practices. Unity Blog [Електронний ресурс]. URL: https://blog.unity.com/ (дата звернення: 07.05.2025).
9. Stack Exchange Inc. Game Development Discussions & Best Practices. Stack Overflow [Електронний ресурс]. URL: https://gamedev.stackexchange.com/ (дата звернення: 07.05.2025).
10. Udemy. Unity Game Development for Beginners [Електронний ресурс]. URL: https://www.udemy.com/ (дата звернення: 07.05.2025).
11. Unity Technologies. Технічна документація та поради розробників. Офіційний форум Unity [Електронний ресурс]. URL: https://forum.unity.com/ (дата звернення: 07.05.2025).
12. Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994. 416 с.
13. Wenderlich, R. 2D Game Development with Unity. New York: Razeware LLC, 2020. 340 с.
14. GitHub, Inc. Game Development Repositories [Електронний ресурс]. URL: https://github.com/topics/unity (дата звернення: 07.05.2025).

15. Google. Приклади аркадних ігор. Google Play Store [Електронний ресурс]. URL: https://play.google.com/store/apps/category/ARCADE (дата звернення: 07.05.2025).

ДОДАТКИ

А. Скриншоти існуючих аркадних ігор

1. Pac-Man



2. Space Invaders

3. Donkey Kong

4. Jetpack Joyride

5. Subway Surfers

6. Geometry Dash

7. **Crossy Road**

8. 

Б. Скриншоти інтерфейсу розробленої гри

Рисунок 1. Головне меню

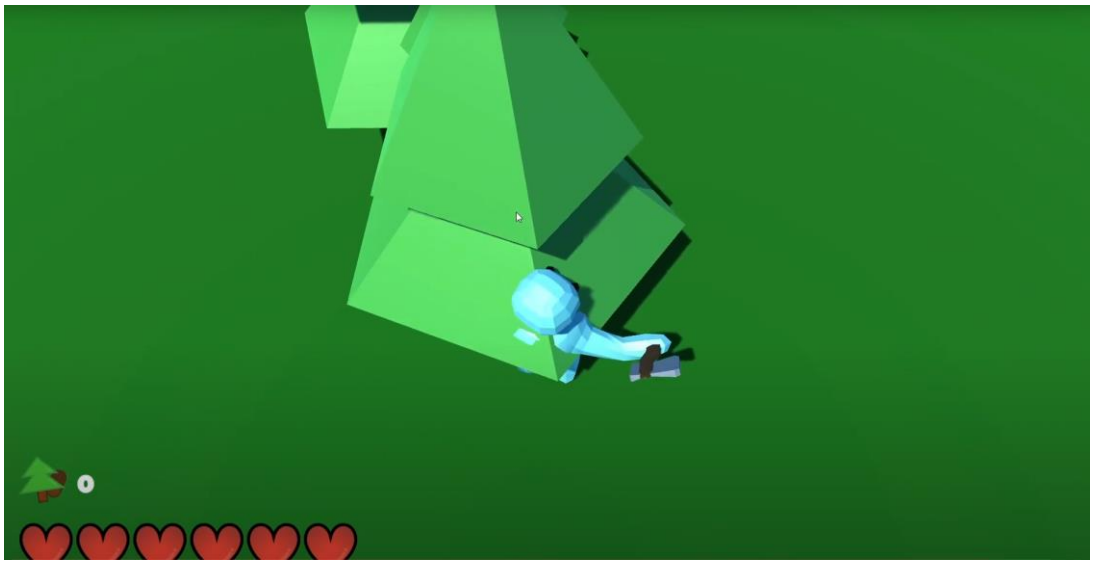


Рисунок 2. Процес рубки дерев

Рисунок 3. Результат рубки дерев (в лівому нижньому екрані)

Рисунок 4. Ворог підходить до головного персонажа

Рисунок 5. Ворог б’є головного персонажа

Рисунок 6. Вбивство ворога

Рисунок 7. Результат смерті персонажа

Рисунок 8. Початковий екран

В. ДОДАТОК А — Лістинг скрипту Enemy.cs

```
using System;
using System.Collections;
using UnityEngine;
using UnityEngine.AI;

namespace Tutorial
{
   public class Enemy : MonoBehaviour
   {
      [SerializeField] private Transform hitPoint;
      [SerializeField] private float radius;
      [SerializeField] private float reloadTime;
      [SerializeField] private int damage;

      private NavMeshAgent _agent;
      private Transform _target;
      private Animator _anim;
      private bool _canHit;
      private bool _isReloaded = true;


      private void Start()
      {
         _anim = GetComponent<Animator>();
         _agent = GetComponent<NavMeshAgent>();
         _target = FindObjectOfType<Player>().transform;
      }

      private void Update()
      {
         _agent.SetDestination(_target.position);

         _anim.SetBool("isWalk", _agent.velocity.magnitude > 0.1f);

         bool isDistance = Vector3.Distance(transform.position, _target.position) <
_agent.stoppingDistance;
         _canHit = isDistance;

         if (_canHit && _isReloaded)
         {
            Collider[] colliders = Physics.OverlapSphere(hitPoint.position, radius);
            for (int i = 0; i < colliders.Length; i++)
            {
```

```
                if (colliders[i].TryGetComponent(out Player player))
                {
                    _anim.SetTrigger("hit");
                    player.GetDamage(damage);
                }
            }

            StartCoroutine(Reload());
        }
    }

    IEnumerator Reload()
    {
        _isReloaded = false;
        yield return new WaitForSeconds(reloadTime);
        _isReloaded = true;
    }
  }
}
```

Г. ДОДАТОК В — Лістинг скрипту Player.cs

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

namespace Tutorial
{
  public class Player : MonoBehaviour
  {
    [SerializeField] private float speed;
    [SerializeField] private float rotationSpeed;
    [SerializeField] private float reloadTime;
    [SerializeField] private int startHealth;
    [SerializeField] private PlayerUI ui;
    [SerializeField] private Transform hitPoint;
    [SerializeField] private float hitRadius;

    private Rigidbody _rb;
    private Animator _anim;
    private bool _canHit = true;
    private int _health;
```

```
    private void Start()
    {
        _rb = GetComponent<Rigidbody>();
        _anim = GetComponent<Animator>();

        _health = startHealth;
    }

    private void Update()
    {
        Vector3 moveInput = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));

        if (moveInput.magnitude > 0.1f)
        {
            Quaternion rotation = Quaternion.LookRotation(moveInput);
            rotation.x = 0;
            rotation.z = 0;
            transform.rotation = Quaternion.Lerp(transform.rotation, rotation, rotationSpeed * Time.deltaTime);
        }

        _anim.SetBool("isWalk", moveInput.magnitude > 0.1f);

        _rb.velocity = moveInput * speed;


        if (Input.GetMouseButton(0) && _canHit == true)
        {
            _anim.SetTrigger("axeHit");
            StartCoroutine(Reload());
        }
    }

    IEnumerator Reload()
    {
        _canHit = false;
        yield return new WaitForSeconds(reloadTime);
        _canHit = true;
    }

    public void GetDamage(int damage)
```

```
        {
            _health -= damage; // _health = _health - damage
            ui.SetHealth(_health);

            if (_health <= 0)
            {
                SceneManager.LoadScene(0);
            }
        }

        private void Hit()
        {
            Collider[] colliders = Physics.OverlapSphere(hitPoint.position, hitRadius);
            for (int i = 0; i < colliders.Length; i++)
            {
                if (colliders[i].TryGetComponent(out Tree tree) && tree.isRespawned)
                {
                    ui.TreeCount++;
                    tree.Destroy();
                }

                if (colliders[i].TryGetComponent<Enemy>(out _))
                {
                    Destroy(colliders[i].gameObject);
                }
            }
        }
    }
}
```

Д. ДОДАТОК С — Лістинг скрипту EnemySpawner.cs

```
 using System;
using System.Collections;
using UnityEngine;
using Random = UnityEngine.Random;

namespace Tutorial
{
    public class EnemySpawner : MonoBehaviour
    {
        [SerializeField] private Transform[] spawnPoints;
        [SerializeField] private GameObject enemyPrefab;
```

```
    [SerializeField] private float timeToSpawn;


    private void Start()
    {
        StartCoroutine(Spawner());
    }

    private void Spawn()
    {
        GameObject enemy = Instantiate(enemyPrefab);
        enemy.transform.position = spawnPoints[Random.Range(0,
spawnPoints.Length)].position;
    }

    IEnumerator Spawner()
    {
        while (true)
        {
            yield return new WaitForSeconds(timeToSpawn);
            Spawn();
        }
    }
  }
}
```

Є. Інструкції для технічного персоналу та користувачів

Для користування даним програмним забезпеченням необхідно встановити універсальний інсталятор Unity і програмне забезпечення для керування проектами під назвою “Unity hub”.

Після відкриття Unity Hub потрібно ввійти за допомогою облікового запису Unity. Якщо у вас немає облікового запису, ви можете створити його тут (рис. 1).

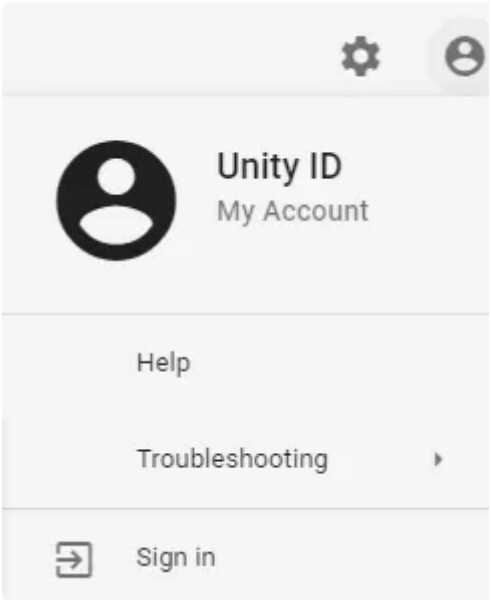


Рисунок 1. Обліковий запис Unity

У Unity Hub потрібно спочатку натиснути кнопку 'Installs', а потім 'ADD' (рис.2).

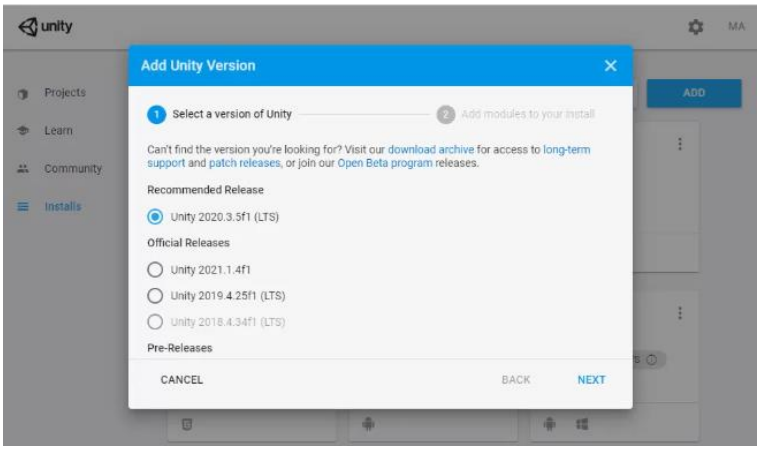


Рисунок 2. Вибір версії програми “unity hub”

Вам буде запропоновано вибір модулів. Модулі — це доповнення, необхідні для створення гри для певної платформи. Наприклад, якщо ви плануєте

створювати ігри для Android та iOS, виберіть 'Android Build Support' та/або 'iOS Build Support'. Ви можете вибрати скільки завгодно модулів (рис.3).

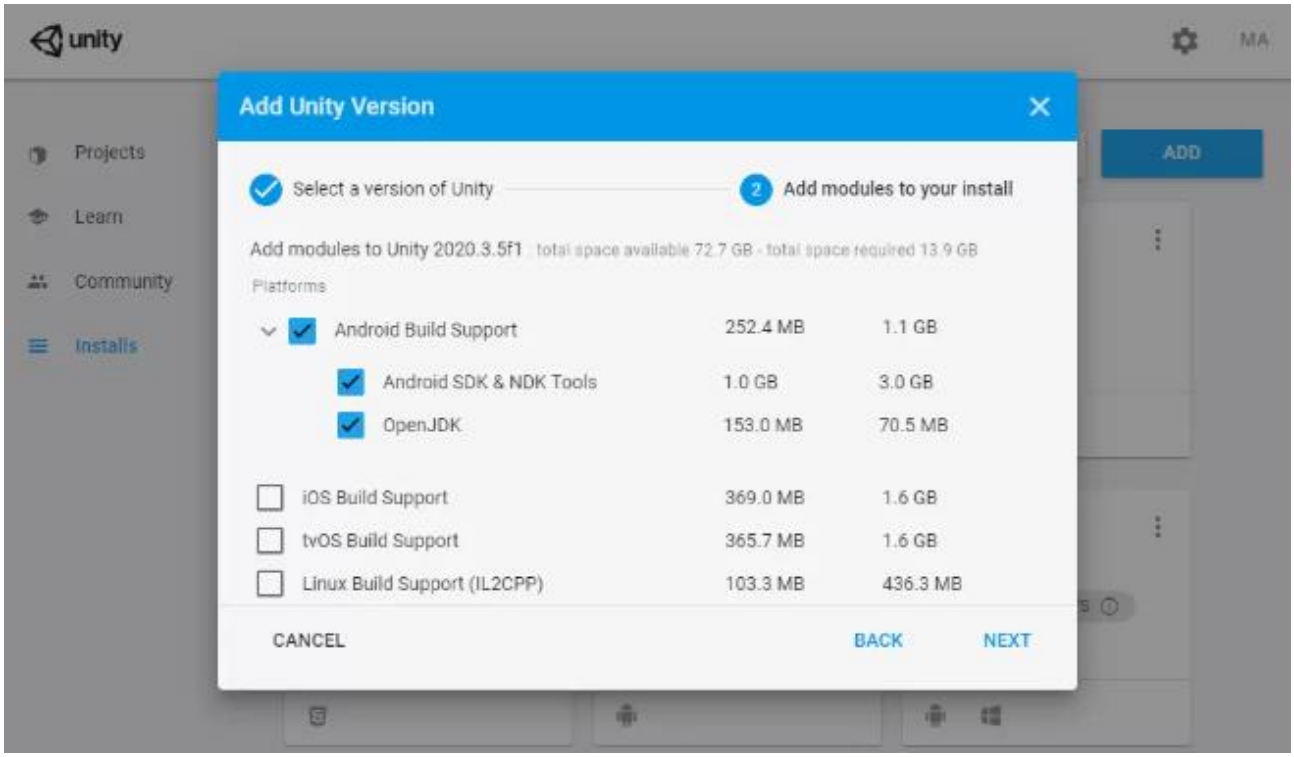


Рисунок 3. Додавання модулів в “Unity hub”

В Unity Hub натисніть 'Projects', а потім натисніть 'NEW'. З’явиться нове вікно, де ви зможете вказати назву проекту, вибрати шаблон проекту та вибрати місце, де буде створено папку проекту (рис.4).

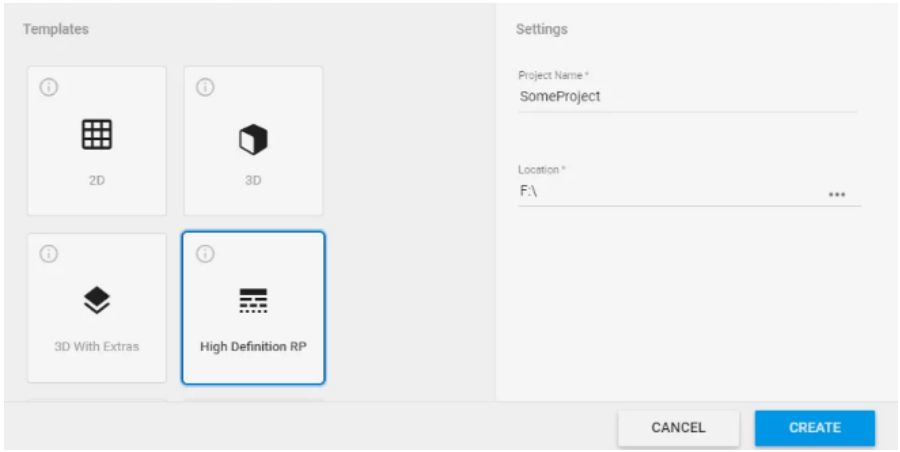


Рисунок 4. Меню “Unity hub”

Після створення проекту автоматично відкриється редактор і з’явиться можливість створювати власний проект. [2]