**Міністерство освіти і науки України**
**Тернопільський національний технічний університет імені Івана Пулюя**

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету )
Кафедра програмної інженерії
(повна назва кафедри)

# КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка комп'ютерної гри у жанрі «клікер» на базі ігрового рушія Unity

Виконав(ла): студент(ка) 4 курсу, групи СП-43
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Тимчак М. І.
(підпис) (прізвище та ініціали)

Керівник Коноваленко І. В.
(підпис) (прізвище та ініціали)

Нормоконтроль Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент Паламар А.М
(підпис) (прізвище та ініціали)

Тернопіль 2025

# АНОТАЦІЯ

Кваліфікаційна робота бакалавра на тему «Розробка комп’ютерної гри у жанрі «клікер» на базі ігрового рушія Unity» написана Тимчак Михайлом Ігоровичем, студентом Тернопільського національного технічного університету імені Івана Пулюя, факультет комп’ютерних та інформаційних систем та програмної інженерії, кафедра програмної інженерії, група СП-43.

Відомості про обсяг: сторінок – 53, рисунків – 23, таблиць – 0, частин – 4, додатків – 2, посилань – 12, формул – 0.

Ключові слова: Unity, клікер, C#, об’єктно-орієнтоване програмування, апгрейди.

Метою наукової роботи є створення комп’ютерної гри в жанрі «клікер» з використанням ігрового рушія Unity.

Для цього будуть використані методи проектування ігрової механіки, програмна реалізація логіки взаємодії об’єктів, а також засоби графічної візуалізації та інтерфейсу користувача.

Розроблена гра дозволить дослідити особливості побудови циклічного ігрового процесу та системи винагород, характерної для жанру «клікер». Результатом буде ігровий додаток з базовим ігровим процесом, системою накопичення ресурсів, можливістю апгрейдів та збереження прогресу.

У розробці використовується C# як основна мова програмування, середовище Unity Editor, підхід до проектування з використанням принципів об'єктно-орієнтованого програмування, а також сучасні практики гнучкої розробки.

# ABSTRACT

Bachelor's qualification work on the topic "Development of a computer game in the "clicker" genre based on the Unity game engine" was written by Tymchak Mykhailo Igorovych, a student of the Ivan Puluj Ternopil National Technical University, Faculty of Computer and Information Systems and Software Engineering, Department of Software Engineering, Group SP-43.

Information about the scope: pages – 53, figures – 23, tables -0 , sections – 4, appendices – 2, literature – 12, formulas – 0.

Keywords: Unity, clicker, C#, object-oriented programming, upgrades.

The purpose of the scientific work is to create a computer game in the "clicker" genre using the Unity game engine.

For this, methods of designing game mechanics, software implementation of the logic of object interaction, as well as graphic visualization and user interface tools will be used.

The developed game will allow us to explore the features of building a cyclic game process and a reward system typical of the "clicker" genre. The result will be a game application with a basic gameplay, a resource accumulation system, the ability to upgrade and save progress.

The development uses C# as the main programming language, the Unity Editor environment, an object-oriented approach to programming, and modern Agile development practices.

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення.

JSON (JavaScript Object Notation) – формат збереження структурованих даних.

ООП (Об’єктно-орієнтоване програмування) – парадигма програмування, в якій основою є класи та об’єкти, які між собою взаємодіють.

UML (Unified Modeling Language) – уніфікована мова графічного представлення та об’єктного моделювання в області розробки програмного забезпечення парадигми об’єктно-орієнтованого програмування.

Апгрейд – покращення характеристики.

Активний апгрейд – тимчасове покращення з ручною активацією.

Пасивний апгрейд – покращення, що діє постійно.

Cooldown – період часу очікування для наступної активації.

ЗМІСТ

# ВСТУП

Стрімкий розвиток індустрії розваг, зокрема ігрової індустрії, сприяє активному впровадженню новітніх інформаційних технологій у створення комп'ютерних ігор. Ігри є важливим засобом не лише дозвілля, а й інструментом психологічної розрядки, розвитку реакції, логіки та навіть навчання. Особливої популярності набули ігри у жанрі «клікер» (англ. clicker), які вирізняються простим геймплеєм, високою втягуваністю та зручністю реалізації на різних платформах.

Актуальність теми – вивчення процесу створення ігор у жанрі «клікер» із використанням ігрового рушія Unity, що є одним з найпопулярніших середовищ розробки ігор, завдяки своїй гнучкості, кросплатформеності та широкому набору функціоналу.

Мета роботи – розробка комп'ютерної гри у жанрі «клікер» на базі ігрового рушія Unity з реалізацією основних механік жанру: накопичення ресурсів, апгрейди та збереження прогресу.

Завдання полягає у створенні логіки гри, розробці графічного інтерфейсу, інтеграції системи збереження прогресу та тестуванні застосунку для забезпечення стабільної роботи.

Об'єктом дослідження є процес розробки комп'ютерних ігор на базі Unity.

Предметом дослідження є специфіка реалізації ігрових механік жанру «клікер» та побудова архітектури гри з урахуванням вимог зручності, ефективності та розширюваності.

# 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ ТА ТЕХНІЧНЕ ЗАВДАННЯ

## 1.1. Аналіз предметної області

Комп’ютерні ігри — це цифрові розваги, які працюють на електронних пристроях, зокрема ігрових консолях, смартфонах, планшетах, ПК або гарнітурах віртуальної реальності. Гравці можуть взаємодіяти з такими іграми через Інтернет, локальну мережу або у повністю автономному режимі. Як і традиційні фізичні ігри, цифрові ігри відзначаються великою різноманітністю: від складних багатокористувацьких світів до простих логічних задач для одного гравця. Вони різняться за жанром, тривалістю проходження, рівнем сюжетної глибини та ігровою механікою [1].

Розробка відеоігор — це процес створення комп’ютерних ігор, яким займається один розробник або команда, що може працювати як централізовано, так і дистанційно з різних куточків світу. Комерційні проєкти для ПК і консолей зазвичай підтримуються видавцями й можуть вимагати кількох років розробки. Натомість інді-ігри частіше створюються окремими ентузіастами або невеликими студіями, що зменшує часові й фінансові витрати. Активний розвиток індустрії незалежних розробників став можливим завдяки доступності рушіїв, таких як Unity та Unreal Engine, цифровим платформам розповсюдження (Steam, Uplay) та зростанню ринку мобільних ігор для Android і iOS [2].

У 1980-х роках, з появою домашніх комп’ютерів і перших консолей, одна людина могла самостійно створити гру від початку до кінця. Сьогодні ж створення ігор вимагає участі багатьох фахівців із різними компетенціями та підтримки спеціалізованої команди до складу яких зазвичай входять наступні представники:

1. Продюсер (цю роль також можна описати як позицію менеджера проекту, керівника проекту чи директора).

2. Видавець – організація, що займається публікацією / випуском комп’ютерних ігор.

3. Команда розробників, до складу якої входять наступні посади:

1) Дизайнер гри– особа, що створює ігровий процес, вигадуючи і розробляючи правила та структуру гри;

2) Художник – особа, яка відповідає за візуальний вигляд гри;

3) Ігровий програміст – інженер-програміст, основна діяльність якого полягає у розробці комп’ютерних ігор або програмного забезпечення до них (наприклад, інструментів для створення ігор, інструментів для модифікації ігор тощо);

4) Дизайнер рівнів – особа, яка розробляє рівні, завдання або місії для комп’ютерних відеоігор, використовуючи спеціалізоване програмне забезпечення;

5) Звукорежисери – спеціалісти, які відповідають за звукові ефекти, музичний та звуковий супровід у грі;

6) Тестувальник аналізує комп’ютерну гру і фіксує знайдені ним помилки, баги і дефекти, що є частиною процесу забезпечення якості [2].

Процес розробки гри зазвичай включає наступні етапи:

1. Базова концепція гри;

2. Концепція гри;

3. Документація дизайну гри;

4. Прототип;

5. Виробництво

Етапи можуть варіюватися залежно від переваг компанії та специфіки проекту.

## 1.2. Жанр клікер

Клікер (також відомий як incremental або idle game) — це жанр відеоігор, у яких взаємодія користувача мінімальна: переважно натисканням (клік або тап). Основна ідея полягає в простоті: кліками генерується ресурс, який потім витрачається на апгрейди, автоматизацію або генерацію пасивного доходу. Часто в таких іграх присутній механізм «престижу» — обнулення прогресу заради отримання бонусів, що стимулює гравця повертатися до гри з новою силою [3].

Однією з перших таких ігор вважали Progress Quest , випущену в 2002 році. Проте, це був сатиричний проект.

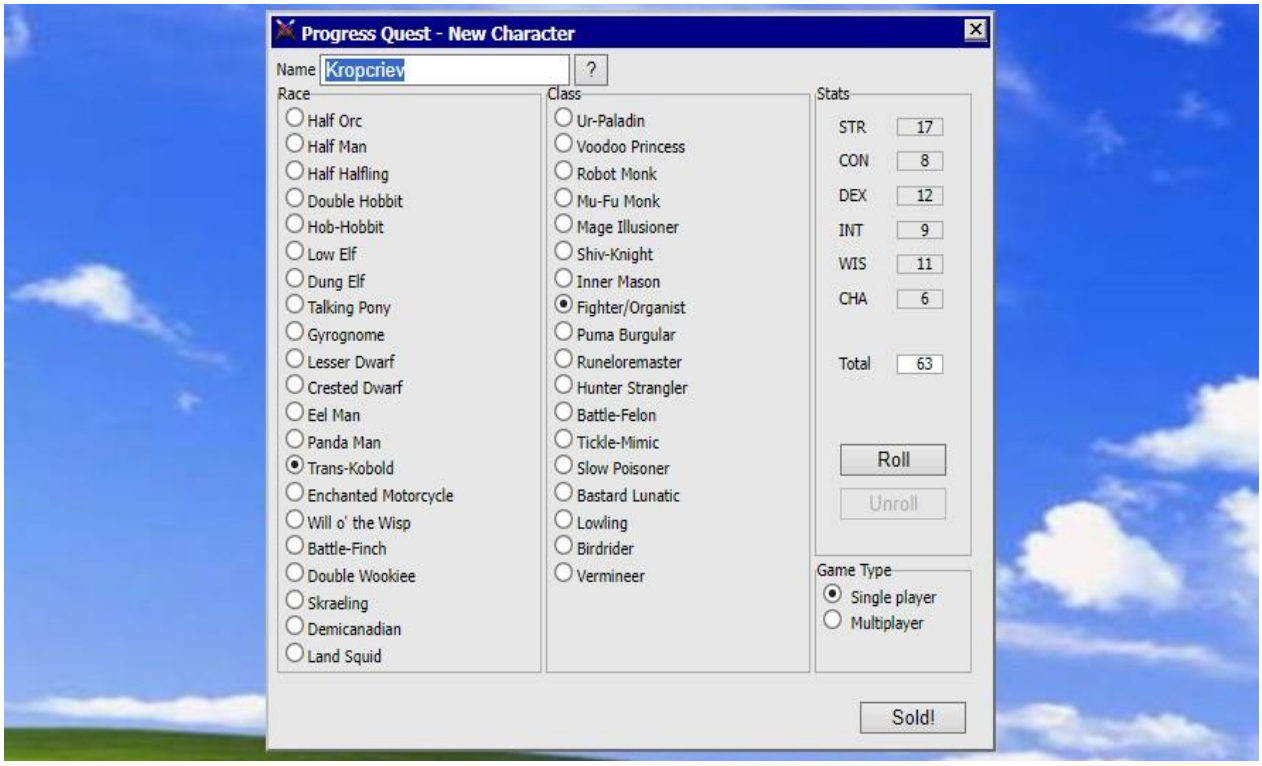


Рисунок 1.1 – Progress Quest

В цій грі гравці створювали власного персонажа, а процес гри відбувався автоматично, від збільшення рівня до битв з ворогами. Загалом, гравці радше дивилися історію свого персонажа, ніж грали ним.

Через свою простоту та мінімалізм, цей жанр був ігнорований роками. Проте

це змінилось в 2013 році з успіхом гри Cookie Clicker, розробленою Orteli.


Рисунок 1.2 – Cookie Clicker

Гра дозволяла гравцю натискати на велике печиво, відображене на екрані, та отримати 1 печиво в клік. За зароблене печиво, гравець купував будівлі та апгрейди для них, щоб заробляти більше печива.

Хоч ця гра виглядала простою, вона отримала багато прихильників та показала потенціал даного жанру [3].

## 1.3. Огляд ігрових рушіїв

Unity – це універсальний ігровий рушій, який дозволяє створювати ігри в 2D та 3D форматах для будь-якого жанру. Його переваги полягають у великій базі знань, обширній кількості готових рішень і шаблонів, активній спільноті та легкості освоєння.

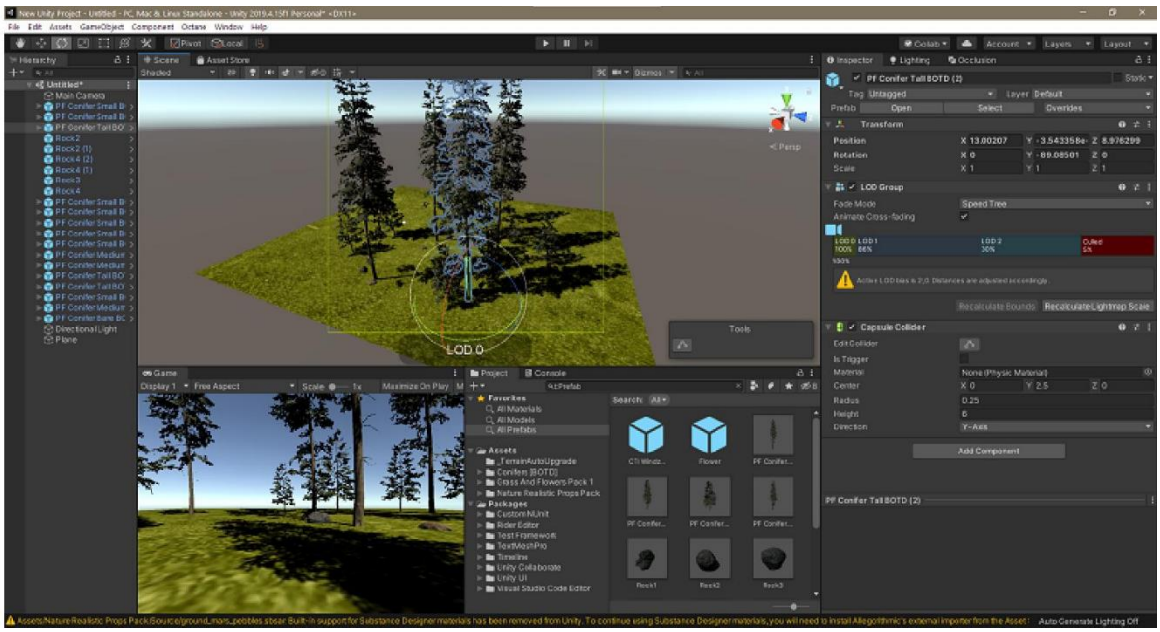

Рисунок 1.3 –Приклад роботи в Unity

Мови програмування: C#, NoCode (Bolt).

Цінова політика: Доступно безкоштовно, а також існують платні плани для великих проектів.

Переваги Unity:

- адаптивний та модульний двигун. Велика кількість компонентів доступна відразу;
- багатство шаблонів та прикладів;
- величезна база знань, велика спільнота;
- численні успішні проекти;

Недоліки Unity:

- необхідність глибокого занурення в деталі двигуна для оптимальної розробки;
- присутність недороблених версій та помилок;
- відсутність відкритого коду для малих команд розробників;
- значний розмір;
- фокус на розробці оновлень рушія для мобільних платформ;
- орієнтація підтримки на бізнес [4].

Далі йде UnrealEngine – ігровий двигун з довгою історією, який сьогодні

представляє передові можливості для створення масштабних ігор. Розробляється EpicGames.

Рисунок 1.4 – Приклад роботи в Unreal Engine

Мови програмування: C++, NoCode (Blueprints).

Цінова політика: Безкоштовно (за умовами).

Плюси UnrealEngine:

- потужний редактор для будь-яких задач;
- гнучка структура рушія;
- орієнтований на розробників, а не на бізнес, на відміну від Unity.
- готовність до великих проектів "з коробки";

Недоліки UnrealEngine:

- високий поріг входу;
- менш відкрита та численна спільнота;
- акцент на великі проекти;
- об'ємний рушій та його вимогливість [4].

Stride – це ігровий двигун з відкритим кодом, схожий на Unity. Почав розроблятися не так давно (раніше був відомий як Xenko), але вже пропонує

відкритість коду та безкоштовність.

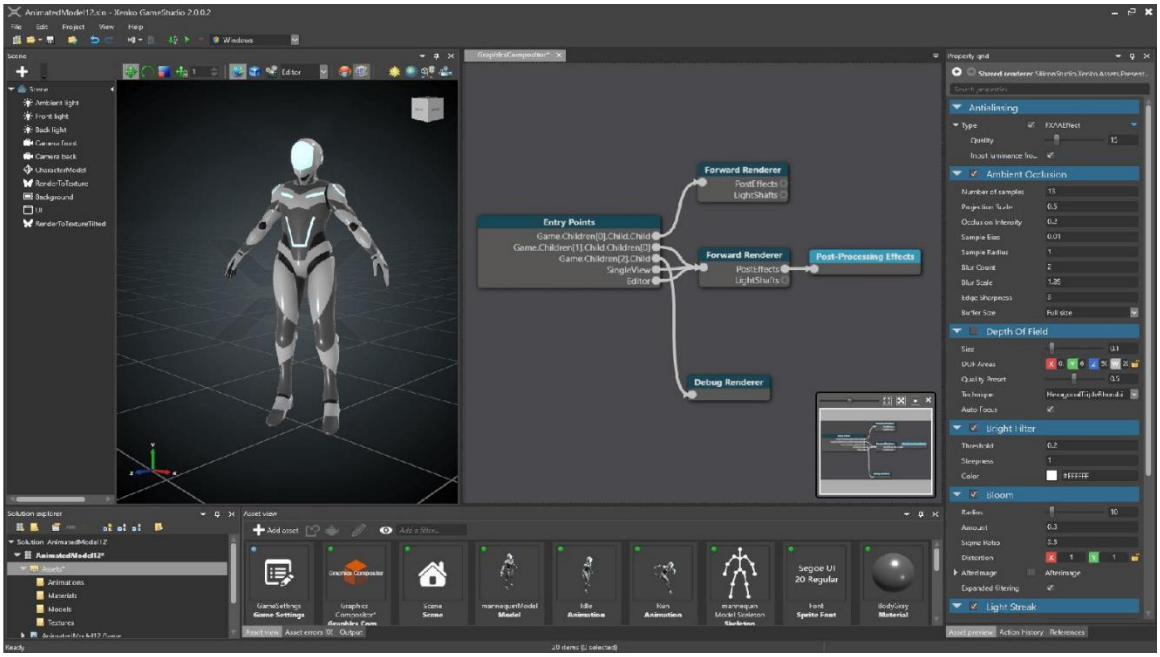


Рисунок 1.5 – Приклад роботи в Stride

Мова програмування: C#.

Цінова політика: Безкоштовно (OpenSource).

Переваги Stride:

- відкритий код;
- нижчий поріг входу порівняно з UnrealEngine;
- схожість з Unity за архітектурою та інструментами;
- доступ до новітніх технологій у порівнянні з Unity;
- підтримка VR на високому рівні.

Мінуси Stride:

- рушій та спільнота ще на ранньому етапі розвитку;
- обмежена кількість прикладів та навчальних матеріалів [4].

GameMakerStudio 2 – ще один інструмент для створення ігор, що включає безліч функцій, шаблонів і прикладів.


Рисунок 1.6 – Приклад роботи в GameMakerStudio

Мова програмування: не потрібна, але підтримується використання скриптів.

Цінова політика: Безкоштовно (з обмеженнями), повний доступ за місячну підписку.

Переваги GameMakerStudio:

- не вимагає знань у програмуванні;
- потужні інструменти;
- велика кількість шаблонів та прикладів;
- активна спільнота.

Мінуси GameMakerStudio:

- не найкраща оптимізація рушія;
- деякі аспекти потребують доопрацювання [4].

# 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА СТРУКТУРИ ПРОГРАМНОГО ПРОДУКТУ

## 2.1. Аналіз інструментальних засобів розробки

При створенні гри важливо вибрати ігровий движок та програмне середовище, які нададуть всі необхідні інструменти. Після дослідження популярних ігрових движків, я вирішив зупинити свій вибір на Unity, і ось чому:

- версія без оплати;
- великий асортимент інструментів для створення;

Вибір програмної мови та середовища є вирішальним для розробки проекту. Він впливає на продуктивність та швидкість розробки, а також на складність та витрати. Необхідно звернути увагу на добре продуману архітектуру системи при виборі. Обраний ігровий рушій підтримує мову програмування C#. Серед переваг вибору C# слід відзначити велику кількість онлайн ресурсів з прикладами та бібліотеками, а також її уніфікованість та доступність.

Для мови програмування C# існують такі середовища, як VS Code, JetBrains Rider та інші. У цьому випадку було обрано програмне середовище VS Code. Воно забезпечує велику гнучкість налаштувань, можливості для розширення та пропонує зручний інтерфейс для користувача. VSCode доступний для безкоштовного завантаження на офіційному сайті.

## 2.2 Пошук акторів та варіантів використання

Провівши аналіз предметної області, представлено актора гри – гравець.

Для зображення можливостей гравця при роботі з майбутньою грою була розроблена діаграма варіантів використання (рис. 2.1).

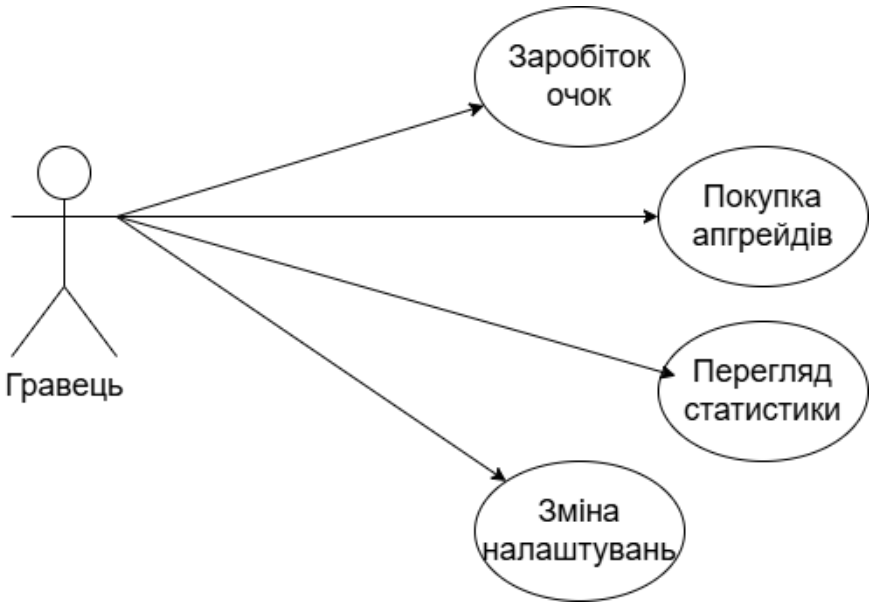


Рисунок 2.1 – Діаграма варіантів використання

Гравець має наступні можливості

- Заробіток очок;
- Покупка апгрейдів;
- Перегляд статистики;
- Зміна налаштувань.

## 2.3 Опис варіантів використання

Для кращого розуміння представленої раніше діаграми варіантів використання (рис. 2.1), буде ретельніше розглянуто ключові ВВ:

- Заробіток очок – гравець шляхом багаторазового натискання на активну область інтерфейсу генерує внутрішньоігрові очки, які використовуються як валюта.
- Покупка апгрейдів – гравець має можливість витрачати накопичені очки на апгрейди, які збільшують ефективність генерації очок або відкривають нові функції (автоматичні кліки, множники тощо).

- Перегляд статистики  гравець має можливість перегляду інформації про гру: кількість накопичених очок за весь час, кількість очок в клік та час ігрової сесії.
- Зміна налаштувань  гравець може змінювати налаштування звуку та зберігати прогрес гри.

## 2.4 Абстрактний рівень системи

У межах розробки даної гри було обрано об’єктно-орієнтовану (ООП) парадигму як різновид структурного програмування. Цей підхід вирізняється високим рівнем модульності, який забезпечується за рахунок ключових принципів ООП: інкапсуляції, успадкування, поліморфізму та абстракції.

Сьогодні ООП є однією з найпоширеніших парадигм у світі розробки. Вона дозволяє створювати гнучку й масштабовану архітектуру проєкту, що легко підтримується та розширюється. Нижче наведено основні риси, які стали вирішальними при виборі саме цієї парадигми:

- Інкапсуляція (Приватність) – Доступ до внутрішніх полів та методів класу регулюється через інтерфейси та модифікатори доступу. Це мінімізує ймовірність непередбачуваних змін у логіці роботи класів. Закриті (приватні) компоненти не можуть бути використані за межами класу, що захищає цілісність даних та поведінки об’єкта.
- Лаконічність та ефективність – Кожен об’єкт містить лише необхідний мінімум полів і методів. Такий підхід дозволяє уникнути надмірності, покращує читабельність коду та оптимізує використання ресурсів системи.
- Повторне використання коду (мультизадачність) – Завдяки механізмам успадкування та поліморфізму вже створені компоненти можна легко адаптувати до нових завдань із мінімальними змінами. Це зменшує час

розробки та підвищує ефективність використання пам’яті.

Для візуалізації архітектури системи буде використано UML-діаграми.

UML (Unified Modeling Language) – це уніфікована мова візуального моделювання, яка широко застосовується для опису структури та поведінки програмних систем [5]. Вона дозволяє наочно відобразити всі ключові компоненти системи, їх взаємозв’язки та функціональність, що особливо важливо у контексті об'єктно-орієнтованого проєктування. Серед можливостей UML – використання спеціалізованої нотації, семантики та позначень для створення схем, які полегшують аналіз, розробку та підтримку ПЗ. UML-діаграми стали невіддільною частиною розробки ООП-систем, оскільки вони дозволяють структуровано зобразити класи, їх атрибути, методи та зв’язки між ними.

Першою і базовою діаграмою є діаграма класів. Вона є фундаментом моделювання, оскільки демонструє абстрактну структуру системи: як класи взаємодіють між собою, які мають зв’язки та ієрархії.

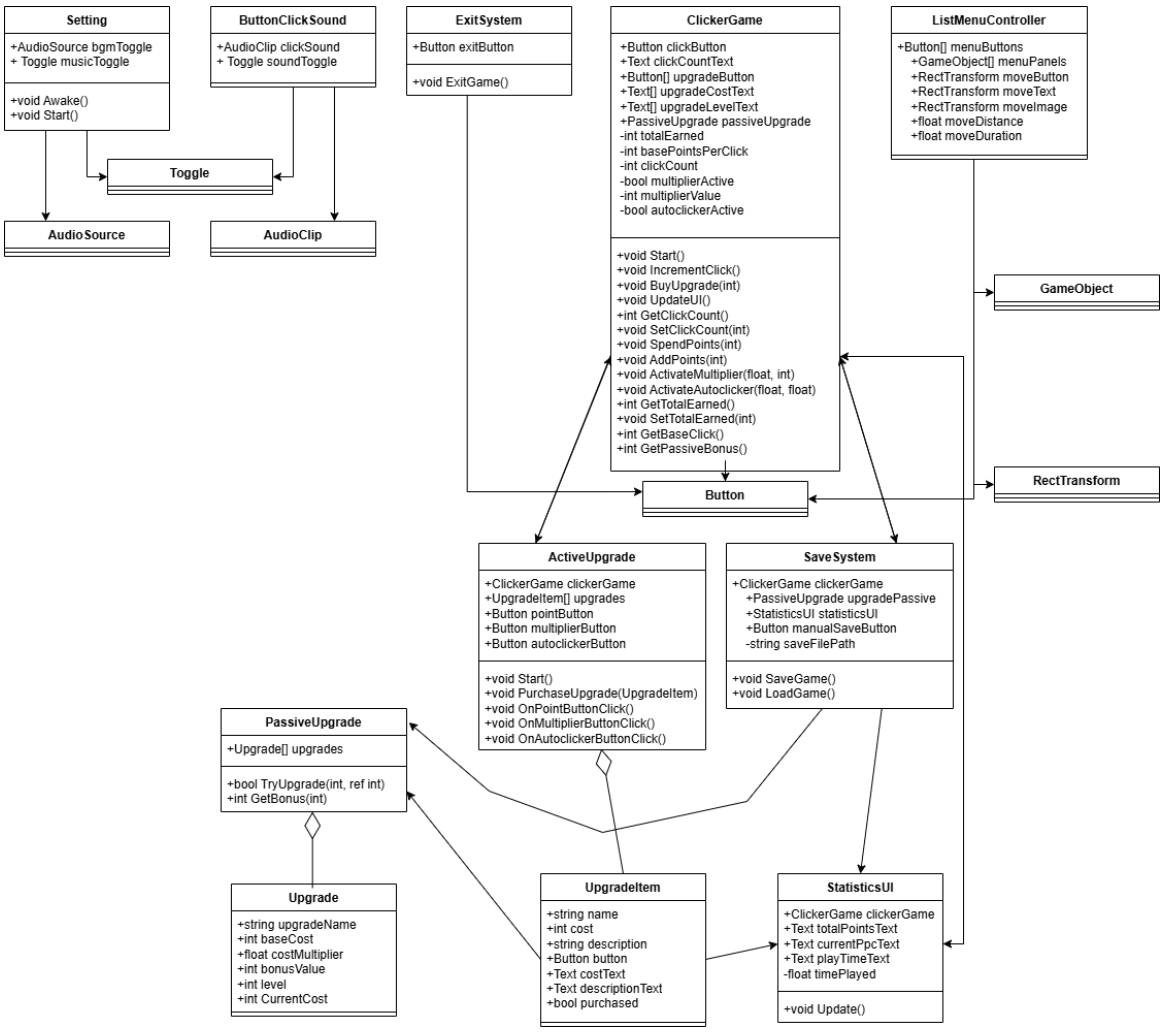


Рисунок 2.2 Діаграма класів

На вище вказаній діаграмі відображено всі ключові класи програми. ClickerGame є основним класом, який містить логіку гри:

- Керування апгрейдами;
- Облік кліків;
- Оновлення UI;

Класи PassiveUpgrade та ActiveUpgrade є реалізацією пасивних та активних апгрейдів відповідно. PassiveUpgrade містить логіку покупки та нарахування бонусу до основного кліку, ActiveUpgrade – логіку активації бонусу на час.

Класи Upgrade і UpgradeItem відповідають за структуру апгрейдів для PassiveUpgrade та ActiveUpgrade відповідно.

Клас StatisticsUI виводить статистику гри: загальну кількість очків, поточну кількість очок за клік, тривалість активної сесії гри.

Клас ListMenuController керує перемиканням між вкладками меню. Містить масиви кнопок і панелей, а також параметри анімації переміщення.

Клас SaveSystem керує збереженням і завантаженням прогресу гри. Дані беруться з класів ClickerGame, PassiveUpgrade, ActiveUpgrade, StatisticsUI.

Класи Setting, ButtonClickSound відповідають за налаштування звуку, ExitSystem – за логіку завершення застосунку.

# 3. РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

## 3.1 Вибір мови програмування та технологій розробки

При виборі інструментарію для створення гри критично важливо зосередитися на доступних бібліотеках, які забезпечують високу ефективність у роботі з даними. Спочатку потрібно вирішити, які інструменти оптимально підійдуть для клієнтської частини проекту. Клієнтська частина передбачає розробку додатку з підтримкою різних платформ.

У процесі проектування гри було застосовано об’єктно-орієнтовану методологію та мову програмування C#. Аналізуючи ключові класи та сутності, було використано специфічні рішення та поставлені завдання. Розробка велася методом реверс-інжинірингу, що дозволило створити основні компоненти та їх взаємодії, а далі - скласти UML-діаграму класів. Такий метод був обраний через складнощі оцінки повного обсягу та структури системи на початкових етапах розробки.

Вибір зручного середовища розробки також відіграє ключову роль у створенні додатку. З огляду на стрімкий розвиток інтегрованих середовищ розробки від різноманітних компаній у минулі роки, вибір тепер залежить від особистих уподобань розробника. Серед найбільш відомих редакторів коду сьогодні є JetBrains Rider, Visual Studio та Visual Studio Code. Завдяки досвіду роботи в Visual Studio та VSCode, а також вибору мови програмування C#, перевагу було надано Visual Studio Code. На відміну від інших інтегрованих середовищ розробки (IDE) та багатьох інших, VSCode підтримує велику кількість мов програмування. Це IDE є модульним, що означає, що воно найкраще працює з розширеннями, які спрощують процес програмування завдяки готовим сніппетам, можливості їх створення, покращеній підсвітці тексту та багатьом іншим корисним функціям [6].

Всі вищезгадані технології, мови, IDE та інші елементи були обрані з урахуванням вимог задачі та архітектури, а також особистих потреб. Це

забезпечить повне втілення запланованого додатку для зручності користувача.

## 3.2. Розробка основної ігрової логіки

В даній грі реалізовано методи різної складності котрі відповідають за логіку гри, бонусів, збереження прогресу та інші. Користувацький інтерфейс та основні параметри були задані в редакторі Unity. Повний лістинг коду знаходиться в додатку А.

Основою гри є класи ClickerGame та ListMenuController. Вони відповідають за наступні функції:

- Керування апгрейдами;
- Облік кліків;
- Оновлення UI;
- Контроль меню.

Перш за все було створено процес гри: натискання на кнопку збільшує кількість очок. На рисунках нижче відображено лістинг методів Start та IncrementClick, які відповідають за даний процес.

```
void Start()
{
    if (clickButton != null)
    {
        clickButton.onClick.AddListener(IncrementClick);
    }
```

Рисунок 3.1 Лістинг методу Start

```
void IncrementClick()
{
    int passiveBonus = 0;

    for (int i = 0; i < passiveUpgrade.upgrades.Length; i++)
    {
        passiveBonus += passiveUpgrade.GetBonus(i);
    }

    int total = basePointsPerClick + passiveBonus;
```

Рисунок 3.2 – Лістинг методу IncrementClick

Наступним етапом було створення методу, яка відповідає за оновлення користувацького інтерфейсу. Ця функція потрібна для коректного відображення змінних значень (кількість очок, рівні апгрейдів, їх ціна тощо). На рисунку нижче відображено лістинг методу UpdateUI, який відповідає за даний процес.

```
public void UpdateUI()
{
    if (clickCountText != null)
    {
        clickCountText.text = "Points: " + clickCount.ToString();
    }

    for (int i = 0; i < passiveUpgrade.upgrades.Length; i++)
    {
        if (upgradeCostText != null && i < upgradeCostText.Length)
        {
            upgradeCostText[i].text = "Cost: " + passiveUpgrade.upgrades[i].CurrentCost;
        }
        if (upgradeLevelText != null && i < upgradeLevelText.Length)
        {
            upgradeLevelText[i].text = "Level: " + passiveUpgrade.upgrades[i].level;
        }
    }
    for (int i = 0; i < passiveUpgrade.upgrades.Length; i++)
    {
        var upgrade = passiveUpgrade.upgrades[i];
        if (i < upgradeButton.Length)
        {
            upgradeButton[i].interactable = clickCount >= upgrade.CurrentCost;
        }
    }
}
```

Рисунок 3.3 – Лістинг методу UpdateUI

## 3.3. Розробка системи апгрейдів

Для даної гри було реалізовано два типи апгрейдів: пасивні (клас PassiveUpgrade) та активні (клас ActiveUpgrade).

Пасивні апгрейди збільшують кількість зароблених очок за клік. Вони складаються з наступних параметрів, записаних в класі Upgrade:

- Назва;
- Базова ціна;
- Множник ціни;
- Кількість бонусних очок;
- Рівень.

На рисунку нижче відображено лістинг класу PassiveUpgrade та методу TryUpgrade, який відповідає за процес покупки апгрейду.

```
public class PassiveUpgrade : MonoBehaviour
{
    5 references
    public Upgrade[] upgrades;

    0 references
    public bool TryUpgrade(int i, ref int points)
    {
        if (i < 0 || i >= upgrades.Length)
        {
            Debug.LogError("Invalid upgrade index.");
            return false;
        }

        Upgrade currentUpgrade = upgrades[i];
        int cost = currentUpgrade.CurrentCost;
        if (points >= cost)
        {
            points -= cost;
            currentUpgrade.level++;
            Debug.Log(currentUpgrade.upgradeName + " upgraded to level " + currentUpgrade.level + " (Cost: " + cost + ")");
            return true;
        }
        else
        {
            Debug.Log(currentUpgrade.upgradeName + " upgrade failed. Required: " + cost + ", Available: " + points);
            return false;
        }
    }
```

Рисунок 3.4 – Лістинг класу PassiveUpgrade

Активні апгрейди дають гравцю одноразові або тимчасові бонуси. В даній грі реалізовано наступні бонуси:

- + 2500 очок за активацію. Cooldown – 30 секунд;

- x3 множник для очок за клік. Тривалість – 15 секунд. Cooldown – 45 секунд;
- Авто-клікер. Тривалість – 10 секунд, інтервал між кліками – 20 мілісекунд. Cooldown – 90 секунд.

На рисунку нижче відображено лістинг методу PurchaseUpgrade класу ActiveUpgrade, який відповідає за процес покупки апгрейду.

```
void PurchaseUpgrade(UpgradeItem upgrade)
{
    if (upgrade.purchased || clickerGame.GetClickCount() < upgrade.cost)
        return;

    clickerGame.SpendPoints(upgrade.cost);
    upgrade.purchased = true;
    upgrade.button.interactable = false;

    Debug.Log("Purchased: " + upgrade.name);

    switch (upgrade.name)
    {
        case "Point Cache":
            pointButton.gameObject.SetActive(true);
            break;
        case "Rich Gains":
            multiplierButton.gameObject.SetActive(true);
            break;
        case "Auto-Click":
            autoclickerButton.gameObject.SetActive(true);
            break;
    }
}
```

Рисунок 3.5 – Лістинг методу PurchaseUpgrade

## 3.4. Система збереження прогресу

Клас SaveSystem реалізує систему збереження гри. Він автоматично зберігає дані гри (кількість кліків, загальний заробіток, рівні пасивних покращень та стан купівлі активних покращень) кожні 5 хвилин, дозволяє ручне збереження через кнопку, а також завантажує збережені дані при старті гри. Дані упаковуються в

клас GameData і серіалізуються в формат JSON, який зберігається в папці збережень.

На рисунках нижче відображено лістинг методів SaveGame та LoadGame, які відповідають за процес збереження даних та їх завантаження.

```
public void SaveGame()
{
    GameData data = new GameData();
    data.clickCount = clickerGame.GetClickCount();
    data.total = clickerGame.GetTotalEarned();

    int count = upgradePassive.upgrades.Length;
    data.upgradeLevels = new int[count];
    for (int i = 0; i < count; i++)
        data.upgradeLevels[i] = upgradePassive.upgrades[i].level;

    if (upgradeActive != null && upgradeActive.upgrades != null && upgradeActive.upgrades.Length > 0)
    {
        int activeCount = upgradeActive.upgrades.Length;
        data.activePurchased = new bool[activeCount];
        string purchasedStates = "";
        for (int i = 0; i < activeCount; i++)
        {
            data.activePurchased[i] = upgradeActive.upgrades[i].purchased;
            purchasedStates += $"[{upgradeActive.upgrades[i].name}: {data.activePurchased[i]}] ";
        }
        Debug.Log("Saving active upgrades: " + purchasedStates);
    }

    string json = JsonUtility.ToJson(data, prettyPrint: true);
    File.WriteAllText(saveFilePath, json);
    Debug.Log("Game saved to " + saveFilePath);
}
```

Рисунок 3.6 – Лістинг методу SaveGame

```
public void LoadGame()
{
    if (File.Exists(saveFilePath))
    {
        string json = File.ReadAllText(saveFilePath);
        GameData data = JsonUtility.FromJson<GameData>(json);

        clickerGame.SetClickCount(data.clickCount);

        for (int i = 0; i < data.upgradeLevels.Length && i < upgradePassive.upgrades.Length; i++)
            upgradePassive.upgrades[i].level = data.upgradeLevels[i];

        if (data.activePurchased != null && upgradeActive != null && upgradeActive.upgrades != null)
        {
            // Hide all first, then show purchased
            upgradeActive.HideAllActiveUpgradeButtons();
            for (int i = 0; i < data.activePurchased.Length && i < upgradeActive.upgrades.Length; i++)
            {
                upgradeActive.upgrades[i].purchased = data.activePurchased[i];
                if (upgradeActive.upgrades[i].button != null)
                    upgradeActive.upgrades[i].button.interactable = !data.activePurchased[i];
            }
            // Ensure buttons are shown for purchased upgrades
            upgradeActive.RefreshActiveUpgradeButtons();
        }

        clickerGame.SetTotalEarned(data.total);

        clickerGame.UpdateUI();

        Debug.Log("Game loaded from " + saveFilePath);
    }
}
```

Рисунок 3.7 – Лістинг методу LoadGame

### 3.5. Статистика та налаштування

Клас StatisticsUI відповідає за вивід статистики гри на екран: загальну кількість очків, поточну кількість очок за клік, тривалість поточної сесії гри. На рисунку нижче відображено лістинг даного класу.

```
public class StatisticsUI : MonoBehaviour
{
    4 references
    public ClickerGame clickerGame;

    1 reference
    public Text totalPointsText;
    1 reference
    public Text currentPpcText;
    1 reference
    public Text playTimeText;

    2 references
    private float timePlayed = 0f;

    0 references
    void Update()
    {
        if (clickerGame == null) return;

        timePlayed += Time.deltaTime;

        int totalPoints = clickerGame.GetTotalEarned();
        int baseClick = clickerGame.GetBaseClick();
        int bonusClick = clickerGame.GetPassiveBonus();
        int ppc = baseClick + bonusClick;

        totalPointsText.text = "Total Points Earned: " + totalPoints;
        currentPpcText.text = "Points per Click: " + ppc;
        playTimeText.text = "Current Session: " + FormatTime(timePlayed);
    }

    1 reference
    string FormatTime(float seconds)
    {
        int mins = Mathf.FloorToInt(seconds / 60);
        int secs = Mathf.FloorToInt(seconds % 60);
        return string.Format("{0:00}:{1:00}", mins, secs);
```

Рисунок 3.8 – Лістинг класу StatisticsUI

За налаштування звуку відповідають 2 класи: Setting (перемикач для музики) та ButtonClickSound (перемикач для звуків). На рисунках нижче відображено лістинг методів даних класів.

```
void Awake()
{
    if (bgmSource == null)
        bgmSource = GetComponent<AudioSource>();

    if (musicToggle == null)
        Debug.LogWarning("MusicManager: musicToggle is not assigned in the Inspector.");
}

0 references
void Start()
{

    if (bgmSource != null)
    {
        bgmSource.loop = true;
        bgmSource.Play();
    }

    if (musicToggle != null)
    {
        musicToggle.isOn = bgmSource != null && !bgmSource.mute;
        musicToggle.onValueChanged.RemoveAllListeners();
        musicToggle.onValueChanged.AddListener(HandleToggleValueChanged);
    }
}

1 reference
private void HandleToggleValueChanged(bool isOn)
{
    if (bgmSource == null) return;
    bgmSource.mute = !isOn;
}
```

Рисунок 3.9  Лістинг методів класу Setting

```
private void Start()
{
    Button button = GetComponent<Button>();
    if (button != null && clickSound != null)
    {
        button.onClick.AddListener(PlayClickSound);
    }
}

1 reference
void PlayClickSound()
{
    if (clickSound == null)
        return;

    if (soundToggle != null && !soundToggle.isOn)
        return;

    audioSource.PlayOneShot(clickSound);
}
```

Рисунок 3.10  Лістинг методів класу ButtonClickSound

## 3.6. Ілюстрація роботи створеного програмного забезпечення

В результаті проведеної роботи на виході отримано застосунок, з простим для розуміння користувацьким інтерфейсом.

Після запуску гри гравець зіткнеться з ігровим полем, яке складається з наступних частин:

- Верхня панель з кількістю очок;
- Ігрова кнопка у вигляді слайма;
- Нижня панель з кнопками меню.

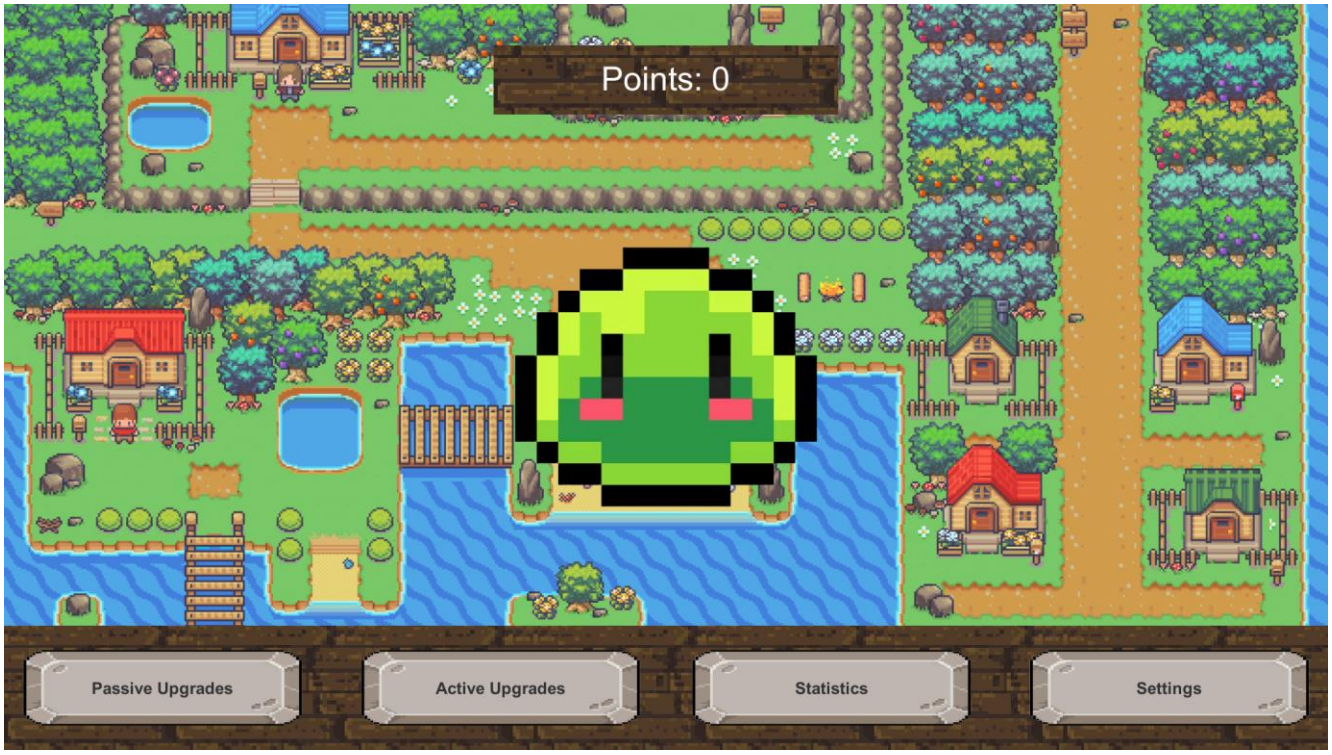


Рисунок 3.11 – Ігрове поле

При натисканні на слайма, програється короткий звук та додається очко у верхній панелі.

При натисканні будь-якої кнопки меню, слайм та верхня панель зсуваються вліво. З правої сторони з’являється панель меню, вибрана гравцем. На наступних рисунках відображено кожну панель меню.

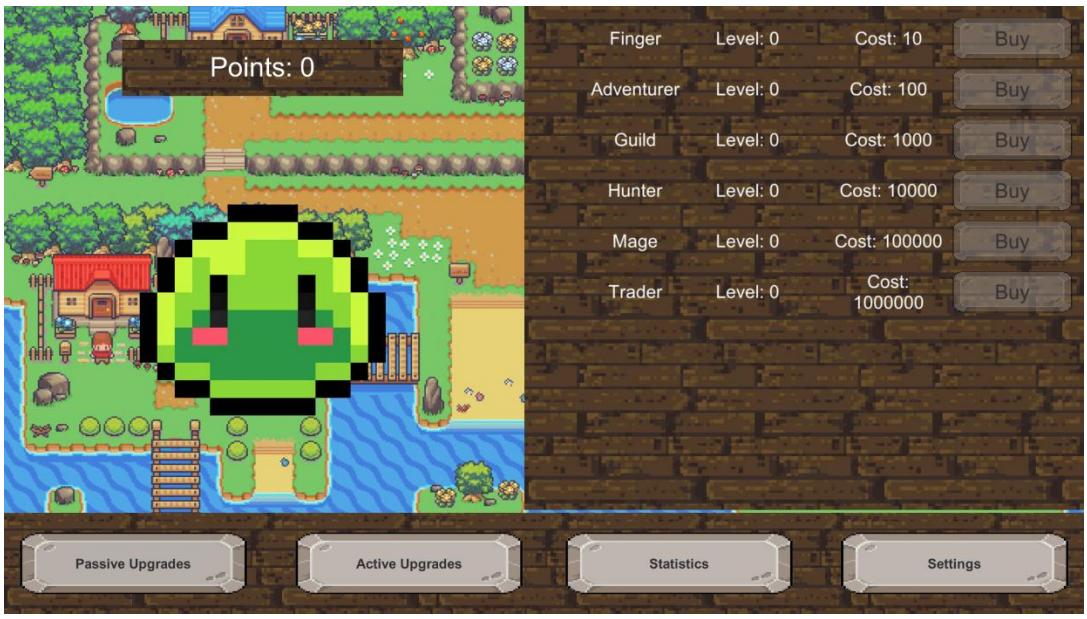


Рисунок 3.12 – Панель меню “Passive Upgrades”

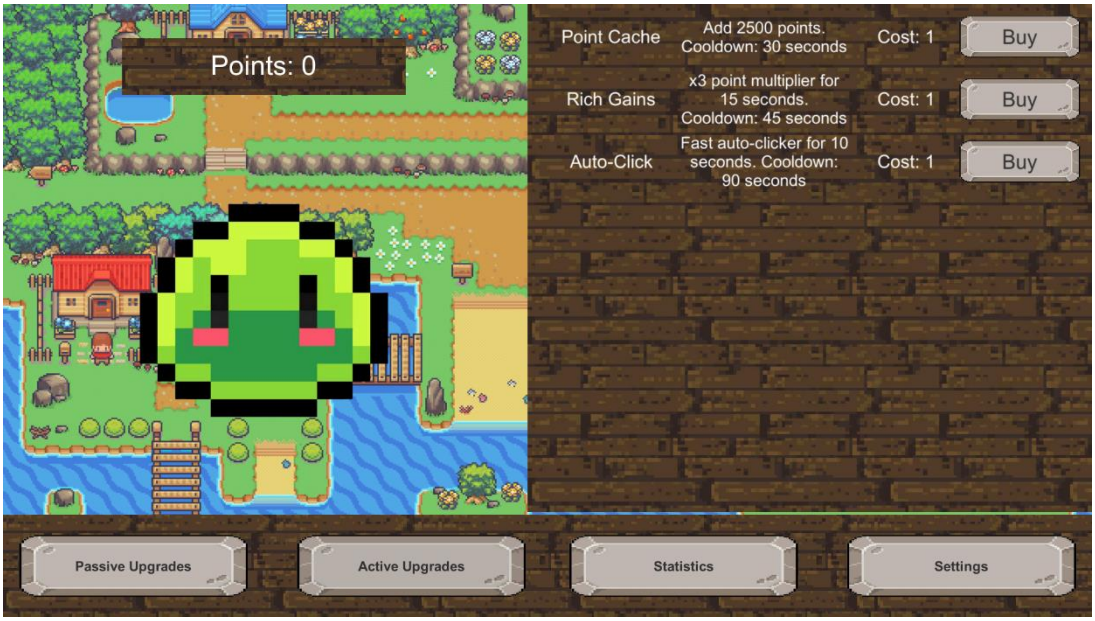


Рисунок 3.13 – Панель меню “Active Upgrades”

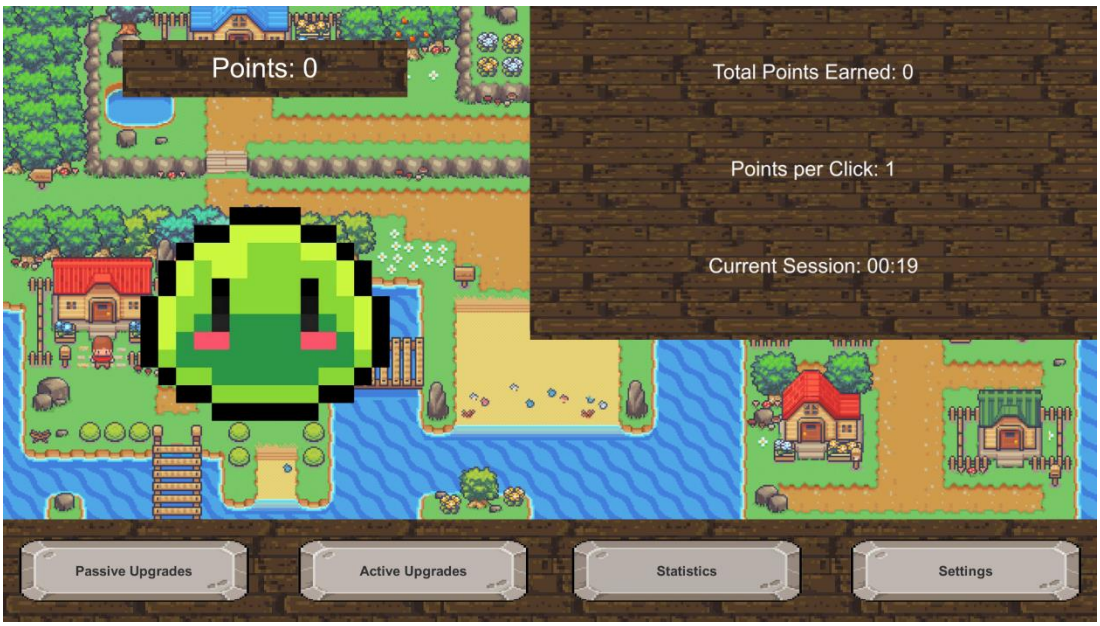


Рисунок 3.14 – Панель меню “Statistics”

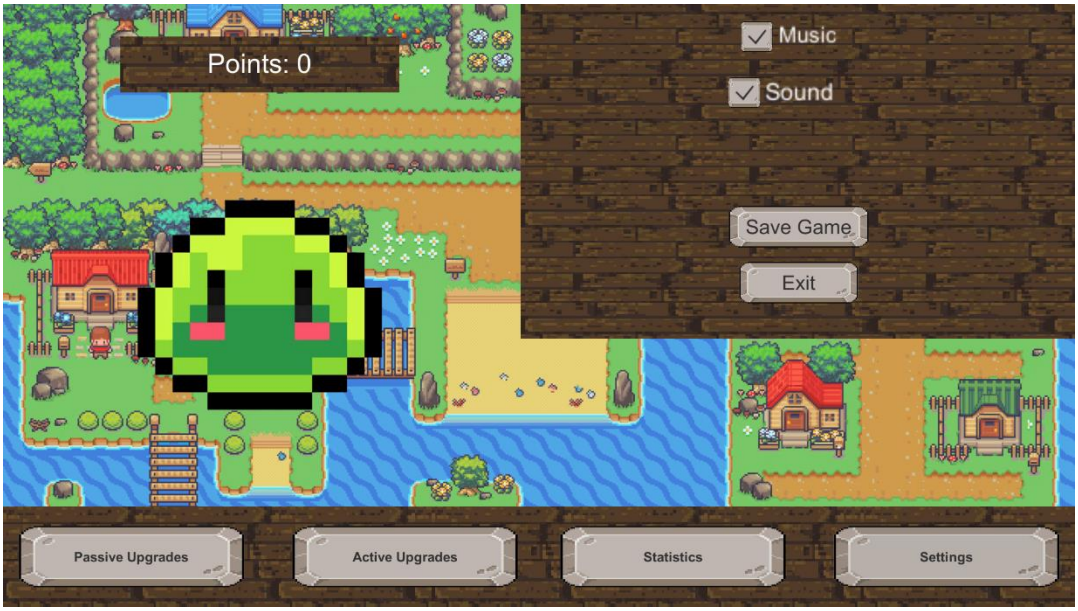


Рисунок 3.15 – Панель меню “Settings”

Для гри було створено 6 пасивних апгрейдів. Кожен з них має власну ціну рівня, множник цієї ціни та бонус очок до кожного кліку. Їх перелік наступний:

- Finger: ціна - 10 очок, множник - 1.1, бонус до кліку - 1;
- Adventurer: ціна - 100 очок, множник - 1.2, бонус до кліку - 5;
- Guild: ціна - 1000 очок, множник - 1.3, бонус до кліку - 25;
- Hunter: ціна - 10000 очок, множник - 1.4, бонус до кліку - 125;

- Mage: ціна - 100000 очок, множник - 1.5, бонус до кліку - 625;
- Trader: ціна - 1000000 очок, множник - 1.6, бонус до кліку - 3125.

## 3.7 Тестування програмного забезпечення та оцінка якості

Не існує програм без помилок. Дана гра потребує тестування, щоб переконатися, що реалізовані функції працюють коректно.

У межах тестування було також перевірено роботу активних і пасивних апгрейдів, збереження та завантаження ігрових даних, систему підрахунку очок, таймери бонусів, взаємодію елементів інтерфейсу з логікою гри, а також оновлення статистики в реальному часі.

З метою перевірки правильності роботи внутрішніх алгоритмів проводилось модульне тестування, під час якого окремо перевірялися ключові методи, що відповідають за клік, додавання очок, нарахування пасивного доходу, запуск автоклікера тощо. Це дозволило переконатися, що кожна функція працює ізольовано та передбачувано.

Окрім цього, використовувались інтерактивні (ручні) тести, під час яких у процесі гри перевірялася робота звуку, перемикачів, реагування інтерфейсу на зміну стану, та поведінка кнопок під час різних сценаріїв (наприклад, коли очок недостатньо для покупки).

Таким чином, тестування підтвердило працездатність основних функцій гри.

# 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

## 4.1 Долікарська допомога при шоку

Згідно з Порядком надання домедичної допомоги постраждалим при підозрі на шок в наказі Міністерства охорони здоров'я України від 09 березня 2022 року № 441, шок – це невідкладний стан, спричинений системним порушенням оксигенації тканин організму та, як наслідок, розвитком порушення функції життєво важливих органів та систем (дихальної, серцево-судинної, нервової). Ознаками шоку є: бліда, холодна і волога шкіра, загальна слабкість, неспокій, роздратованість, сухість в роті, відчуття спраги, часте або повільне дихання, порушення свідомості, непритомність [8].

Ненадання своєчасної допомоги при шоковому стані може призвести до серйозних ускладнень і навіть летального наслідку. У зв'язку з цим особливого значення набуває здатність осіб, які першими прибули на місце події, надати постраждалому належну долікарську допомогу. Знання алгоритму дій у таких ситуаціях дозволяє виграти дорогоцінний час до прибуття медичного персоналу та суттєво підвищити шанси на збереження життя потерпілого.

Діагностика шоку ґрунтується на визначенні показників, які характеризують загальний стан потерпілого. Найважливіший показник  рівень артеріального тиску. Що нижчим він є, то глибший розлад функцій організму, його життєдіяльності. Величина крововтрати – найбільш об'єктивний показник ступеня важкості шоку [9].

При наданні домедичної допомоги постраждалим при підозрі на шок спочатку потрібно переконатися у відсутності небезпеки. Якщо небезпеки немає, потрібно провести огляд потерпілого, здійснити виклик екстреної медичної допомоги, усунути причину виникнення шоку та надати постраждалому протишокове положення: покласти постраждалого на землю, покласти під ноги ящик, валик з одягу таким чином, щоб ступні ніг знаходились на рівні його підборіддя, підкласти під голову одяг чи подушку, вкрити постраждалого покривалом [10].

Під час шоку усувають дію травмуючих факторів і факторів розвитку шоку: зупиняють кровотечу, перев'язують рани, усувають загрозу асфіксії; вводять S-подібну трубку (повітропровід); при порушенні зовнішнього дихання очищують порожнини рота і носоглотки, усувають западання язика для відновлення прохідності дихальних шляхів; при пневмотораксі накладається пов'язка; проводиться інгаляція киснем, зупинка зовнішньої кровотечі; вводяться серцево-судинні і аналептичні засоби (виконує фельдшер); здійснюється іммобілізація кінцівок. Ввівши повторно знеболювальні засоби, дають гарячий чай та інші напої [9].

## 4.2 Розрахунок рівня шуму. Заходи щодо його зниження

Шум є одним із найпоширеніших фізичних чинників, що негативно впливають на умови праці та здоров’я людини. Тривале перебування в умовах підвищеного шумового навантаження може призвести до порушення слуху, підвищеної втомлюваності, зниження концентрації уваги, порушень нервової системи та розвитку професійних захворювань. Саме тому контроль та оцінка рівня шуму на робочому місці є важливим аспектом охорони праці.

Згідно з Санітарними нормами виробничого шуму, ультразвуку та інфразвуку ДСН 3.3.6.037-99, вимірювання шуму в октавних смугах або рівня шуму проводиться за допомогою шумоміра, який відповідає діючим вимогам Держстандарту України, перевірений в органах Держстандарту. і має посвідчення про перевірку [11].

Вимірювання на постійних робочих місцях – в зонах розташування органів керування технологічним устаткуванням виконують в такий спосіб: мікрофон шумоміра розташовують на висоті 1,5 м, на відстані 0,5 – 1м від обладнання (при дослідженні рівня шуму в кабінах мікрофон встановлюють у її центрі), виміри виконують за шкалою А шумоміра, у режимі «повільно». При дослідженні

постійного шуму фіксують рівні звукового тиску в октавних смугах на середньо геометричних частотах, а при непостійному – еквівалентні рівні звуку [9].

У разі перевищення допустимих значень роботодавець зобов'язаний вжити заходів щодо зниження шуму. Ці заходи поділяються на організаційні, медико-профілактичні, архітектурно-планувальні, технічні.

До організаційних та архітектурно-планувальних заходів відносяться угруповання приміщень з підвищеним рівнем шуму в одній зоні будинку, відділення їх коридорами, підсобними, допоміжними, складськими приміщеннями.

Медико-профілактичними заходами є проведення періодичних оглядів, прийом вітамінів для підвищення опірності, санітарно-курортне лікування.

Технічні заходи поділяються на 3 напрямки: усунення причин виникнення або зниження рівня шуму в джерелі, ослаблення шуму на шляху його поширення та індивідуальний захист працюючих.

Найбільш ефективним шляхом зниження шуму є заміна гучних технологічних операцій на малошумні. Проте реалізація таких методів захисту не завжди реальна та доцільна з економічної точки зору, тому застосовують зниження шуму в джерелі: застосування в механізмах матеріалів із звукопоглинаючими властивостями, своєчасне проведення профілактики й планово-попереджувальних ремонтів. Одним з найбільш простих рішень щодо зниження шуму на шляху його поширення є застосування звукоізолюючих кожухів – звуковідбиваючих або звукопоглинаючих.

Як індивідуальні засоби захисту від шуму застосовують спеціальні вкладиші у вушну раковину при величині шуму до 100 дБ, шумозахисні навушники при величині шуму від 100 дБ до 120 дБ а також шолом з навушниками при величині шуму більше 120 дБ [9].

# ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра було розроблено комп’ютерну гру у жанрі клікер з реалізацією кастомного ігрового процесу та підтримкою активних і пасивних покращень. Для побудови архітектури застосовано об’єктно-орієнтований підхід, що дозволив забезпечити масштабованість, модульність та розширюваність коду.

У грі реалізовано базову механіку кліків, а також систему модернізації: одноразові активні покращення з тимчасовим ефектом, пасивні покращення з прогресивним зростанням вартості. Крім того, було впроваджено систему збереження та завантаження прогресу, яка дозволяє зберігати дані вручну або автоматично з певним інтервалом.

Також реалізовано статистичний модуль, який відображає загальну кількість зароблених очок, тривалість сесії гри та поточну ефективність одного кліку. Важливою складовою стала система інтерфейсу: кнопки реагують на можливість покупки, звук кліку керується окремим перемикачем, що забезпечує персоналізацію досвіду користувача.

Гра є прикладом інтерактивного застосунку з розвиненою логікою та широкими можливостями для подальшого розвитку: впровадження досягнень, додавання нових систем тощо. Отримані результати демонструють можливість створення повноцінної гри у середовищі Unity із застосуванням сучасних підходів до програмування та проєктування.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ScienceDaily. Computer and video games. [Електронний ресурс] – Режим доступу: https://www.sciencedaily.com/terms/computer_and_video_games.htm/.

2. GameDesigning. The Video Game Development Essentials Guide. [Електронний ресурс] – Режим доступу: https://www.gamedesigning.org/video-game-development/.

3. Mr. Mine Game Blog. The Evolution and Origins of Idle Clicker and Incremental Games. [Електронний ресурс] – Режим доступу: https://mrmine.com/blog/the-evolution-and-origins-of-idle-clicker-and-incremental-games/.

4. YoungWonks. Top 10 Game Development Engines Today. [Електронний ресурс] – Режим доступу: https://www.youngwonks.com/blog/Top-10-Game-Development-Engines-Today/.

5. Документація UML [Електронний ресурс] – Режим доступу: http://www.uml.org/.

6. Редактор коду Visual Studio Code [Електронний ресурс] – Режим доступу: https://code.visualstudio.com/.

7. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329/.

8. НАКАЗ Про затвердження порядків надання домедичної допомоги особам при невідкладних станах. [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0356-22#n543.

9. ТЕМА 9. Медицина катастроф. Долікарська допомога при шоку. [Електронний ресурс] – Режим доступу до ресурсу: https://dl.tntu.edu.ua/content.php?cid=299870.

10. ТЕМА 14. Вібрація. Шум, ультра та інфразвук. Виробничий шум.

Параметри, вплив на працюючих. Нормування, заходи і засоби захисту працюючих. [Електронний ресурс] – Режим доступу до ресурсу: https://dl.tntu.edu.ua/content.php?cid=289160.

11. ПОСТАНОВА Санітарні норми виробничого шуму, ультразвуку та інфразвуку ДСН 3.3.6.037-99. [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va037282-99#Text.

12. Методичні вказівки до виконання дипломної роботи освітнього рівня - бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення/ Укладачі : Петрик М.Р.,Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

# ДОДАТКИ

# ДОДАТОК А – Лістинг коду інформаційної системи

## Лістинг коду 1 – ActiveUpgrade.cs

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

public class ActiveUpgrade : MonoBehaviour
{
    [System.Serializable]
    public class UpgradeItem
    {
        public string name;
        public int cost;
        public string description;
        public Button button;
        public Text costText;
        public Text descriptionText;
        public bool purchased;
    }

    public ClickerGame clickerGame;
    public UpgradeItem[] upgrades;

    // Buttons for active upgrades
    public Button pointButton;
    public Button multiplierButton;
    public Button autoclickerButton;

    private bool pointCooldown = false;
    private bool multiplierCooldown = false;
    private bool autoclickerCooldown = false;

    // Call this to hide all special upgrade buttons
    public void HideAllActiveUpgradeButtons()
    {
        if (pointButton != null) pointButton.gameObject.SetActive(false);
        if (multiplierButton != null)
multiplierButton.gameObject.SetActive(false);
        if (autoclickerButton != null)
autoclickerButton.gameObject.SetActive(false);
    }

    void Start()
    {
        foreach (var upgrade in upgrades)
        {
            if (upgrade.button != null)
            {
                upgrade.costText.text = "Cost: " + upgrade.cost;
                upgrade.descriptionText.text = upgrade.description;
                upgrade.button.onClick.AddListener(() =>
PurchaseUpgrade(upgrade));
            }
        }

        // Always hide all special buttons at start
        HideAllActiveUpgradeButtons();

        if (pointButton != null)
```

```
pointButton.onClick.AddListener(OnPointButtonClick);
        if (multiplierButton != null)
multiplierButton.onClick.AddListener(OnMultiplierButtonClick);
        if (autoclickerButton != null)
autoclickerButton.onClick.AddListener(OnAutoclickerButtonClick);
    }

    void PurchaseUpgrade(UpgradeItem upgrade)
    {
        if (upgrade.purchased || clickerGame.GetClickCount() < upgrade.cost)
            return;

        clickerGame.SpendPoints(upgrade.cost);
        upgrade.purchased = true;
        upgrade.button.interactable = false;

        Debug.Log("Purchased: " + upgrade.name);

        switch (upgrade.name)
        {
            case "Point Cache":
                pointButton.gameObject.SetActive(true);
                break;
            case "Rich Gains":
                multiplierButton.gameObject.SetActive(true);
                break;
            case "Auto-Click":
                autoclickerButton.gameObject.SetActive(true);
                break;
        }
    }

    void OnPointButtonClick()
    {
        if (pointCooldown) return;

        clickerGame.AddPoints(2500);
        StartCoroutine(PointCooldownRoutine());
    }

    IEnumerator PointCooldownRoutine()
    {
        pointCooldown = true;
        pointButton.interactable = false;
        yield return new WaitForSeconds(30f);
        pointButton.interactable = true;
        pointCooldown = false;
    }

    void OnMultiplierButtonClick()
    {
        if (multiplierCooldown) return;

        clickerGame.ActivateMultiplier(15f, 3); // 3x multiplier for 15s
        StartCoroutine(MultiplierCooldownRoutine());
    }

    IEnumerator MultiplierCooldownRoutine()
    {
        multiplierCooldown = true;
        multiplierButton.interactable = false;
        yield return new WaitForSeconds(45f);
        multiplierButton.interactable = true;
        multiplierCooldown = false;
```

```
    }

    void OnAutoclickerButtonClick()
    {
        if (autoclickerCooldown) return;

        clickerGame.ActivateAutoclicker(10f, 0.02f); // 10s duration, 20ms
click interval
        StartCoroutine(AutoclickerCooldownRoutine());
    }

    IEnumerator AutoclickerCooldownRoutine()
    {
        autoclickerCooldown = true;
        autoclickerButton.interactable = false;
        yield return new WaitForSeconds(90f);
        autoclickerButton.interactable = true;
        autoclickerCooldown = false;
    }

    // Call this after loading to update button visibility for purchased
upgrades
    public void RefreshActiveUpgradeButtons()
    {
        if (pointButton != null) pointButton.gameObject.SetActive(false);
        if (multiplierButton != null)
multiplierButton.gameObject.SetActive(false);
        if (autoclickerButton != null)
autoclickerButton.gameObject.SetActive(false);

        foreach (var upgrade in upgrades)
        {
            if (upgrade.purchased)
            {
                switch (upgrade.name)
                {
                    case "Point Cache":
                        if (pointButton != null)
pointButton.gameObject.SetActive(true);
                        break;
                    case "Rich Gains":
                        if (multiplierButton != null)
multiplierButton.gameObject.SetActive(true);
                        break;
                    case "Auto-Click":
                        if (autoclickerButton != null)
autoclickerButton.gameObject.SetActive(true);
                        break;
                }
            }
        }
    }
}
```

Лістинг коду 2 – ButtonClickSound.cs

```
using UnityEngine;
using UnityEngine.UI;

public class ButtonClickSound : MonoBehaviour
{
    public AudioClip clickSound;
    public AudioSource audioSource;
    public Toggle soundToggle;
```

```
    private void Start()
    {
        Button button = GetComponent<Button>();
        if (button != null && clickSound != null)
        {
            button.onClick.AddListener(PlayClickSound);
        }
    }

    void PlayClickSound()
    {
        if (clickSound == null)
            return;

        if (soundToggle != null && !soundToggle.isOn)
            return;

        audioSource.PlayOneShot(clickSound);
    }
}
```

Лістинг коду 3 – ClickerGame.cs

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

public class ClickerGame : MonoBehaviour
{
    // Elements for the clicker game.
    public Button clickButton;
    public Text clickCountText;

    //Arrays for upgrade buttons and their text.
    public Button[] upgradeButton;
    public Text[] upgradeCostText;
    public Text[] upgradeLevelText;

    public PassiveUpgrade passiveUpgrade;

    private int totalEarned = 0; // Total points
    private int basePointsPerClick = 1; //Starting point
    private int clickCount = 0; // Current points

    private bool multiplierActive = false;
    private int multiplierValue = 1;


    private bool autoclickerActive = false;

    void Start()
    {
        if (clickButton != null)
        {
            clickButton.onClick.AddListener(IncrementClick);
        }

        for (int i = 0; i < upgradeButton.Length; i++)
        {
            int index = i;
            if (upgradeButton[index] != null)
            {
                upgradeButton[index].onClick.AddListener(() =>
BuyUpgrade(index));
```

```
            }
        }

        UpdateUI();
    }

    void IncrementClick()
    {
        int passiveBonus = 0;

        for (int i = 0; i < passiveUpgrade.upgrades.Length; i++)
        {
            passiveBonus += passiveUpgrade.GetBonus(i);
        }

        int total = basePointsPerClick + passiveBonus;

        if (multiplierActive)
        {
            clickCount += total * multiplierValue;
            totalEarned += total * multiplierValue;
        }
        else
        {
            clickCount += total;
            totalEarned += total;
        }

        UpdateUI();
    }

    void BuyUpgrade(int index)
    {
        if (passiveUpgrade != null)
        {
            bool purchased = passiveUpgrade.TryUpgrade(index, ref
clickCount);
            if (purchased)
            {
                Debug.Log("Upgrade purchased for index " + index);
            }
        }
        UpdateUI();
    }

    public void UpdateUI()
    {
        if (clickCountText != null)
        {
            clickCountText.text = "Points: " + clickCount.ToString();
        }

        for (int i = 0; i < passiveUpgrade.upgrades.Length; i++)
        {
            if (upgradeCostText != null && i < upgradeCostText.Length)
            {
                upgradeCostText[i].text = "Cost: " +
passiveUpgrade.upgrades[i].CurrentCost;
            }
            if (upgradeLevelText != null && i < upgradeLevelText.Length)
            {
                upgradeLevelText[i].text = "Level: " +
passiveUpgrade.upgrades[i].level;
            }
```

```
        }
        for (int i = 0; i < passiveUpgrade.upgrades.Length; i++)
        {
            var upgrade = passiveUpgrade.upgrades[i];
            if (i < upgradeButton.Length)
            {
                upgradeButton[i].interactable = clickCount >=
upgrade.CurrentCost;
            }
        }
    }

    public int GetClickCount() => clickCount;
    public void SetClickCount(int value)
    {
        clickCount = value;
        UpdateUI();
    }

    public void SpendPoints(int amount)
    {
        clickCount -= amount;
        UpdateUI();
    }

    public void AddPoints(int amount)
    {
        clickCount += amount;
        totalEarned += amount;
        UpdateUI();
    }

    public void ActivateMultiplier(float duration, int multiplier)
    {
        if (!multiplierActive)
            StartCoroutine(MultiplierRoutine(duration, multiplier));
    }

    private IEnumerator MultiplierRoutine(float duration, int multiplier)
    {
        multiplierActive = true;
        multiplierValue = multiplier;

        yield return new WaitForSeconds(duration);

        multiplierActive = false;
        multiplierValue = 1;
    }

    public void ActivateAutoclicker(float duration, float interval)
    {
        if (!autoclickerActive)
            StartCoroutine(AutoclickerRoutine(duration, interval));
    }
    private IEnumerator AutoclickerRoutine(float duration, float interval)
    {
        autoclickerActive = true;

        float elapsed = 0f;
        while (elapsed < duration)
        {
            IncrementClick();
            yield return new WaitForSeconds(interval);
            elapsed += interval;
```

```
        }

        autoclickerActive = false;
    }

    public int GetTotalEarned() => totalEarned;

    public void SetTotalEarned(int amount)
    {
        totalEarned = amount;
        UpdateUI();
    }

    public int GetBaseClick() => basePointsPerClick;

    public int GetPassiveBonus()
    {
        int bonus = 0;
        for (int i = 0; i < passiveUpgrade.upgrades.Length; i++)
            bonus += passiveUpgrade.GetBonus(i);
        return bonus;
    }
}
```

Лістинг коду 4 – ExitSystem.cs

```
using UnityEngine;
using UnityEngine.UI;

public class ExitSystem : MonoBehaviour
{
    public Button exitButton;

    void Start()
    {
        if (exitButton != null)
        {
            exitButton.onClick.AddListener(ExitGame);
        }
    }

    void Update()
    {
        // Listen for Escape
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            ExitGame();
        }
    }

    // Exits the game or stops play in the Editor.
    public void ExitGame()
    {
        Debug.Log("Exiting game...");

        //Unity Editor stop.
#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
#else
        //Build stop.
        Application.Quit();
#endif
    }
}
```

Лістинг коду 5 – ListMenuController.cs

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

public class ListMenuController : MonoBehaviour
{
    // Arrays for menu buttons and their panels.
    public Button[] menuButtons;
    public GameObject[] menuPanels;

    // UI elements that will move when a menu is opened.
    public RectTransform moveButton;
    public RectTransform moveText;
    public RectTransform moveImage;

    // Movement parameters.
    public float moveDistance = 100f;
    public float moveDuration = 0.5f;

    // Track currently open menu (-1 = none).
    private int currentMenuIndex = -1;

    // Store the original positions of the moving UI elements.
    private Vector2 originalPosButton;
    private Vector2 originalPosText;
    private Vector2 originalPosImage;

    // Flag to track if the UI elements have been moved left.
    private bool elementsMovedLeft = false;

    void Start()
    {
        if (menuButtons.Length != menuPanels.Length)
        {
            Debug.LogError("Buttons not equal to panels.");
            return;
        }

        // Click events for each button.
        for (int i = 0; i < menuButtons.Length; i++)
        {
            int index = i;
            menuButtons[i].onClick.AddListener(() =>
OpenOrToggleMenu(index));
        }

        HideAllMenus();

        // Store the original positions for the UI elements.
        if (moveButton != null)
            originalPosButton = moveButton.anchoredPosition;

        if (moveText != null)
            originalPosText = moveText.anchoredPosition;

        if (moveImage != null)
            originalPosImage = moveImage.anchoredPosition;
    }

    // Method to open a menu or close it if it's already open.
    void OpenOrToggleMenu(int index)
    {
        if (currentMenuIndex == index)
```

```
        {
            HideAllMenus();
            currentMenuIndex = -1;

            // Move UI elements to original positions, if they had been
moved.
            if (elementsMovedLeft)
            {
                if (moveButton != null)
                    StartCoroutine(MoveToPosition(moveButton,
originalPosButton, moveDuration));
                if (moveText != null)
                    StartCoroutine(MoveToPosition(moveText, originalPosText,
moveDuration));
                if (moveImage != null)
                    StartCoroutine(MoveToPosition(moveImage,
originalPosImage, moveDuration));

                elementsMovedLeft = false;
            }
            Debug.Log("Closed menu at index: " + index);
        }
        else
        {
            HideAllMenus();
            if (index >= 0 && index < menuPanels.Length)
            {
                menuPanels[index].SetActive(true);
                Debug.Log("Opened menu at index: " + index);

                //Animate UI, if not moved.
                if (!elementsMovedLeft)
                {
                    if (moveButton != null)
                        StartCoroutine(MoveLeft(moveButton, moveDistance,
moveDuration));
                    if (moveText != null)
                        StartCoroutine(MoveLeft(moveText, moveDistance,
moveDuration));
                    if (moveImage != null)
                        StartCoroutine(MoveLeft(moveImage, moveDistance,
moveDuration));
                    elementsMovedLeft = true;
                }
            }
            currentMenuIndex = index;
        }
    }

    // Method to hide all menu panels.
    void HideAllMenus()
    {
        foreach (GameObject panel in menuPanels)
        {
            if (panel != null)
            {
                panel.SetActive(false);
            }
        }
    }

    // Coroutine to move UI elements to the left.
    IEnumerator MoveLeft(RectTransform uiElement, float distance, float
duration)
```

```
    {
        Vector2 startPos = uiElement.anchoredPosition;
        Vector2 targetPos = startPos + new Vector2(-distance, 0); // move
left
        float elapsed = 0f;
        while (elapsed < duration)
        {
            uiElement.anchoredPosition = Vector2.Lerp(startPos, targetPos,
elapsed / duration);
            elapsed += Time.deltaTime;
            yield return null;
        }
        uiElement.anchoredPosition = targetPos;
    }

    // Coroutine to move UI elements back to a target position.
    IEnumerator MoveToPosition(RectTransform uiElement, Vector2 targetPos,
float duration)
    {
        Vector2 startPos = uiElement.anchoredPosition;
        float elapsed = 0f;
        while (elapsed < duration)
        {
            uiElement.anchoredPosition = Vector2.Lerp(startPos, targetPos,
elapsed / duration);
            elapsed += Time.deltaTime;
            yield return null;
        }
        uiElement.anchoredPosition = targetPos;
    }
}
```

Лістинг коду 6 – PassiveUpgrade.cs

Лістинг коду 7 – SaveSystem.cs

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

public class SaveSystem : MonoBehaviour
{
    public ClickerGame clickerGame;
    public PassiveUpgrade upgradePassive;
    public ActiveUpgrade upgradeActive;

    public StatisticsUI statisticsUI;
    public Button manualSaveButton;
    private string saveFilePath;

    void Awake()
    {
        saveFilePath = Path.Combine(Application.persistentDataPath,
"savegame.json");
    }

    void Start()
    {
        // Manual save
        if (manualSaveButton != null)
            manualSaveButton.onClick.AddListener(SaveGame);

        // Start auto-save loop
        StartCoroutine(AutoSaveRoutine());
```

```
        // Load existing save data if present
        LoadGame();
    }

    private IEnumerator AutoSaveRoutine()
    {
        while (true)
        {
            yield return new WaitForSeconds(300f); // 5 minutes
            SaveGame();
        }
    }

    private class GameData
    {
        public int clickCount;
        public int[] upgradeLevels;
        public bool[] activePurchased;
        public int total;
    }

    /// Saves game data to a JSON file, overwriting previous save.

    public void SaveGame()
    {
        GameData data = new GameData();
        data.clickCount = clickerGame.GetClickCount();
        data.total = clickerGame.GetTotalEarned();

        int count = upgradePassive.upgrades.Length;
        data.upgradeLevels = new int[count];
        for (int i = 0; i < count; i++)
            data.upgradeLevels[i] = upgradePassive.upgrades[i].level;

        // Save active upgrades purchased state
        if (upgradeActive == null)
        {
            Debug.LogWarning("SaveSystem: upgradeActive is null!");
        }
        else if (upgradeActive.upgrades == null)
        {
            Debug.LogWarning("SaveSystem: upgradeActive.upgrades is null!");
        }
        else if (upgradeActive.upgrades.Length == 0)
        {
            Debug.LogWarning("SaveSystem: upgradeActive.upgrades is
empty!");
        }
        else
        {
            int activeCount = upgradeActive.upgrades.Length;
            data.activePurchased = new bool[activeCount];
            string purchasedStates = "";
            for (int i = 0; i < activeCount; i++)
            {
                data.activePurchased[i] =
upgradeActive.upgrades[i].purchased;
                purchasedStates += $"[{upgradeActive.upgrades[i].name}:
{data.activePurchased[i]}] ";
            }
            Debug.Log("Saving active upgrades: " + purchasedStates);
        }

        string json = JsonUtility.ToJson(data, prettyPrint: true);
```

```
        File.WriteAllText(saveFilePath, json);
        Debug.Log("Game saved to " + saveFilePath);
    }

    /// Loads game data from the save file, if it exists.
    public void LoadGame()
    {
        if (File.Exists(saveFilePath))
        {
            string json = File.ReadAllText(saveFilePath);
            GameData data = JsonUtility.FromJson<GameData>(json);

            clickerGame.SetClickCount(data.clickCount);

            for (int i = 0; i < data.upgradeLevels.Length && i <
upgradePassive.upgrades.Length; i++)
                upgradePassive.upgrades[i].level = data.upgradeLevels[i];

            if (data.activePurchased != null && upgradeActive != null &&
upgradeActive.upgrades != null)
            {
                // Hide all first, then show purchased
                upgradeActive.HideAllActiveUpgradeButtons();
                for (int i = 0; i < data.activePurchased.Length && i <
upgradeActive.upgrades.Length; i++)
                {
                    upgradeActive.upgrades[i].purchased =
data.activePurchased[i];
                    if (upgradeActive.upgrades[i].button != null)
                        upgradeActive.upgrades[i].button.interactable =
!data.activePurchased[i];
                }
                // Ensure buttons are shown for purchased upgrades
                upgradeActive.RefreshActiveUpgradeButtons();
            }

            clickerGame.SetTotalEarned(data.total);

            clickerGame.UpdateUI();

            Debug.Log("Game loaded from " + saveFilePath);
        }
    }
}
```

Лістинг коду 8 – Setting.cs

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

public class Setting : MonoBehaviour
{
    public AudioSource bgmSource;
    public Toggle musicToggle;

    void Awake()
    {
        if (bgmSource == null)
            bgmSource = GetComponent<AudioSource>();

        if (musicToggle == null)
            Debug.LogWarning("MusicManager: musicToggle is not assigned in
the Inspector.");
```

```
    }

    void Start()
    {

        if (bgmSource != null)
        {
            bgmSource.loop = true;
            bgmSource.Play();
        }

        if (musicToggle != null)
        {
            musicToggle.isOn = bgmSource != null && !bgmSource.mute;
            musicToggle.onValueChanged.RemoveAllListeners();

musicToggle.onValueChanged.AddListener(HandleToggleValueChanged);
        }
    }

    private void HandleToggleValueChanged(bool isOn)
    {
        if (bgmSource == null) return;
        bgmSource.mute = !isOn;
    }
}
```

Лістинг коду 9 – StatisticsUI.cs

```
using UnityEngine;
using UnityEngine.UI;

public class StatisticsUI : MonoBehaviour
{
    public ClickerGame clickerGame;

    public Text totalPointsText;
    public Text currentPpcText;
    public Text playTimeText;

    private float timePlayed = 0f;

    void Update()
    {
        if (clickerGame == null) return;

        timePlayed += Time.deltaTime;

        int totalPoints = clickerGame.GetTotalEarned();
        int baseClick = clickerGame.GetBaseClick();
        int bonusClick = clickerGame.GetPassiveBonus();
        int ppc = baseClick + bonusClick;

        totalPointsText.text = "Total Points Earned: " + totalPoints;
        currentPpcText.text = "Points per Click: " + ppc;
        playTimeText.text = "Current Session: " + FormatTime(timePlayed);
    }

    string FormatTime(float seconds)
    {
        int mins = Mathf.FloorToInt(seconds / 60);
        int secs = Mathf.FloorToInt(seconds % 60);
        return string.Format("{0:00}:{1:00}", mins, secs);
    }
}
```

ДОДАТОК Б – Диск із кваліфікаційною роботою бакалавра