**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

**Пояснювальна записка**

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: **«Розробка гри «Прибульці вчать слова» в жанрі «пригода» (adventure) з використанням платформи Unity 3D та мови програмування C#»**

Виконав: студент 5 курсу, групи ППЗ-52

спеціальності:

121 Інженерія програмного забезпечення

Борисюк Олександр Вячеславович

Керівник:

Шевченко Світлана Миколаївна

Рецензент:

Київ 2021

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти «Бакалавр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

**ЗАТВЕРДЖУЮ**
Завідувач кафедри
Інженерії програмного забезпечення
_____ Негоденко О.В.
"___" _____2021 року

# З А В Д А Н Н Я
## НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

БОРИСЮКУ ОЛЕКСАНДРУ ВЯЧЕСЛАВОВИЧУ
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка гри «Прибульці вчать слова» в жанрі «пригода» (adventure) з використанням платформи Unity 3D та мови програмування C#»

Керівник бакалаврської роботи Шевченко С.М., доцент кафедри ІПЗ, к. п. н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від 12.03.2021 року № 65

2. Строк подання студентом роботи 01.06.2021

3. Вхідні дані роботи:

У дослідженні застосовуються принципи розробки програмного забезпечення та методи дослідження актуальності програмного продукту на ринку цифрових товарів.
В якості засобу розробки виступає ігровий рушій Unity та мова програмування C#. До допоміжних засобів належать технічна документація до вищевказаних а також IDE Visual Studio та СКВ Git.
Для розробки моделі ПЗ застосовуються UML діаграми.

4. Зміст розрахунково-пояснювальної записки

4.1 Описати та обґрунтувати вибраний алгоритм щодо розв'язання задачі створення відеогри «Прибульці вчать слова» в жанрі «пригода» (adventure) з використанням платформи Unity 3D та мови програмування C#
4.2 Провести опис моделі алгоритмів ПЗ
4.3 Здійснити вибір програмних та архітектурних рішень
4.4 Виконати проектування програмного додатку «Прибульці вчать слова»
4.5 Здійснити опис розробленого програмного продукту
4.6 Провести тестування розробленого програмного продукту

5. Перелік графічного матеріалу

1. Титульний лист
2. Аналоги
3. Мета, об'єкт та предмет дослідження
4. Технічне завдання
5. Застосовані архітектурні рішення
6. Діаграма прецидентів
7. Діаграмма класів
8. Результат розробки
9. Висновок
10. Хвилина відео з додатку
11. Завершальний слайд

6. Дата видачі завдання: 19.04.2021

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів бакалаврської роботи | Строк виконання етапів роботи до | Примітка |
|---|---|---|---|
| 1. | Підбір науково-техічної літератури | 19.04.2021 | виконано |
| 2. | Вимоги до встановленого ПО | 29.04.2021 | виконано |
| 3. | Оцінка якості тестування до системи | 04.05.2021 | виконано |
| 4. | Концепція та архітектура веб-додатку | 10.05.2021 | виконано |
| 5. | Вступ, висновки, реферат | 21.05.2021 | виконано |
| 6. | Розробка презентації | 25.05.2021 | виконано |
| 7. | Здача роботи | 01.06.2021 | виконано |

Студент _____ Борисюк О.В
(підпис) (прізвище та ініціали)

Керівник бакалаврської роботи _____ Шевченко С.М.
(підпис) (прізвище та ініціали)

# РЕФЕРАТ

*Об’єкт дослідження* – Програмне забезпечення типу комп’ютерна гра. Процесс розробки прикладного програмного забезпечення мовою C# в поєднанні з ігровим рушієм Unity

*Предмет дослідження* – Комп’ютерна гра повчального характеру на рушію Unity.

*Мета роботи* – розробка комп’ютерної гри для зацікавлення та полегшення вивчення слів Української мови.

*Методи дослідження* – методи розробки, методи контролю версіями, методи навчання

В процессі виконання роботи було проведено аналіз процессу розробки комп’ютерної гри. Комп’ютерні ігри можуть бути зручним інструментом для навчання дітей дошкільного віку, оскільки дають можливість візуалізувати інформацію у привабливій для пізнання формі. А процесс взаємодії з віртуальним простором комп’ютерної гри сприяє кращому фокусуванню на предметі вивчення та зацікавлює до подальшого вивчення поданого матеріалу в приємній та невимушеній формі.

В результаті виконання роботи було розроблено програмне забезпечення типу «комп’ютерна гра» що дозволяє в процессі проходження вивчати нові слова, та отримувати практичні навички лічби та читання.

Проведено дослідження методів розробки програмного забезпечення мовою програмування C# та застосування середовища розробки Unity. Визначено доцільність використання ігрового рушія як основи при розоробці комп'ютерних ігор та програмних додатків віртуальної та доповненої реальності.

*Галузь використання* – Дошкільні навчальні заклади України або персональне домашнє застосування.

*Ключові слова* – C#, UNITY 3D, КОМП'ЮТЕРНА ГРА, ЖАНР ADVENTURE, GIT,

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

**Рушій гри** - це базове програмне забезпечення гри. Комплекс готових програмних рішень котрі можна розглядати як основу для створення нових ігор без виконання значних змін в процессі розробки. Таким чином можна значно зкоротити час розробки використовуючи деякі готові рішення з індивідуальним налаштуванням компонентів.

**Система керування версіями (СКВ)** це засіб, який записує зміни до файлу або групи файлів таким чином, щоб з часом можна було відновити певну версію за бажанням користувача. Таким чином можна створювати файли, змінювати документи, видаляти абож іншим чином експериментувати з файлами, а потім в любий момент відновити попередню структуру файлів та каталогів.

**Commit** в Git це закріплення/збереження змін репозиторію. При його створенні додається запис до історії змін.

# ВСТУП

В зв’язку з незупинним рухом технологічного прогрессу, на сьогоднішній день все частіше стаються непорозуміння між молодшим та старшим поколінням. Найчастіше це непорозуміння виникає при спостереженні за дітьми котрі значну частину свого вільного часу проводять переглядаючи сторінки в світовій мережі, абож граючи в ігри різних жанрів та змісту.

Зважаючи на заповненість повсякденного життя інформаційними технологіями та медіаджерелами, можна зробити логічний висновок про неможливість цілковитого відгородження об’єкта опіки від небажаного впливу вищевказаних. Але можна направити цікавість в корисне русло застосовуючи метод навчання під час гри. Така методика дозволяє дітям з користю проводити вільний час за власним бажанням, а не за настановою старших.

При нинішньому рівні поширення обчислювальної електроніки (смартфон, планшетний ПК, портативні ПК та стаціонарні ПК) нові представники людського виду з наймолодшого віку знайомляться з цим технологічним дивом XXI століття. Більшість експертів зходяться на думці, що 2-х річний вік є оптимальним для надання дитині доступу до цифрових приладів та ЗМІ. Але також зазначається, що така взаємодія має бути обмежена за часом та виключно повчальним змістом.

В зв’язку з поширенням необхідності в повчальних комп’ютерних іграх темою дипломної роботи стало створення повчальної гри, котра дала-б змогу весело та корисно проводити час дітям дошкільного віку. В процессі гри дитина в інтерактивному режимі зможе здобути навички лічби, розуміння та читання слів.

# 1 АНАЛІЗ ТА ХАРАКТЕРИСТИКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 1.1 Актуальність

Різні відеоігри в залежності від архітектури та застосованих методів розробки можуть розповсюджуватись та виконуватись лише на певному типі обладнання тобто платформі. В такому випадку можна поділити відеігри на такі що залежні від середовища виконання абож кросс-платформенні – ті котрі незалежать від цільової платформи користувача.

Серед популярних платформ на сьогоднішній день можна виділити персональний комп'ютер з операційною системою сімейства Windows, смартфон з ОС Android або iOS, а також гральна консоль PlayStation чи XBOX. На момент написання роботи, згідно з даними іноземного видання Newzoo, 52% ринку відеоігор складає портативна платформа, а саме відеоігри для смартфонів з показником в 45% від загального ринку.

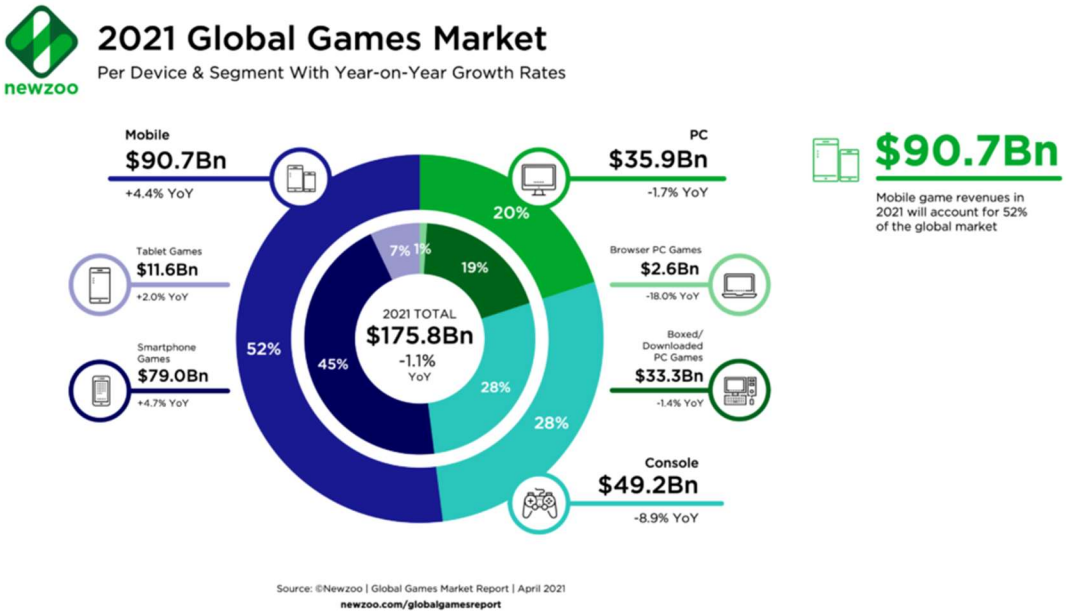



Рисунок 1.1.1 – Статистика дослідження ринку відеоігор за 2021 рік

Така популярність мобільної платформи легко пряснюється доступністю смартфонів. Звернувшись до статистики опублікованої Pew Research Center в 2018

році можна отримати результат опитування серед дорослих з різних країн. За результатами цього опитування 59% опитаних користуються смартфоном в своєму повсякденному житті. Додавши до цієї статистики неопитаних підлітків та дітей шкільного віку отримаємо результат такої домінації мобільної платформи на світовому ринку відеоігор.

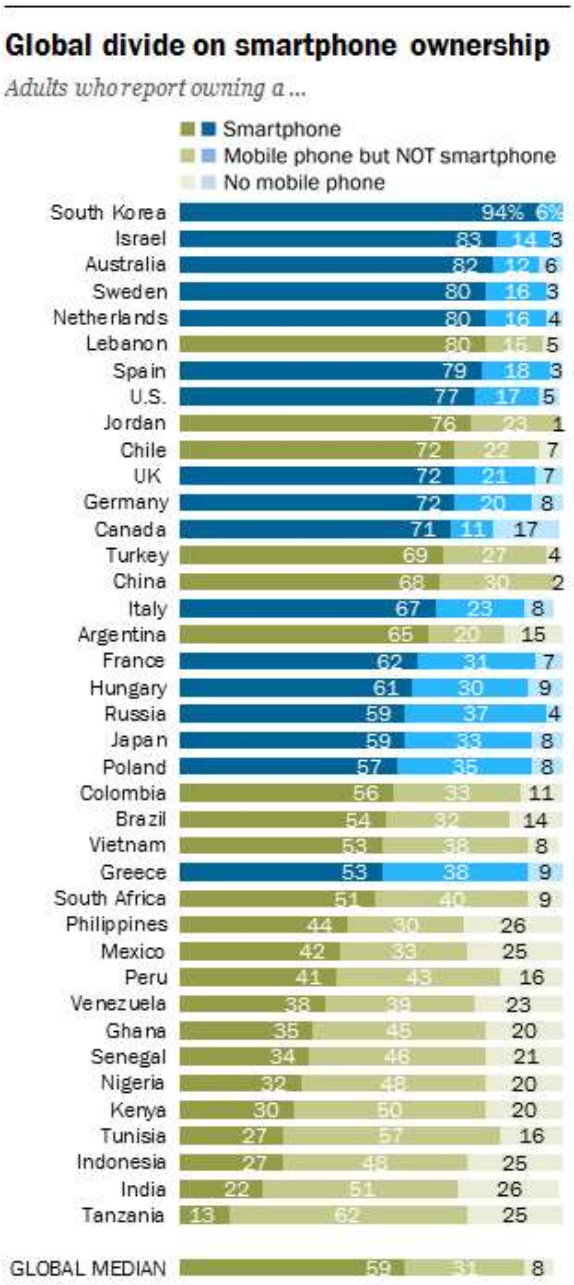



Рисунок 1.1.2 – Статистика поширення смартфонів серед дорослих

Пристрій котрий завжди знаходиться під рукою, неодноразово скрашував мої подорожі громадським транспортом, абож час очікування в черзі. Іноді такий тип

розваг стає єдиним з доступних в конкретний момент часу і залишається лише віддатись насолоді та отримувати задоволення від чергової безкінечної гри з ціллю побити свій попередній рекорд, або відкрити новий, недоступний раніше вигляд для певного ігрового елемента.

Розглянувши актуальні платформи для розробки відеоігор перейдемо до повчальних можливостей програмнії засобів цього типу. Перше що приходить на думку це якийсь симулятор, котрий дає змогу майбутньому пілоту здобути розуміння та навички керування літаком. Такий підхід до навчання дозволяє уникнути ризиків на початковому етапі становлення пілотом коли при недостачі практики помилка може коштувати дуже дорого. Але така річ як управління літаком може зацікавити лише свідому дорослу людину, абож дитину зацікавлену авіаційною тематикою.

Для навчання дітей чомусь новому їх в першу чергу потрібно зацікавити в самому процессі вивчення. Більшість інформації людина сприймає через зір, саме тому в повчальних іграх для дітей дошкільного віку найбільшу увагу потрібно приділити саме візуальній складовій відеогри. Саме завдяки кольоровим, та яскравим образам можна звернути увагу молодого дослідника на ту інформацію, котру потрібно донести до розуміння. Відеоігри дають можливість стіорювати ці самі цікаві образи а завдяки засобам програмування описувати складні сценарії взаємодії з користувачем для індивідуального підбору найпродуктивнішого методу навчання.

Таким чином зацікавленість в проходженні гри позитивно впливає на зфокусованість на предметі вивчення та бажанні отримувати нові позитивні враження в ході гри.

Саме для вирішення проблеми низької кількості якісних корисних розвоваючих ігор в списках популярних відеоігор було прийнято рішення створити відеогру для вивчення слів Української мови.

## 1.2 Порівняння аналогів

Вчимо і граємо. Українська мова - Словник та ігри – представляє з себе типову гру – набір карток з зображеннями предметів відповідно до попередньо обраної теми підписаними внизу.Є три режими гри:

1. Вгадуємо слово – по зображенню необхідно вгадати правильну назву предмета
2. Вгадуємо картинку – по слову необхідно підібрати відповідне зображення з запропонованих варіантів.
3. Орфографія - по зображенню необхідно написати правильну назву предмета

До додаткових функцій можна віднести статистику котра відображає відсоток правильних відповідей окремо по категоріях в кожному з режимів. Вивчення слів відбувається шляхом почергового перелистування карток в одній з попередньо обраних категорій. Кожне нове зображення супроводжується голосом диктора, котрий називає той чи інший предмет.

Хто у горах? – гра для дітей з приємною візуальною складовою. Відсутня перевірка вивчених слів. Приємний звуковий супровід. Голос диктора за допомогою інтонації звартає увагу на предмет вивчення. Вивчення слів відбувається шляхом перегляду 2D анімації з фокусуванням на певних складових об’єктах композиції

Весела Розмальовка – розмальовка. Коли картина повністю заваршена гра задає питання «Що зображено на малюнку?» і 4 варіанти відповіді. Невірна відповідь приводить кнопку до здійснення коливальних рухів вліво-вправо. При виборі вірного варіанту, питання та варіанти відповіді просто зникають.

Всього в грі є 8 категорій (Хеллоуїн, казки, транспорт, тварини, муз. інструменти, різдво, пасха, числа), на початку доступні лише 2. В кожній було по 10 зображень

Провівши дослідження наявних продуктів подібного характеру мною не було виявлено кандидатів спроможних змагатись з тривимірною відеогрою. Все що було

знайдено використовує в своїй концепції або плоскі зображення або 2D анімацію. І не дає свободи гравцю для переміщення в 3D просторі

# 2 АНАЛІЗ ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

## 2.1 Ігровий рушій

На етапі проектування комп’ютерної гри розробник має обрати один з двох доступних шляхів розвитку проекту. Повне програмування з нуля, абож застосування одного з доступних рушіїв ігор, котрі можна знайти у вільному доступі в мережі інтернет. Кожен з цих варіантів супроводжують складнощі, та ризики що можуть виникнути безпосередньо в процессі розробки. Для зменшення можливих ризиків необхідно правильно оцінити вимоги до кінцевого продукту та проаналізувати алгоритми що будуть застосовуватись в ході написання ігрової логіки.

Рушій гри - це базове програмне забезпечення гри. Комплекс готових програмних рішень котрі можна розглядати як основу для створення нових ігор без виконання значних змін в процессі розробки. Таким чином можна значно зкоротити час розробки використовуючи деякі готові рішення з індивідуальним налаштуванням компонентів.

В більшості доступних на сьогоднішній день ігрових рушіїв, базові можливості зможуть задовольнити потреби переважної кількості початківців, професіоналів або-ж малих ігрових студій, котрі в силу свого низького бюджету немають фінансової спроможності виділити значну частину часу розробки на створення власних вирішень основних тривіальних задач.

Згідно з вищесказаним, на основі аналізу доступних ігрових рушіїв мій вибір пав на ігровий рушій«Unity».


Рисунок 2.1.1 – Логотип Unity

Для написання ігрової логіки з застосуванням даного рушія використовується мова програмування C#, а для обрахунку фізичних явищ застосовується фізичний рушій PhysX від NVIDIA.

В липні 2005 року була анонсована та випущена перша версія програмного забезпечення ексклюзивно для MAC OS. Метою розробників було створення широкого набору інструментів для полегшення доступності та простоти програмування комп’ютерних ігор.

Файли сирцевого коду зберігаються в теці проєкту. В таких файлах мовою програмування C# описуються правила поведінки окремих ігрових об’єктів, а сам редактор Unity дозволяє прикріплювати файли до цих ігрових об’єктів. Таким чином можливе багаторазове застосування одних і тих самих користувацьких скриптів до багатьох ігрових об’єктів при необхідності однакової поведінки у кількох компонентів системи.

Для персонального користування ігровий рушій поширюється безкоштовно при умові, що щорічний прибуток з гри не перевищує 100 тис. долларів.

Згідно з попередньо отриманими даними, можна зробити висновок про можливість застосування ігрового рушія Unity для виконання поставленої задачі – «Написання комп'ютерної гри «Прибульці вчать слова»». Застосовуючи дану технологічну базу можна значно зкоротити час розробки комп'ютерної гри пропустивши етап написання коду для рендерингу 3D сцен та моделей, а також опис правил фізичної взаємодії між елементами системи.

## 2.2 Система контролю версіями

Система керування версіями (СКВ) це засіб, який записує зміни до файлу або групи файлів таким чином, щоб з часом можна було відновити певну версію за бажанням користувача. Таким чином можна створювати файли, змінювати документи, видаляти абож іншим чином експериментувати з файлами, а потім в любий момент відновити попередню структуру файлів та каталогів.

Зазвичай такі системи використовуються розробниками для відстеження, документування та контролю над поступовими змінами в файлах проєкту або готової інформаційної системи. Такий підхід дозволяє багатьом розробникам одночасно вносити зміни до різних файлів проєкту та загальними зусиллями вирішувати проблеми, що виникають в ході розробки.

До основних типів таких систем відносяться: Локальні, централізовані та розподілені СКВ

Локальні СКВ являються простим програмним засобом, котрий записує зміни в файлах до бази данних.

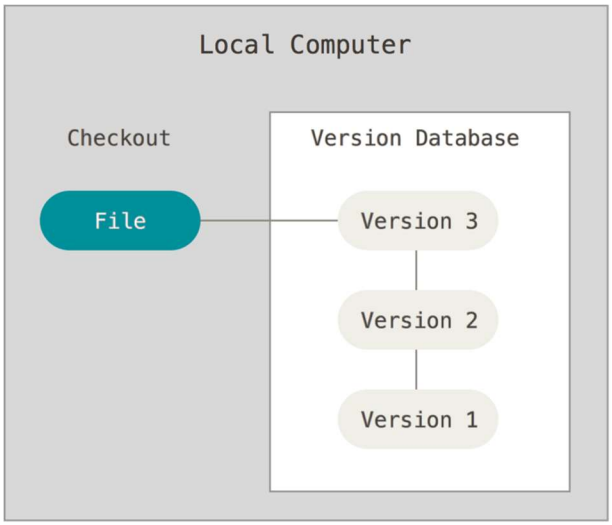



Рисунок 2.2.1 – Структурна схема локальної системи керування версіями

Таким чином здійснюється облік про всі зміни проведені над файлом, та дає змогу відновити попередні версії в разі потреби. Недоліком таких систем являється неможливість вести паралельну розробку багатьом користувачам, оскільки система

працює лише на одному локальному ПК користувача. Для вирішення цієї проблеми були розроблені централізовані СКВ.

Централізовані СКВ зберігають всі версії файлів на сервері та надсилають збережені копії файлів за запитом користувача.

Таким чином стає можлива багатокористувацька розробка, а також адміністрування користувачів СКВ. Надання прав на редагування файлів, підтвердження змін, та відновлення попередніх версій. На протязі довгого часу застосування цього типу СКВ являлось стандартом але й тут є недоліки.

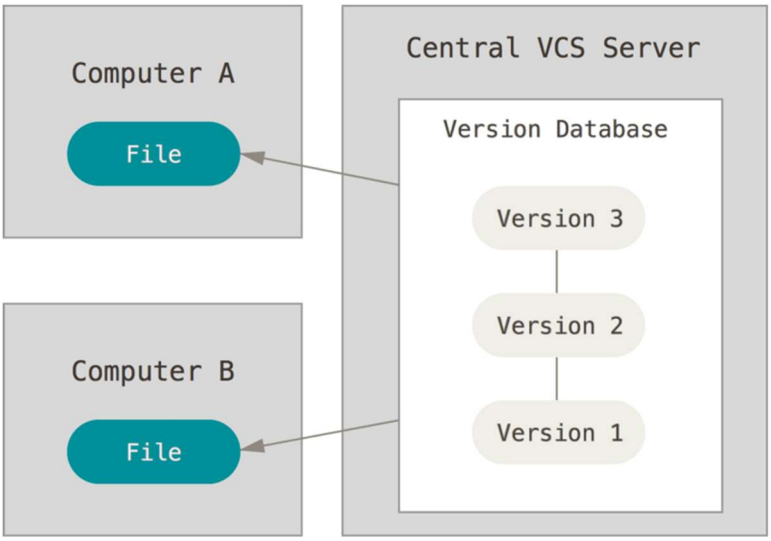



Рисунок 2.2.2 – Структурна схема централізованої системи керування версіями

Головним недоліком як не дивно є той самий сервер котрий зберігає всю інформацію. При виникненні неполадок на сервері постає проблема повної неможливості подальшої розорбки в звязку з неможливістю обміну даними між користувачами та внесенням змін. А при пошкодженні носія інформації можна взагалі втратити всю історію змін проєкту (за вийнятком поодиночних файлів на локальних ПК користувачів ЦСКВ).

Розподілені СКВ вирішують недоліки централізованих СКВ створюючи повну копію репозиторіїв на персональних ПК користувачів.

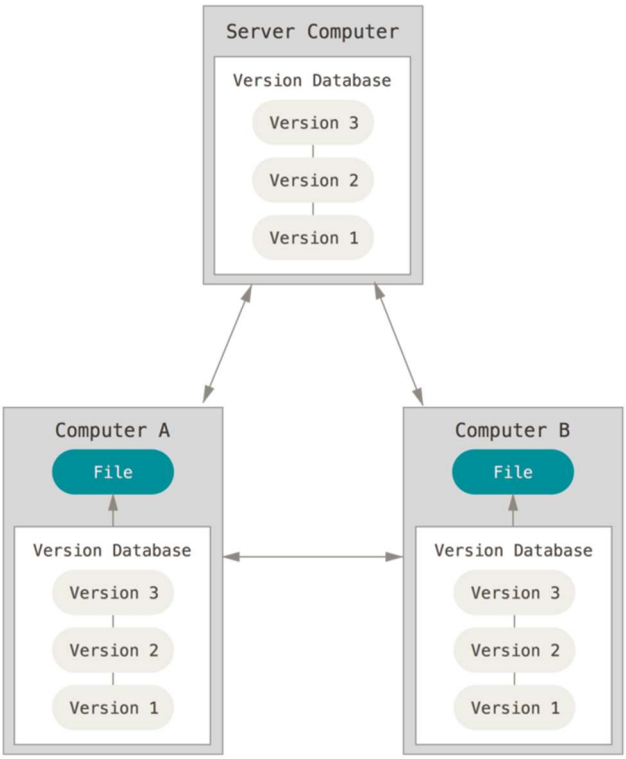



Рисунок 2.2.3 – Структурна схема розподіленої системи керування версіями

Таким чином, при виникненні несправностей на сервері через котрий проводилась розробка, любий клієнтський репозиторій може бути скопійований на інший сервер для продовження роботи. До розподілених СКВ належать Git, Mercurial, Bazaar, Darcs, Fossil та TeamWare.

Найбільш відомою та широко поширеною являється Git. Завдяки простоті використання та вільності програмного продукту дана СКВ здобула прихильність великої кількості розробників по всьому світі.


Рисунок 2.2.4 – Логотип Git

Проєкт був створений Лінусом Торвальдсом в квітні 2005 року.для керування розробкою ядра Linux. Необхідність створення власної СКВ виникла в зв’язку з попередньою втратою ліцензії на використання пропрієтарної системи BitKeeper.

Цікавим є той факт, що початкова розробка велась менше тиждня: 3 квітня 2005 року почалась розробка, а вже 7 квітня код Git керувався неготовою системою. 16 червня Linux було переведено на Git, а 25 червня Торвальдс відмовився від посади головного розробника.

Таким чином ознайомившись з історією та функціональними можливостями розподіленої системи керування версіями “Git” було прийнято рішення інтегрувати систему в процесс розробки програмного продукту. Застосування системи контролю версій дозволить створювати резервні копії проєкту в ході написання програмного коду. Та значно полегшить орієнтування в структурі файлів проєкту при проведенні експериментів з різними модулями програмного продукту.

На цьому моменті необхідно зберегтись.

## 2.3 Мова програмування C# та Unity

Для написання ігрової логіки при застосуванні рушія «Unity» використовується мова програмування C#. Це об'єктно орієнтована мова програмування високого рівня загального застосування. Була створена компанією Microsoft в 2000 році протягом розробки .NET Framework.

Для написання ігрової логіки мовою C# з використанням ігрового рушія Unity виконується прикріплення файлів-скриптів до конкретних ігрових об’єктів. Таким чином при створенні нового користувацького скрипту в вікні редактора Unity отримуємо файл з наступним змістом:

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

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

    }

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

    }
}
```

В створеному файлі одразу вказується простір імен UnityEngine. Таким чином проводиться зв'язок між мовою програмування та інструментами керування властивостями та функціями рушія гри. Стає доступним використання операторів для взаємодії з ігровим рушієм та команди для подальшого написання сирцевого коду ігрової логіки.

Також в автоматичному режимі генеруються функції Start() та Update(). Це особливі функції рушія Unity котрі визначають умови та час виконання описаних в них операцій. Всього в Unity є 5 таких функцій котрі виконуються автоматично:

Awake() – викликається одноразово або коли активний GameObject що містить скрипт ініціалізовано при завантаженні сцени, або коли попередньо неактивний GameObject перемикається в активний стан.

Start() – викликається одноразово при завантаженні скрипту

Update() – викликається один раз на кадр. Використовується для постійного оновлення змінних та виконання функцій пов’язаних з ігровою логікою.

FixedUpdate() – викликається з постійною частотою 50 викликів/секунду. Застосовується для виконання обчислень пов’язаних з фізичною складовою гри.

LateUpdate() – подібно до Update() але викликається в кінці кожного кадру. В деяких ситуаціях корисно мати цикл для перевірки змінних попередньо змінених в Update().

Таким чином маючи доступ до зчитування та зміни властивостей ігрових об’єктів за допомогою C# проводяться зміни над цими параметрами для досягнення необхідних розробнику ігрових механік.

## 2.4 Інтегроване середовище розробки Visual Studio

Для написання всього програмного коду буде використано інтегроване середовище розробки Microsoft Visual Studio Community. Застосування цього інстременту дозволить користуватись перевагами над іншими текстовими редакторами що полягають в підсвічуванні структурних елементів програмного коду: змінні, типи даних, модифікатори доступу, функції та константи.

Також зручною функцією даного IDE функція інтелектуального продовження рядків. Завдяки інтеграції з засобами ігрового рушія це дозволяє отримувати релевантні пропозиції щодо функцій та параметрів певного об’єкта.

Абсолютно незамінним засобом при написанні програмного коду є виділення помилок. Такий інструмент дозволяє виправляти помилки в синтаксисі котрі

можуть виникати в процессі написання скриптів. Таким чином программіст отримує оперативну інформацію щодо помилкових сегментів коду, та може виправити їх одразу.

Ще одним корисним бонусом являється інтеграція з системою керування версіями Git. При знаходженні файлів проєкту в теці – репозиторію Git, середовище розробки розпізнає це та дозволяє виконувати операції над репозиторієм. Можна робити commit змін та переглядати попередні версії файлів з можливістю їх відновлення.

Community версія програмного забезпечення розповсюджується безкоштовно з необхідністю входу через профіль користувача Microsoft через місяць використання.

# 3 КОНЦЕПЦІЯ

***Назва:***

Проста назва AliensLearnWords або скорочена аббревіатура ALW що буквально перекладається як «Прибульці вчать слова». Ця назва чітко зазначує повчальний вміст відеогри а також частково знайомить з історією котру несе в собі гра типу Пригода.

***Аудиторія:***

Подібні відеоігри можна використовувати для вивчення слів дітьми дошкільного віку. Можлива необхідність допомоги батьків в деяких моментах, але навіть так, відеогра залишається засобом зближення дітей та батьків в процессі гри та навчання.

***Маркетинг:***

Продукт можна продвигати за рахунок презентацій та відгуків користувачів. А також засобами таргетингової реклами від компанії Google.

***Короткий опис відеогри:***

Головні події відбуваються в паралельному всесвіті де прибульці прилітають до системи з кількох планет. Засоби навігації були пошкоджені і тому для

налагоджння комунікації з гравцем, прибульцям необхідно допомогти зібрати предмети розкидані по планетах системи. Збираючи нові предмети словатний запас прибульців розширюється, та з часом гравець зможе зрозуміти що саме хочуть сказати ці створіння дивакуватої форми.

Всього для проходження доступні декілька рівнів, кожен з котрих ставить перед собою задачу відкрити певну кількість слів для розуміння. А також режим мережевої гри де кілька гравців можуть змагатися на швидкість збору певних ігрових об’єктів.

# 4 МОДЕЛЮВАННЯ ТА РОЗРОБКА

## 4.1 Застосування Git

Створимо новий репозиторій в теці проєкту попередньо вказавши використання gitignore для Unity. Це дозволить економити час на створенні нових комітів (збереження змін).

Gitignore це текстовий файл в котрому записано перелік файлів та тек котрі повинні ігноруватись при порівнянні попередньо збереженої версії репозиторію з тими що знаходяться в репозиторію в реальному часі.

Коміт в Git це закріплення/збереження змін репозиторію. При його створенні додається запис до історії змін.

Тепер можна буде відслідковувати історію змін проєкту та оцінювати масштабність змін відносно попередніх версій. Завдяки вкладці Changes можна переглягути відмінності в файлах відносно попередньої версії, а на вкладці History можна переглянути список попередніх комітів та відновити стан файлів на певний момент часу (коли було створено вибраний коміт).

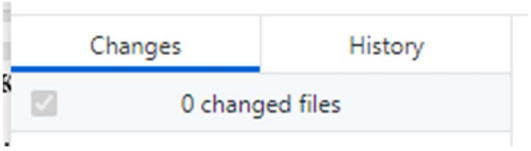



Рисунок 4.1.1 – Вкладки перемикання відображення

## 4.2 Налаштування сцен Unity

Для створення нового проєкту відеогри в Unity спершу необхідно запустити на виконання програмний додаток Unity Hub. Даний застосунок дозволяє керувати проєктами Unity, обирати версію Unity для окремих проєктів, а також вказувати цільову платформу для котрої буде вестись розробка програмного забезпечення. Також тут можна встановлювати нові версії середовища розробки або додавати пакети до встановлених версій.

Створимо новий проєкт Unity натиснувши кнопку New попередньо вказавши цільову версію середовиша 2020.3.5f1. А в наступному вікні необхідно вказати що наш проєкт буде 3D грою, а також вказати назву та місце зберігання. І натискаємо кнопку Create

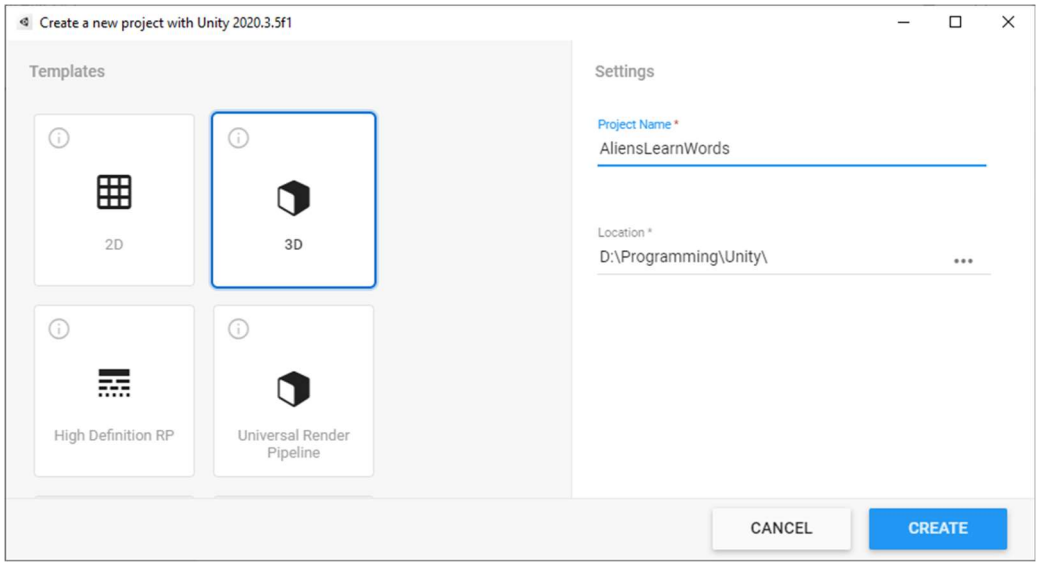



Рисунок 4.2.1 – Вікно створення нового проєкта Unity Hub

Проєкт в Unity складається з файлів-рівнів в котрих зберігається інформація про сцену, об’єкти, сценарії та налаштування. Сцени можуть містити не тільки моделі, але й пусті об’єкти котрі слугують для категоризації об’єктів або ж являються технічними засобами для опису взаємодій компонентів сцени. Кожен об’єкт містить параметри (позиція, обертання, розмір, активність, та інші) значення

котрих можуть змінювати користувацькі скрипти під час безпосереднього виконання програмного коду.

В результаті виконання попередньо описаних операцій отримуємо пусту сцену з попередньо згенерованими об’єктами Main Camera та Directional Light.

Компонент Main Camera в подальшому буде налаштовано як головна камера для відслідковування об’єкту типу гравець. А Directional Light стане головним джерелом світла для сцени.

На цьому етапі необхідно вказати цільову платформу, для котрої буде розроблено програмне забезпечення.. Ма'ємо наступні варіанти платформ доступних для вибору: ПК(Windows, Linux, Mac), Android, iOS, Universal Windows Platform, PlayStation версій 4 та 5, XBOX, tvOS та WebGL. А вказується цільова платформа в вікні Build Settings відкрити котре можна скориставшись комбінацією клавіш Ctrl+Shift+B

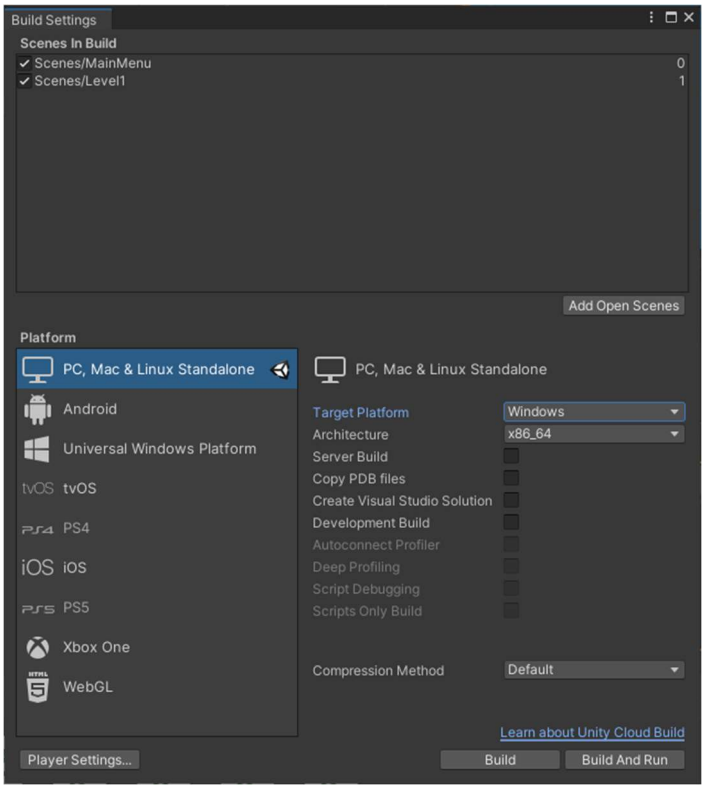



Рисунок 4.2.2 – Вікно параметрів збирання програмного продукту

В подальшому ми будемо використовувати це вікно налаштування для зміни цільової платформи оскільки в кінці створимо версію гри для ПК та для смартфона.

В лівій частині вікна редактора знаходиться список об’єктів що в даний момент розміщено на сцені. Цей список деревовидного типу дозволяє відслідковувати зв’язки між об’єктами та впорядковувати їх в зручний для розробника спосіб.

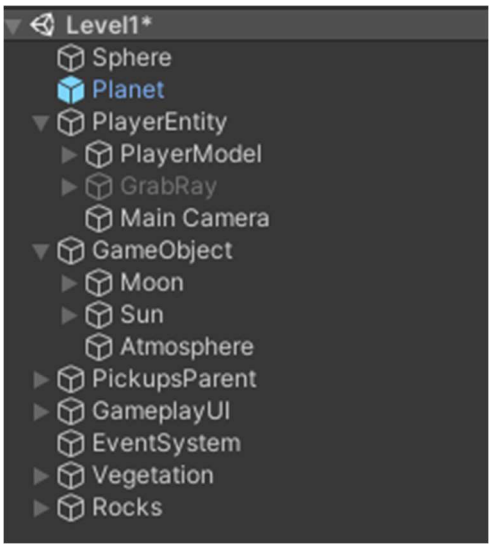



Рисунок 4.2.3 – Панель об’єктів сцени Unity

Темно-сірим кольором позначається неактивний об’єкт. Такий об’єкт на проявляє себе в жоден спосіб та не завантажується до оперативної памяті ло моменту його активації під час безпосереднього виконання програми.

Змінювати властивості об’єктів дозволяє вікно інспектора що за замовчуванням знаходиться в правій стороні вікна. При виділенні об’єкта в цьому вікні показуються його характеристики, закріплені компоненти та скрипти. Тут можна змінювати стан, назву об’єкта, вказати тег для групування об’єктів абож змінити шар на котрому об’єкт буде відображатись.

Далі йдуть компоненти якими об’єкт описується, або задається його поведівна в віртуальному просторі. Базовий компонент Transform дозволяє переглядати та змінювати такі парамерти тіла як позиція, обертання та розмір. Кожен з цих параметрів приймає числове значення, а трансформація відбувається відносно осей координат XYZ.

Компонент Capsule (Mesh Filter) показує що об’єкт являє собою модель-капсулу (сфера проекструдована відносно одного з діаметрів.)

MeshRenderer дає змогу налаштувати відображення проекції об’єкта на монітор користувача. Тут можна налаштувати матеріал/текстури об’єкта а також проходження чи відбивання світла об’єктом.

BoxColider вказує фізичному рушію сприймати об’єкт як паралелепіпед для обчислень перетину з іншими об’єктами. Встановлення флажка isTrigger дозволяє оброблювати події коллізії з іншими об’єктами.

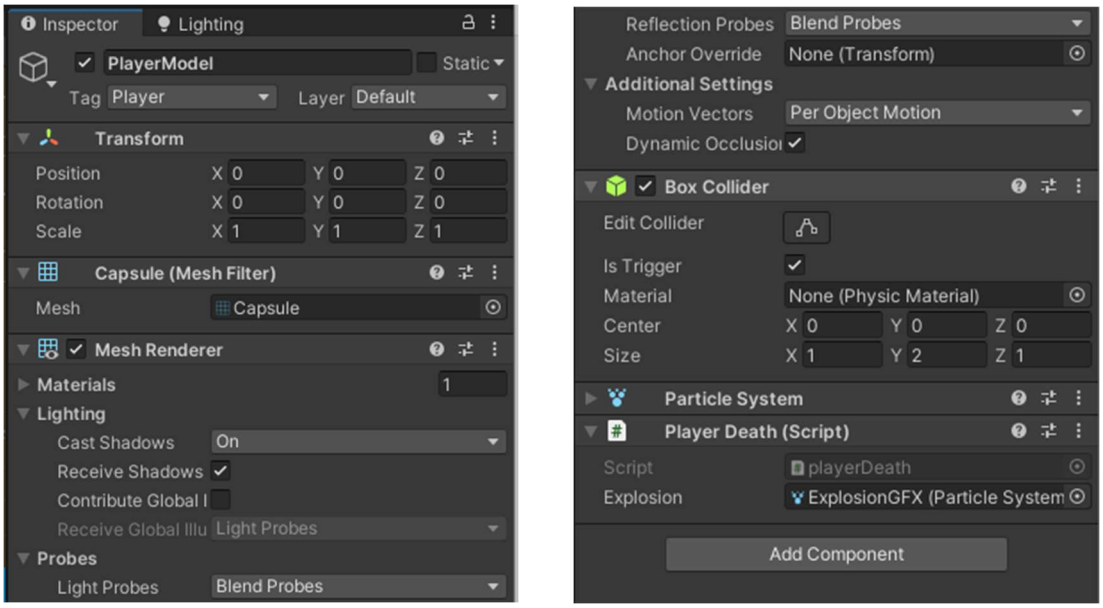



Рисунок 4.2.4 –Панель інспектора об’єктів в Unity

В нижній частині редактора знаходиться засіб перегляду файлової системи поточного проєкту, та консоль для виведення помилок при збірці програми або повідомлення для відлагодження помилок. Перемикання між цими вікнами здійснюється за допомогою вибору необхідного інструмента в лівій верхній частина панелі.

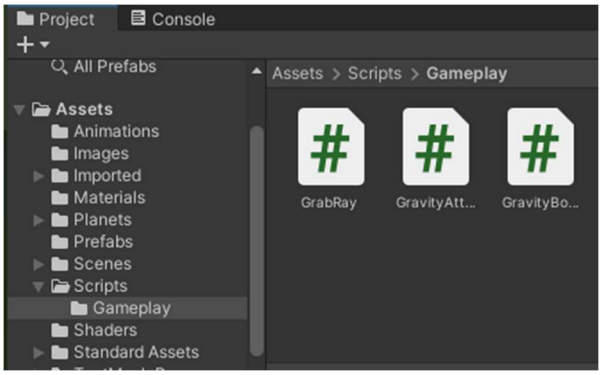



Рисунок 4.2.5 – Перегляд структури проєкта в Unity.

Таким чином можна впорядковувати компоненти програмного забезпечення да створювати нові компоненти для подальшого застосування.

За допомогою кліка правою кнопкою миші в області списку – ієрархії об’єктів необхідно додати до сцени сферу та капсулу. Розмістимо їх одне над іншим та створимо новий файл-скрипт на мові C#. Також одразу створимо файли-матеріали для наших об’єктів, щоб простіше розрізняти їх в просторі редактора.

Для сфери в вікні інспектора необхідно встановити Tag (ярлик) “Planet” таким чином можна буде отримати швидкий доступ до параметрів цього об’єкту при написанні скриптів.

На даному етапі сфера та капсула являють собою схематичне представлення майбутньої гри. Після написання ігрової логіки вигляд ігрових об’єктів буде змінено. Така заміна стає можливою завдяки модульності ігрових об’єктів в Unity. Змінивши значення поля Mesh в компоненті Mesh Filter можна легко змінювати геометрію об’єкта.

Далі на доступному для гравця просторі необхідно розмістити об’єкти для збору. Для цього створимо пустий ігровий об’єкт з назвою “PickupsParent” таким чином об’єднаємо всі подібні ігрові об’єкти в одну группу, а також встановимо для них ярлик “Pickup”.

Для збору буде використана перевірка на колізію між компонентами Box Colider. Створимо об’єкт циліндр, та розмістимо його під моделлю гравця. В вікні інспектора встановимо назву GrabRay. Встановимо неактивний стан за

замовчуванням, та вимкнемо відображення моделі. Пізніше для цього об’єкта буде описана логіка поведінки при натисканні кнопки

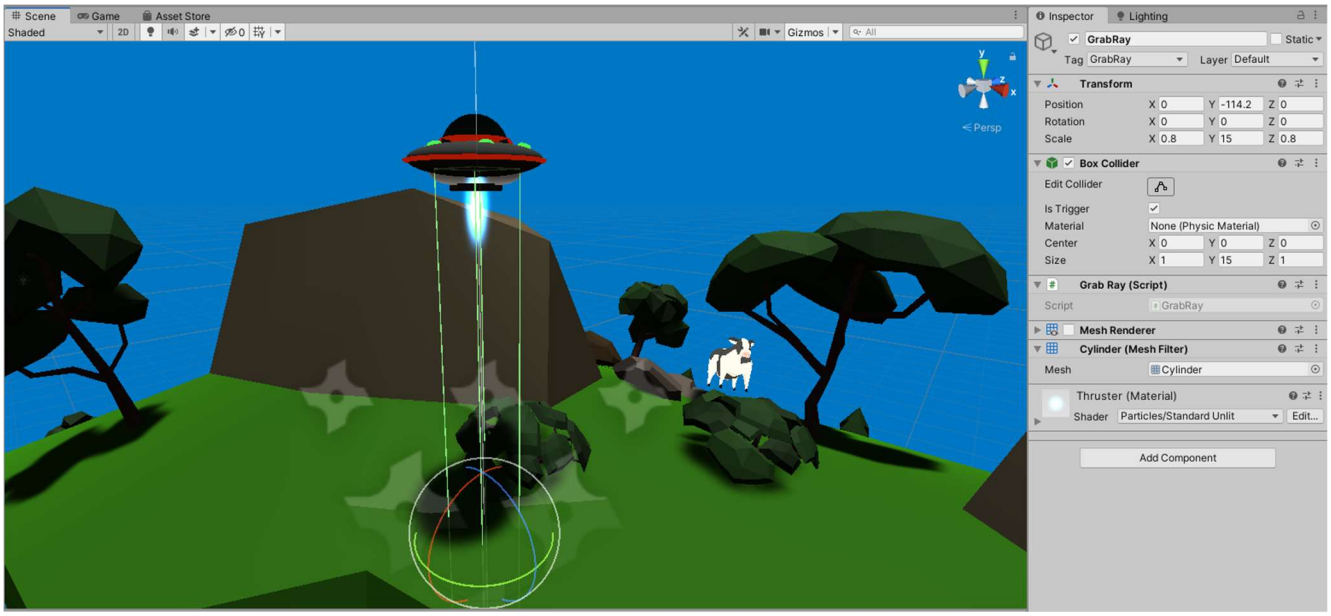



Рисунок 4.2.6 – Ігровий об’єкт GrabRay з відкритим вікном інспектора.

Встановлення змінної IsTrigger дозволяє обробляти зіткнення з іншими ігровими об’єктами.

## 4.3 Скрипти ігрової логіки

Попередньо налаштувавши деякі параметри компонентів приступаємо до написання першого елементу ігрової логіки – гравітаційної взаємодії між капсулою та планетою.

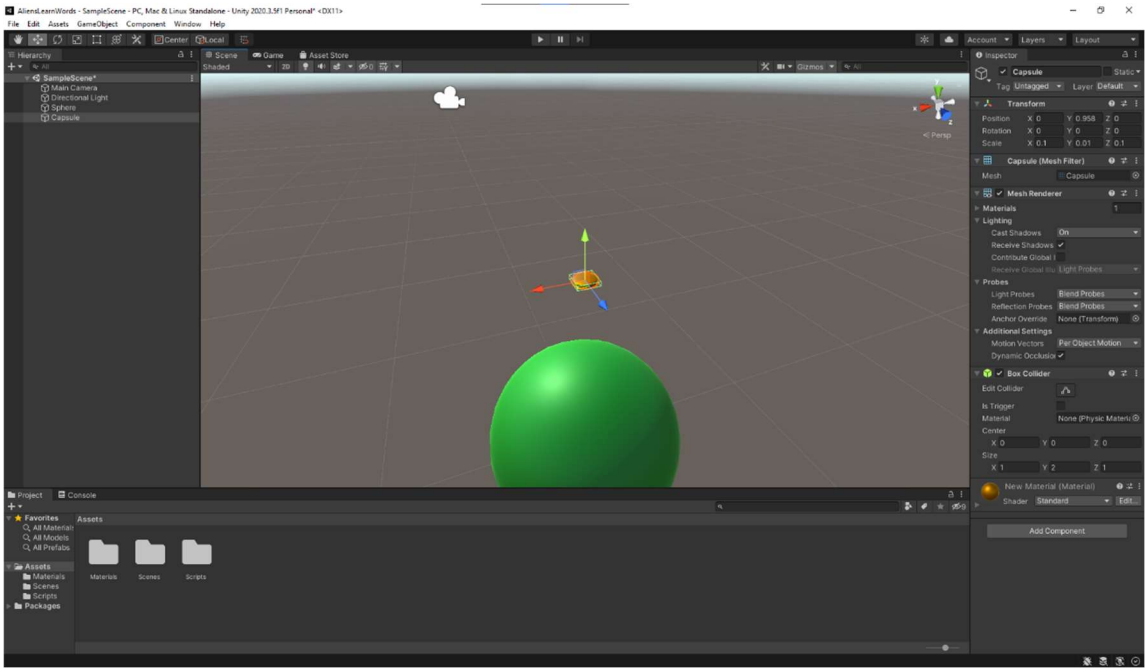


Рисунок 4.3.1 – Головне вікно редактора Unity

Перший скрипт описує поведінку такого типу об’єкту як гравітаційний аттрактор. Сутність що притягує до себе об’єкти з компонентом RigidBody котрі в подальшому будуть передавати свої екземпляри до функції Attract класу GravityAttractor.

```
public class GravityAttractor : MonoBehaviour
{
    public float gravity = -10f;

      public void Attract(Rigidbody body)
      {
            Vector3 gravityUp = (body.position -
transform.position).normalized;
            Vector3 localUp = body.transform.up;

            body.AddForce(gravityUp * gravity);

            body.rotation = Quaternion.FromToRotation(localUp,
gravityUp) * body.rotation;
      }
}
```

В даному скрипті виконується оголошення гравітаційної сталої для даної системи а також функція Attract класу GravityAttractor:

1. Визначення напрямку вектора одиничної довжини від поточної позиції гравітаційного тіла до позиції гравітаційного аттрактора, тобто центра сфери, оскільки позиція визначається центральною точкою.
2. Визначення локального напрямку «вгору» для гравітаційного тіла
3. Застосування сили до гравітаційного тіла в напрямку аттрактора
4. Вирівнювання локальної координати тіла Z з центром планети

Тепер необхідно описати клас гравітаційного тіла, що дозволить отримати бажану взаємодію двох тіл.

```
[RequireComponent(typeof(Rigidbody))]
public class GravityBody : MonoBehaviour
{

      GravityAttractor planet;
      Rigidbody GravityBodyRB;

      void Awake()
      {
            planet =
GameObject.FindGameObjectWithTag("Planet").GetComponent<GravityAttractor
>();
            GravityBodyRB = GetComponent<Rigidbody>();
            GravityBodyRB.useGravity = false;
            GravityBodyRB.constraints =
RigidbodyConstraints.FreezeRotation;
      }

      void FixedUpdate()
      {
            // Allow this body to be influenced by planet's gravity
            planet.Attract(GravityBodyRB);
      }
}
```

Перший рядок позначає необхідність прикріплення компоненту типу Rigidbody до об’єкта класу GravityBody оскільки взаємодія між об’єктами буде відбуватись з огляду на фізичні розрахунки рушія.

Потім оголошуються змінні для прикріплення відповідного атрактора гравітаційному тілу та змінна для зчитування та зміни даних компонента Rigidbody.

Далі описується сегмент коду котрий виконується одноразово при завантаженні скрипта де завдяки попередньо присвоїному тегу «Planet» до об’єкта Sphere проводиться пошук цього об’єкта на сцені та відбувається його ініціалізація як змінна planet типу GravityAttractor. Змінній GravityBodyRB присвоюються поточні значення та параметри компонента об’єкта Rigidbody Значання параметра useGravity встановлюється на false а в додаткових параметрах блокується обертання тіла фізичним рушієм.

Тепер лишається лише записати оновлення положення тіла як виконання функції об’єкта класу GravityAttractor Attract над гравітаційним тілом GravityBodyRB.

Для переміщення гравця напишемо скрипт з наступним змістом:

```
public class PlayerController : MonoBehaviour
{

// public vars
public float walkSpeed = 1f;
public float altitudeDeltaSpeed = 10;
public LayerMask groundedMask;
private Transform planet;
public GameObject grabRay;

// System vars
Vector3 moveAmount;
Vector3 smoothMoveVelocity;
float verticalLookRotation;
Transform cameraTransform;
Rigidbody PlayerRigidbody;

    void Awake()
```

```
	{
		planet = GameObject.FindGameObjectWithTag("Planet").GetComponent<Transform>();
		Cursor.lockState = CursorLockMode.Locked;
		Cursor.visible = false;
		cameraTransform = Camera.main.transform;
		PlayerRigidbody = GetComponent<Rigidbody>();
	}

	void Update()
	{
		if (!UIScript.gameIsPaused && UIScript.playerIsAlive)
		{
			// Look rotation:
			transform.Rotate(Vector3.up * Input.GetAxis("Mouse X") * UIScript.mouseSensitivity);
			verticalLookRotation += Input.GetAxis("Mouse Y") * UIScript.mouseSensitivity;
			verticalLookRotation = Mathf.Clamp(verticalLookRotation, -40, -16);
			cameraTransform.localEulerAngles = Vector3.left * verticalLookRotation;

			// Calculate movement:
			float inputX = Input.GetAxisRaw("Horizontal");
			float inputY = Input.GetAxisRaw("Vertical");

			Vector3 moveDir = new Vector3(inputX, 0, inputY).normalized;
			Vector3 targetMoveAmount = moveDir * walkSpeed;
			moveAmount = Vector3.SmoothDamp(moveAmount, targetMoveAmount, ref smoothMoveVelocity, .3f);

			// Altitude controll
```

```
                if (Input.GetButton("Altitude"))
                {
                      Vector3 alt_change = new Vector3(0.01f, 0.01f,
0.01f);
                      if (Input.GetKey(KeyCode.LeftShift) &&
planet.localScale.x <= 1.1)
                      {
                            planet.localScale += alt_change *
Time.deltaTime * altitudeDeltaSpeed;
                      }
                      if (Input.GetKey(KeyCode.LeftControl) &&
planet.localScale.x > 0.9)
                      {
                            planet.localScale += alt_change *
Time.deltaTime * (-1) * altitudeDeltaSpeed * 2;
                      }
                }

                //Grab ray controll
                if (Input.GetButton("Fire2"))
                      grabRay.SetActive(true);
                else
                      grabRay.SetActive(false);
          }
          else
                grabRay.SetActive(false);
     }
     void FixedUpdate()
     {
          if (!UIScript.gameIsPaused && UIScript.playerIsAlive)
          {
                // Apply movement to rigidbody
                Vector3 localMove =
transform.TransformDirection(moveAmount) * Time.fixedDeltaTime;
```

```
                PlayerRigidbody.MovePosition(PlayerRigidbody.position +
localMove);
            }
        }
}
```

Даний скрипт описує логіку для переміщення гравця по віртуальному простору гри. Тут в функції Update відбувається перевірка поточного статусу гравця (живий/мертвий) та гри (чи гра на паузі) і на основі цих даних відбувається переміщення ігрового об’єкта по поверхні сфери та обертання ігрової камери навколо моделі гравця. Окремо можна виділити зміну стану активності об’єкта GrabRay при натисненні відповідних клавіш. Таким чином відбувається зчитування події перетину компонентів BoxCollider

```
public class pickupAnimation : MonoBehaviour
{
    public bool isColided = false;
    private float rotSpeed = 100;
    public Vector3 startPosition;
    public float distanceToTarget = 0f;
    public GameObject player;
    void Start()
    {
        startPosition = transform.position;
    }

    // Update is called once per frame
    void Update()
    {
        if(isColided)
        {
            float followSpeed = distanceToTarget / 0.35f;
            float shrinkSpeed = transform.localScale.x / 0.25f;
```

```
                transform.position = Vector3.MoveTowards(transform.position, player.transform.position, followSpeed * Time.deltaTime);
                transform.localScale -= Vector3.one * shrinkSpeed * Time.deltaTime;
            }
            transform.Rotate(0, rotSpeed * Time.deltaTime,0);
        }
        private void OnTriggerEnter(Collider triggerObj)
        {
            if (triggerObj.gameObject.CompareTag("GrabRay"))
            {
                distanceToTarget = Vector3.Distance(transform.position, player.transform.position);
                isColided = true;

            }

            if (triggerObj.gameObject.CompareTag("Player"))
            {
                gameObject.SetActive(false);
                UIScript.progressSliderCount++;
            }
        }
    }
```

Даний скрипт описує необхідну поведінку об’єкта типу Pickup при колізії з об'єктом GrabRay. В стані спокою до ігрового об’єкта застосовується обертання зі швидкістю описаною змінною rotSpeed.

При перетині колайдерів виконується функція `private void OnTriggerEnter(Collider triggerObj)`, Котра отримує властивості об’єкта з котрим відбувається перетин. При перевірці ярлика відбувається визначення відстані до цільового об’єкта (модель гравця) а також змінюється значення змінної

isColided на істину. Це активує виконання рядків коду написаних в тілі функції Update, де описано виконання руху в напрямку гравця а також зміна параметра localScale компонента transform. Таким чином реалізована анімація руху та зменшення ігрового об’єкта за заданий проміжуток часу. Після цього відбувається переведення ігрового об’єкта в неактивний стан а до глобольної змінної `progressSliderCount` виконується запис про успішний збір об’єкта.

Наступний скрипт являється компонентом ігрового об’єкта GameplayUI і в ньому зберігаються налаштування користувача та відбувається контроль над ігровим процесом. Також тут відбувається комунікація між системою та користувачем.

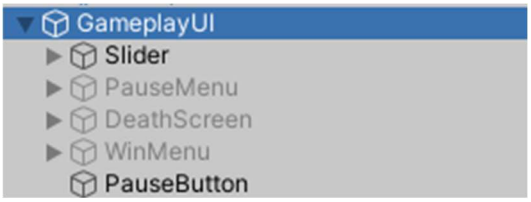



Рисунок 4.3.2 – Ієрархія об’єктів

```
public class UIScript : MonoBehaviour
{
    public static bool gameIsPaused = false;
    public static bool playerIsAlive = true;
    public static float mouseSensitivity = 1f;
    public static int progressSliderCount = 0;
    public bool subMenuIsActive = false;
    public TextMeshProUGUI countText;
    public Slider progressSlider;
    public Slider mouseSensitivitySlider;

    public GameObject fadingPanel;
    public GameObject pickupsParent;
    public GameObject pauseMenuUI;
    public GameObject mainPauseMenu;
```

```
        public GameObject settingsMenu;
        public GameObject musicAudio;
        public GameObject SFXAudio;

        int pickupsToCollectCount;


        private void Awake()
        {
            mouseSensitivitySlider.value = mouseSensitivity;
            mouseSensitivitySlider.minValue = 0.2f;
            mouseSensitivitySlider.maxValue = 3;
            pickupsToCollectCount = pickupsParent.transform.childCount;
        }
        private void Start()
        {
            progressSlider.maxValue = pickupsToCollectCount;
            gameIsPaused = false;
            playerIsAlive = true;
            progressSliderCount = 0;
            gameObject.transform.GetChild(1).gameObject.SetActive(false);
            gameObject.transform.GetChild(2).gameObject.SetActive(false);
            gameObject.transform.GetChild(3).gameObject.SetActive(false);
        }
        private void FixedUpdate()
        {
            progressSlider.value = progressSliderCount;
            countText.text  =  progressSliderCount.ToString()+  "  /  "  +
pickupsParent.transform.childCount;

        }
        void Update()
        {
            if (playerIsAlive)
```

```
            {
                if (progressSliderCount == pickupsToCollectCount)
                {
                    WinMenu();
                }
                else
                {
                    if (Input.GetButtonDown("Cancel"))
                    {
                        if (gameIsPaused)
                        {
                            progressSlider.value = progressSliderCount;
                            if (subMenuIsActive)
                            {
                                settingsMenu.SetActive(false);
                                setSubMenuIsActive();
                                mainPauseMenu.SetActive(true);
                            }
                            else
                            {
                                Resume();
                            }
                        }
                        else
                        {

gameObject.transform.GetChild(4).gameObject.SetActive(false);
                            Pause();
                        }
                    }
                }
                else
                {
```

```
            DiedMenu();
        }

    }
    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        gameObject.transform.GetChild(4).gameObject.SetActive(true);
        Time.timeScale = 1f;
        gameIsPaused = false;
        Cursor.lockState = CursorLockMode.Locked;
    }
    public void WinMenu()
    {
        gameObject.transform.GetChild(3).gameObject.SetActive(true);
        Time.timeScale = 0f;
        gameIsPaused = true;
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }

    public void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        gameIsPaused = true;
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
    void DiedMenu()
    {
        gameIsPaused = true;
        gameObject.transform.GetChild(2).gameObject.SetActive(true);
```

```
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;

    }

    public void BackToMainMenu()
    {
        SceneManager.LoadScene("MainMenu");
    }
    public void QuitGame()
    {
        Application.Quit();
    }
    public void ReloadLevel()
    {

        Scene scene = SceneManager.GetActiveScene();
        SceneManager.LoadScene(scene.name);
    }

    public void setSubMenuIsActive()
    {
        subMenuIsActive = !subMenuIsActive;
    }

    public void setMouseSensitivity()
    {
        mouseSensitivity = mouseSensitivitySlider.value;
    }
}
```

Головною функцією цього скрипту є оновлення інформації на екрані користувача, подача актуальних даних щодо прогресу виконання завдання а також

визначення необхідного інтерфейсу в конкретний момент часу в залежності від введених користувачем даних.

# ВИСНОВОК

В процессі виконання випускної кваліфікаційної роботи було розроблено прикладне програмне забезпечення – відеогра на основі рушія Unity та мови програмування C# в жанрі пригода.

В ході виконання ВКР були вирішені наступні задачі:

Проаналізовано сегмент ринку відеоігор. На основі цього дослідження було зроблено висновок про доцільність проведення розробки програмного продукту для вивчення слів.

Досліджено методи розробки комп’ютерних ігор та написання скриптів ігрової логіки.

Для розробки програмного забезпечення було застосовано технологію програмування мовою C# в поєднанні з рушієм відеоігор Unity. На основі отриманих данних було обґрунтовано доцільність застосування даної технології для створення відеоігор подібного жанру.

# ПЕРЕЛІК ПОСИЛАНЬ.

Unity - Manual: Unity User Manual 2020.3 (LTS) [Електронний ресурс] https://docs.unity3d.com/Manual/index.html

Практическое руководство. Эндрю Троелсен, Филипп Джепикс "Язык программирования C# 7 и платформы .NET и .NET Core" Диалектика, 2018 год, 1328 стр., 8-е изд., ISBN 978-5-6040723-1-8;

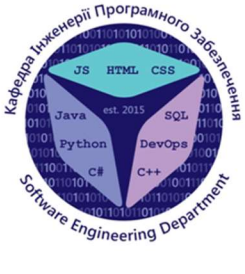
Unity в действии. Мультиплатформенная разработка на C#

Эффективные методы разработки игровых сценариев на C#. Торн А. "Искусство создания сценариев в Unity" ДМК Пресс, 2016 год, 360 стр., пер. с англ. Р. Н. РагимоваДикинсон К.

«Оптимизация игр в Unity 5» DMK-Press, 2017 год, 306 стр., ISBN 978-5-97060-432-8;

# ДОДАТКИ


**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**




Розробка гри «Прибульці вчать слова» в жанрі «пригода» (adventure) з використанням платформи Unity 3D та мови програмування C#

Виконав студент 5 курсу
Групи ППЗ-52
Борисюк О.В.
Керівник роботи: к.т.н, доцент
Шевченко С.М.

Київ - 2021

# Аналоги

| **Назва гри** | **Характеристика** | | |
|---|---|---|---|
| | **3D** | **Сюжет** | **Версія для ПК** |
| Прибульці вчать слова (цей проєкт) | Так | Так | Так |
| Хто у горах? | Ні | Так | Ні |
| Joyful Colorbook | Ні | Ні | Ні |
| Вчи та грай. Українські слова. Словник | Ні | Ні | Так |

# МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

1. **Мета роботи:** розробка комп’ютерної гри для зацікавлення та полегшення вивчення слів Української мови.
2. **Об’єкт дослідження:** Ігровий процесс
3. **Предмет дослідження:** Методи та засоби розробки відеоігор



# ТЕХНІЧНЕ ЗАВДАННЯ

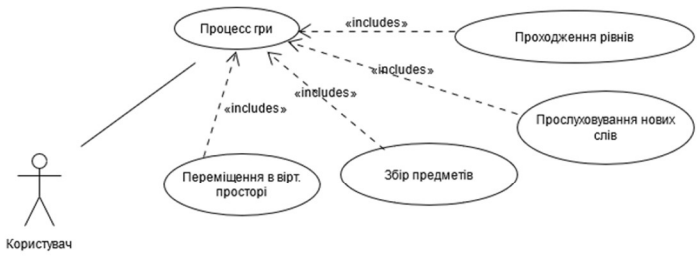
1. Дослідити методи розробки відеоігор з застосуванням рушія Unity
2. Спроєктувати продукт на базі проведеного дослідження
3. Розробити програмну реалізацію переміщення гравця
4. Створити алгоритм для нарахування балів
5. Розробити ігрову логіку переміщення між локаціями
6. Проаналізувати відповідність розробленого продукту початковим вимогам

# ЗАСТОСОВАНІ АРХІТЕКТУРНІ РІШЕННЯ

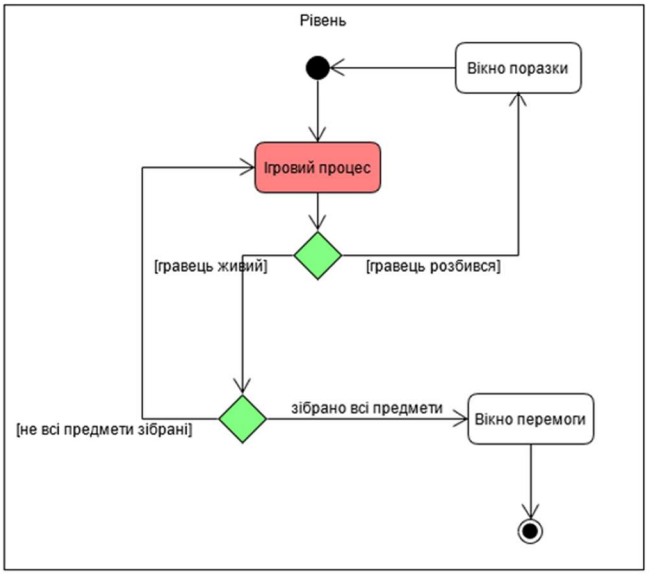
1. Мова програмування C#
2. Ігровий рушій Unity
3. Система керування версіями Git
4. Інтегроване середовище розробки Visual Studio
5. Цільова платформа розробки – ПК, смартфон



# ДІАГРАМА ПРЕЦИДЕНТІВ

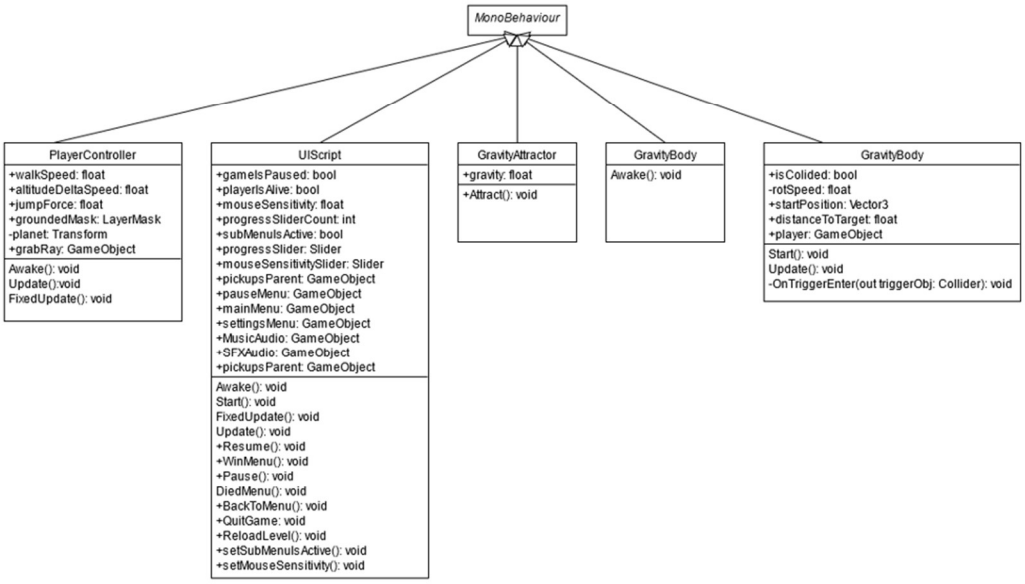


# ДІАГРАМА ДІЯЛЬНОСТІ






# ДІАГРАММА КЛАСІВ



# РЕЗУЛЬТАТ РОЗРОБКИ




# ВИСНОВКИ

1. Проаналізовано сегмент ринку відеоігор.
2. Досліджено методи розробки відеоігор з застосуванням рушія Unity
3. Розроблено програмний додаток на основі проведених досліджень
4. Обгрунтовано доцільність застосування рушія Unity при виконанні задач подібного роду

# ХВИЛИНА ВІДЕО З ДОДАТКУ



# ДЯКУЮ ЗА УВАГУ