МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра інформаційних технологій

Пояснювальна записка

до кваліфікаційної роботи
другого (магістерського) рівня

на тему **РОЗРОБКА ІГРОВОГО ANDROID-ДОДАТКУ НА БАЗІ ПЛАТФОРМИ UNITY**

Виконав: студент 2 курсу, групи ІПЗ
Спеціальності 121
Інженерія програмного забезпечення

_____Терентьєв О. О._____

Керівник_проф. Стрелковська І. В._____

Рецензент _____

Одеса – 2023 р.

ДОВІДКА

кафедри ІТ про виконану магістерську роботу

студента 2 курсу ФКПІ та КН групи ІПЗ

Терентьєва Ігора Олексійовича

на тему РОЗРОБКА ІГРОВОГО ANDROID-ДОДАТКУ НА БАЗІ ПЛАТФОРМИ UNITY

Висновок нормоконтролера пояснювальна записка до кваліфікаційної роботи виконана з незначними порушен. ДСТУ виконано з дотриманням вимог положення ПКТ.

Нормоконтролер викл. каф ІТІ
(науковий ступінь, вчене звання, посада)

[підпис] 15.12.2023 Кийміщина І.В.
(підпис, дата) (і. б. прізвище)

Висновок відповідального за наявність плагіату згідно з сертифікатом ID 1000014 33 унікальність роботи підтверджено.

Відповідальна особа викл. каф ІТІ
(науковий ступінь, вчене звання, посада)

[підпис] 15.12.2023 Кийміщина І.В.
(підпис, дата) (і. б. прізвище)

Попередня експертиза (захист) магістерської роботи
(магістерської роботи чи магістерської роботи)

студ. Терентьєва О.О. проведена "12" грудня 2023р.
(прізвище і.б.)

Висновки Виконана робота відповідає завданню. В роботі розроблено ігровий android-додаток та проведено тестування програмного забезпечення. Робота підтверджує добрий кваліфікаційний рівень випускника. Для розробки додатку в роботі використовувалось ігровий рушій Unity, мова програмування C# та середовище розробки Visual Studio. Магістерська робота відповідає діючим стандартам та рекомендована до захисту.

Члени комісії [підпис] д.т.н. проф Стрелковська І.В
(підпис) (науковий ступінь, вчене звання, посада, прізвище і.б.)

[підпис] к.т.н. проф. Григорьєва Т.Л.
(підпис) (науковий ступінь, вчене звання, посада, прізвище і.б.)

[підпис] к.т.н. доц. Горбачов В.Є.
(підпис) (науковий ступінь, вчене звання, посада, прізвище і.б.)

МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра інформаційних технологій
Освітній ступінь магістр
Галузь знань 12 Інформаційні технології
Спеціальність 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
к.т.н., доц.
Т.І. Григор'єва
"15" 09 2023 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ

Терентьєву Олегу Олексійовичу

1. Тема роботи Розробка ігрового Android-додатку на базі платформи Unity
керівник роботи Стрелковська І. В., д.т.н., професор кафедри ІТ
затверджені наказом закладу вищої освіти від 25.09.2023 р. № 1957, _зі змінами від_04.12.2023 р. № 3102_
2. Строк подання студентом роботи 11.12.2023 р.

3. Вхідні дані до роботи
Технічна документація Unity;
Інформація про конкурентні додатки;
Опис алгоритму покрокової стратегії.

4. Зміст розрахунково-пояснювальної записки
Розділ 1 Аналіз сучасних відеоігр
Розділ 2 Розробка технічного додатку гри
Розділ 3 Розробка програмного коду
Розділ 4 Тестування мобільного додатку
Розділ 5 Аналіз тестування та виправлення помилок

5. Перелік графічного матеріалу (з зазначенням обов'язкових креслень)
Слайд 1 – Тема роботи
Слайд 2 – Основні характеристики роботи
Слайд 3 – Аналіз жанру ігор
Слайд 4 – Аналіз студій-розробників ігор
Слайд 5 – UML-діаграма варіантів використання
Слайд 6 – UML-діаграма станів ігрового додатку
Слайд 7 – Діаграма зв'язків
Слайд 8 – Види тестування

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	25.09.2023-01.10.2023	
2	Огляд і аналіз існуючих аналогів	02.10.2023-08.10.2023	
3	Розроблення діаграми варіантів використання	09.10.2023-15.10.2023	
4	Специфікація функціональних та нефункціональних вимог	16.10.2023-05.11.2023	
5	Проектування та реалізація програмного модуля	06.11.2023-12.11.2023	
6	Перевірка та аналіз результатів роботи ігрового Android-додатку	13.11.2023-19.11.2023	
7	Висновки та рекомендації	20.11.2023-26.11.2023	
	Перелік джерел посилання	27.11.2023-03.12.2023	
	Додаток А Перелік копій демонстраційного матеріалу	04.12.2023-10.12.2023	

Студент _____ О.О. Терентьєв
(підпис)

Керівник роботи _____ І.В. Стрелковська
(підпис)

ВІДГУК

на кваліфікаційну роботу другого (магістерського) рівня

Терентьєва Ігора Олексійовича

на тему «Розробка ігрового Android-додатку на базі платформи Unity»

Кваліфікаційна робота Терентьєв І. О. на тему «Розробка ігрового Android-додатку на базі платформи Unity» належить до актуальних напрямків розвитку інформатизації сучасного суспільства та освіти, оскільки вона фокусується на застосуванні та розробці інформаційних технологій у сфері ігор. У сучасному світі ігри є суттєвою частиною життя людей, надаючи їм можливість відпочивати та отримувати розвагу. Такий підхід висвітлює важливість розуміння та впровадження інновацій у галузі ігрової індустрії.

В роботі розроблено ігровий Android-додаток та проведено тестування програмного забезпечення, яке спрямоване на перевірку відповідності між реальною та очікуваною поведінкою програми. Для розробки додатку в роботі використовуються гейм-движок Unity, мова програмування C# та середовище розробки Visual Studio.

Здобувач Терентьєв І. О. повністю виконав завдання до кваліфікаційної роботи. В процесі роботи здобувач Терентьєв І. О. працював самостійно. Графік консультацій не порушувався. Поставлене завдання виконано у повному обсязі. Пояснювальна записка та демонстраційні аркуші виконано охайно із дотриманням усіх необхідних вимог.

Під час виконання кваліфікаційної роботи здобувач Терентьєв І. О. розібрався з усіма поставленими питаннями та, показав уміння користуватись технічною літературою, було зроблено огляд існуючих платформерів на ігровій платформі «Android». Розроблено групу скриптів, сцену та зручний користувальницький інтерфейс.

Кваліфікаційна робота відповідає вимогам до кваліфікаційних робіт другого (магістерського) рівня та заслуговує оцінки «добре».

Студент Терентьєв І. О. заслуговує присвоєння кваліфікації магістр з інженерії програмного забезпечення за заявленою спеціальністю 121 «Інженерія програмного забезпечення».

Керівник
Декан факультету кібербезпеки
програмної інженерії
та комп'ютерних наук
д.т.н., професор І.В. Стрелковська

РЕЦЕНЗІЯ

на кваліфікаційну роботу другого (магістерського) рівня здобувача
Терентьєва Олега Олексійовича

на тему «Розробка ігрового Android-додатку на базі платформи Unity»

Дана кваліфікаційна магістерська робота належить до актуальних напрямків розвитку інформатизації сучасного суспільства та освіти. Будь-який ігровий двигун надає безліч функціональних можливостей, які застосовуються у різних іграх. Реалізована на Unity двигуні гра отримує всі ці функціональні можливості, крім того, додаються її власні ігрові ресурси і код ігрового сценарію. Основні переваги перед іншими передовими інструментами розробки гри у Unity є: продуктивний візуальний робочий процес і потужна міжплатформенна підтримка.

В роботі розроблено ігровий Android-додаток та проведено тестування програмного забезпечення, яке спрямоване на перевірку відповідності між реальною та очікуваною поведінкою програми. Для розробки додатку в роботі використовується гейм-движок Unity, мова програмування C# та середовища розробки Visual Studio.

До недоліків роботи слід віднести:

- недостатню увагу приділено типам тестування;
- недостатню увагу приділено оформленню графічного дизайну гри.

Проте, зазначені недоліки не знижують цінності виконаної роботи. У цілому, кваліфікаційна робота Терентьєва І. О. відповідає вимогам до випускних кваліфікаційних робіт здобувачів другого (магістерського) рівня та заслуговує оцінки «добре».

Студент Терентьєв О. О. заслуговує присвоєння кваліфікації магістр з інженерії програмного забезпечення за заявленою спеціальністю 121 «Інженерія програмного забезпечення».

Рецензент,
доцент кафедри
комп'ютерних наук,
кандидат технічних наук  О.П.Русу

UNICHECK
by Turnitin

Ім'я користувача:
Анна Серединко

ID перевірки:
1015995238

Дата перевірки:
11.12.2023 23:54:46 MSK

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
17.12.2023 19:12:39 MSK

ID користувача:
100001433

Назва документа: Терентьев_О_диплом___11 (2)

Кількість сторінок: 83 Кількість слів: 11289 Кількість символів: 90615 Розмір файлу: 14.21 MB ID файлу: 1015677904

29.5%
Схожість

Найбільша схожість: 4.11% з джерелом з Бібліотеки (ID файлу: 1015677901)

28.5% Джерела з Інтернету | 902 ..... Сторінка 85

4.45% Джерела з Бібліотеки | 47 ..... Сторінка 93

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

3.12%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за 8 слів та 0%)

3.12% Вилучення з Інтернету | 8 ..... Сторінка 94

Немає вилучених бібліотечних джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 121

РЕФЕРАТ

Пояснювальна записка до магістерської роботи: 79 с., 55 рис., 6 табл., 4 додаток, 8 джерел.

У даній роботі розглянуто аспекти процесу розробки відеоігри на платформі «Android».

Метою роботи є розробка нової гри у жанрі платформер під назвою «Adventus»:

- аналіз ігрових Android-додатків на базі платформи Unity
- специфікація вимог для ігрового Android-додатку;
- проектування ігрового Android-додатку;
- програмна реалізація ігрового Android-додатку;
- тестування ігрового Android-додатку.

У магістерській роботі було зроблено:

- аналіз предметної області;
- огляд і аналіз існуючих аналогів;
- розроблення діаграми варіантів використання;
- специфікація функціональних та нефункціональних вимог;
- проектування та реалізація програмного модуля;
- перевірка та аналіз результатів роботи ігрового Android-додатку.

Результати розробки можуть бути впровадженні будь який магазин додатків на платформі «Android».

РОЗРОБКА ГРИ, ПЛАТФОРМЕР, UML-ДІАГРАМИ, USER INTERFACE, ANDROID, C#, СКРИПТ, СЦЕНА, UNITY, VISUAL STUDIO, МОБІЛЬНИЙ ДОДАТОК, ІГРИ, ТЕСТУВАННЯ, QC, QA, BUG.

ABSTRACT

Explanatory note to the master's thesis: 79 p., 55 figures, 6 tables, 4 appendix, 8 sources.

This work considers the aspects of the video game development process on the Android platform.

The purpose of the work is to develop a new game in the platform genre called "Adventus":

– analysis of game Android applications based on the Unity platform
– specification of requirements for a gaming Android application;
– designing a gaming Android application;
– software implementation of a gaming Android application;
– testing the game Android application.

In the master's work, the following was done:

– analysis of the subject area;
– review and analysis of existing analogues;
– development of a diagram of use cases;
– specification of functional and non-functional requirements;
– design and implementation of the software module;
- checking and analyzing the results of the game Android application.

Development results can be implemented in any application store on the "Android" platform.

GAME DEVELOPMENT, PLATFORMER, UML DIAGRAM, USER INTERFACE, ANDROID, C#, SCRIPT, SCENE, UNITY, VISUAL STUDIO, MOBILE APP, GAMES, TESTING, QC, QA, BUG.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ІС – Інформаційна система;

ПЗ – Програмне забезпечення;

ПС – Програмна система;

ТЗ – Технічне завдання;

ОС – Операційна система;

URL – Uniform Resource Locator (Єдиний вказівник ресурсу);

UML – Unified Modeling Language (Універсальна мовамоделювання);

ERD – Entity-Relationship Diagram (Діаграма сутність-зв'язок);

API – Application Programming Interface (Інтерфейс прикладного програмування).

World-LA (World Level Art) – тип бага який пов’язан з предметами у грі;

STP Issue – баг пов’язаний з алгоритмом поведінки будь-якого NPC;

Missions-LD (Level Design) – будь який баг пов’язаний з налаштуванням рівня;

UI (UserInterface Issue) – будь який баг пов’язаний з «Інтерфесом Користувача»;

Misplaced Texture – баг у якому текстура чи спрайт об’єкту не співпадає з його фізичними властивостями або комонентами;

WTB (walkthrough blocker) – баг через який неможливо пройти гру;

ВСТУП

Актуальністю даної дипломної роботи є одна із сторін процесу інформатизації суспільства та освіти – застосування та створення інформаційних технологій. На сьогоднішній момент арена комп'ютерних ігор неймовірно відома на весь світ. Щорічно виникають нові технології, що орієнтують формування нових можливостей у галузі комп'ютерних ігор.

В наш час неможливо уявити повсякденне життя без постійного притоку інформації. Для задоволення цих потреб використовуються сучасні засоби мультимедіа.

У сучасному світі є дуже багато різних способів провести своє дозвілля деякі проводять його за своїм улюбленим хобі інші за переглядом фільмів, серіалів чи граючи в ігри. У поточний момент майже у кожної людини є телефон, планшет або смартфон, в сучасному житті часто бувають випадки коли людина потрапляє в невелику пробку на дорозі або поки їде на автобусі 2-3 зупинки і чим зайнятися в ці кілька хвилин не завжди зрозуміло начебто і часу достатньо що б можна було витратити на щось корисне, але з іншого боку його і не так багато і тому можна зробити гру в яку можливо заходити на пару хвилин для невеликого відпочинку і отримання задоволення від процесу проходження гри.

Компютерні відеоігри набули небувалої популярності за останні роки та посіли почесне місце на ринку розваг та дозвілля. Результатом став бурний розвиток індустрії розробки комп‘ютерних ігор.

1 АНАЛІЗ СУЧАСНИХ ВІДЕОІГР

1.1 Класифікація відеоігор

У роботі було розглянуто самі популярні додатків конкурентів, які охоплюють область дослідження. Вони є дуже схожі, але мають деякі відмінності.

AdVenture Capitalist [4] – розроблена і видана Hyper Hippo Productions. Вперше був випущений для браузерів і Android в 2014 році. У ній від гравця не вимагають постійно натискати на екран для отримання ігрової валюти, в ній гравець купує різні структури які приносять пасивний дохід валюти і спочатку вони не автоматизовані і необхідно на них натискати щоб отримати валюту, необхідно купувати менеджерів які будуть вже автоматично забирати замість вас дохід зі структури.

Розглянемо особливості кількох жанрів комп'ютерних ігор та його роль розвитку ребенка.

Авантюрні (пригодницькі). Вони спроектовані як анімаційний фільм, але з інтерактивними властивостями – перспективою управління процесом подій. Щоб завершити поставлені завдання, ви повинні мати гарний розум і розвинене логічне мислення. На жаль, більшість ігор передбачають тривалу роботу на ПК. У той час, як діти 7 років за комп'ютером можуть працювати лише 10-20 хвилин. Крім того, цей вид ігор вважається потужним подразником, тому гіперактивним дітям не рекомендується з ними працювати.

Стратегії. Подібні ігри формують у дитини посидючість, уміння планувати власні дії, тренують багатофакторне мислення. На жаль, вони ніяк не підійдуть дітям молодшого шкільного віку через величезний проміжок часу. Шлях до досягнення зазначеної місії зазвичай перегороджують супротивники, з якими необхідно вступити у бій чи бути обдуреним хитрістю. Як і аркадні ігри, вони тренують окомір, увагу, швидкість реакції. Не рекомендується для гіперактивних дітей. Нам потрібен контроль часу з боку дорослих. Симулятор. Вони дають вам можливість перевірити свої сили в нових ситуаціях. Тренуйте окомір, увагу,

швидкість реакції. Не рекомендується для гіперактивних дітей. Нам потрібний контроль часу з боку дорослих.

Логічні. Головоломки, завдання на перестановку фігур, формування малюнка, навчання читання, письма тощо. зазвичай діляться різні завдання, що дозволяє регулювати час роботи дитини за комп'ютером. Вони сприяють розвитку мислення, пам'яті, уваги. Аркадні. Цей тип гри характеризується фрагментацією гри за рівнями, коли нагородою та метою є право перейти до наступного епізоду чи місії. Система нарахування очок та бонусів передбачала особливі заслуги, такі як швидкість проходження, перемога над сильним ворогом, знаходження потаємних дверей тощо. Тренуйте окомір, увагу, швидкість реакції. Не рекомендується для гіперактивних дітей. Необхідний контроль часу з боку дорослих.

Endless (Infinity) runners - ігри на реакцію. Гравцю належить керувати персонажем, ухилятися від перешкод, принагідно збираючи бонуси та монети. Принцип жанру - що далі, то більше хардкора. Це відбивається як під час безпосередньої гри, а й у загальному вигляді гри. У партії, що далі просувається користувач, то швидше біжить персонаж. Чим більше монет ви зберете, тим більше привілеїв зможете прокачати. Тому гра не набридне доти, поки в ній є чого добиватися і куди тікати. Елементи керування зазвичай інтуїтивно зрозумілі з невеликим набором елементів керування. Є ігри, в яких є тільки одна дія-стрибок. Якщо подивитися на загальну концепцію цього жанру, то мета гравця – дійти до точки смерті персонажа. Так, світ жорстокий, але що поробиш. Раннери також є чудовою розвагою для вільних рук. Це ігри, в які грають у громадському транспорті, у черзі чи під час перегляду вітчизняних комедій.

Таблиця 1.1 – Аналіз жанру ігор

Жанр	Піджанр	2D	Динамічність	Сюжет
Екшн	Шутер	x	x	x
	Файтинг	x	x	
	Beat’ em up	x	x	x
	Платформер	x	x	x
Стратегії	Покрокові	x		
	В реальному часі	x		
	Tower Defense	x		
	MOBA	x	x	
	MMORTS	x	x	x
Симулятор		x		
Рольові		x	x	x
Пригоди		x	x	x

1.2 Студії-розробники відеоігор

Capcom-компанія, заснована в 1983 році. У той час з'явилося багато названих компаній, таких як Nintendo. Capcom зробила собі ім'я, коли була випущена ігрова приставка Ness. На рахунку компанії багато пам'ятних ігор, в тому числі Rockman. Але найбільший успіх компанії принесла серія апокаліптичних ігор resident Evil, за якими було знято безліч фільмів.

Naughty Dog-студія, заснована в 1984 році. Засновники компанії працювали у своєму гаражі. Компанія була перейменована в Naughty Dog в 1989 році. Популярність компанії принесла перша частина гри crash Bandicoot, а її розробка почалася в 1994 році. У 2001 році студія була придбана компанією Sony для розробки ексклюзивних ігор для консолей компанії. На даний момент naughty dog знаменита серією популярних невідомих ігор і постапокаліптичній драмою our last.

Infinity World-американська компанія, заснована в 2002 році, яка створює ігри для різних ігрових приставок і ПК. Найбільш важливим брендом компанії є ігри серії Call of Duty. У 2003 році компанія була придбана видавцем ігор Activision. Кожна гра знаменитої серії "conjure" завжди продається мільйонними тиражами.

Bethesda-одна з провідних світових студій з розробки рольових ігор і гонок.1 компанія в першу чергу відома як розробник всесвітньо відомої серії рольових ігор The Elder Scrolls і як розробник апокаліптичного шутера Fallout.

Nintendo-компанія, заснована в 1983 році. Слова "легендарна" недостатньо, щоб описати цю студію, яка зробила кілька революцій в історії ігрової індустрії. Nintendo створив Super Mario, Legend of Zelda, Metroid та багато інших брендів. Компанія, по суті, створила індустрію відеоігор на початку 80-х.крім того, Nintendo змінила спосіб взаємодії з іграми.

Blizzard-компанія, яка подарувала світу стратегії, в які грає весь світ, творці Starcraft і Warcraft 2. Компанія виникла в 1994 році в результаті злиття Vivendi Games і Activison. У тому ж році була випущена їх найлегендарніша гра Warcraft, яка принесла компанії всесвітню популярність і зробила її лідером. Всі ігри цієї студії починаються з першого Warcraft і є бестселерами. У 1996 році компанії вдалося придбати студію Contra Games, яка також розробляла легендарну Diablo. А в 1998 році Blizzard випустила гру Starcraft, яка стала найбільш продаваною грою року і завоювала шалену популярність в Кореї і в усьому світі. Популярність Blizzard різко зросла в 2002 році з виходом 3-й WarCraft і, звичайно ж, в 2004 році, коли з'явилася World of Warcraft, одна з найпопулярніших Mmorpg.

Electronic Arts-компанія, що розробляє ігри різних жанрів, починаючи зі спортивних симуляторів і закінчуючи стратегіями, такими як знамениті EA games. Одна з найстаріших ігрових компаній, заснована Тріпом Копкінсом у 1982 році. Майже весь стартовий капітал був зібраний з його особистих заощаджень. Спочатку EA була просто компанією-видавцем ігор, але вже в кінці 80-х вона почала підтримувати консольні додатки. В даний час під брендом EA випущено безліч спортивних симуляторів, включаючи серії FIFA, NHL, Harry Potter, Need for Speed і The Sims. Багато ігор та франшиз було випущено під логотипом цієї компанії.

Valve була заснована в 1996 році Гейбом Ньюеллом та Майком Харінгтоном, колишніми співробітниками Microsoft. Після покупки ліцензії на движок Quake вони приступили до розробки Half-Life, а для роботи над сценарієм був запрошений

відомий письменник-фантаст Марк Лейдлоу. Гра була продемонстрована на виставці E3 в 1997 році і викликала справжній фурор, якраз через 1 рік після її остаточного виходу. Після успіху Valve випустив кілька ігор та виправлень, включаючи знаменитий Counter-Strike. У 2003 році була анонсована друга частина Half Life, розділена на кілька епізодів. Що ж, сьогодні компанія постачає CS:GO, а до цього 2.

Rock Star Games - той самий розробник, який радує нас 1 з найпопулярніших екшенів. Компанія була заснована в 1998 році як об'єднання відразу багатьох студій. Рок-зірка прославилася завдяки головній франшизі, пов'язаній з компанією з моменту її виходу - GTA, що грає за бандита, легендарної муфті з car thieves Ігри цієї серії виходять з 1997 року, кожен раз залучаючи величезні доходи і нових шанувальників.

Ubisoft-творець assassins, Farrays i sixth heroes - європейська компанія Ubisoft, офіси якої знаходяться більш ніж в 20 країнах, а штаб-квартира знаходиться у Франції. Історія компанії сягає 1986 року, коли вона заснувала Ubisoft у Франції разом з п'ятьма братами. У 1994 році вони відкрили офіс у Канаді та розробили Raymond, гру, в якій Ubisoft та франшиза залишаються донині. У 2000 році Ubisoft приступила до розробки нових ігор, придбавши компанію, яка має право видавати ігри за книгами Тома Кленсі. А в 2011 році компанія створила дочірню студію, яка випускає фільми за мотивами ігор.

Таблиця 1.2 – Аналіз студій розробників ігор

	Шутер	Beat’ em Up	Платформер	Рольова гра	Пригоди
Ubisoft	x	x	x	x	x
RockStar Games	x		x		x
Electronic Arts			x	x	
Valve Corporation	x				x
Blizzard			x		x
Nintendo		x	x	x	x
Infinity World	x				
Bethesda			x	x	x
Capcom			x	x	
Naughty Dog			x		x

1.3 Становлення жанру “Платформер”

Платформери - це своєрідний, досить легко відомий жанр комп'ютерної гри. Він у тому, що персонаж має завдання — пройти різними рівнями, долаючи перешкоди.

При цьому пересуватися належить по різних платформах (звідки й пішла назва), іноді — сходам, трубам та іншим. Платформери в Росії також відомі під назвою "бродилки". Їх характерна мальована графіка, часто — навмисне мультяшна, що підкреслює штучний характер того, що відбувається. Головними героями досить рідко стають люди, хай і умовно зображені. Найчастіше це казкові персонажі, наприклад, русалки чи дракони. Нерідко антропоморфні персонажі.

Платформери настільки характерні, що їх легко впізнають навіть ті, хто ніколи не захоплювався такими іграми. З найпопулярніших родоначальників жанру можна назвати Sonic the Hedgehog, Super Mario, Rockman.

Основні принципи цього жанру. Платформери стали популярними у 1980-х роках. Тоді ж було сформульовано основні засади цих ігор. Вони зводяться до наступного:

- персонаж переміщається рівнями, причому у міру просування проходження вперед стає дедалі складнішим;
- рівні мають секрети, які необхідно приховати;
- активно застосовуються пауер-апи. Це особливі предмети, завдяки яким у персонажів з'являються додаткові можливості. Мова може йти про силове поле, можливість стрибати або літати. Однак їхня дія обмежена в часі;

- простий механізм взаємодії із предметами. Для цього зазвичай достатньо настрибнути або ж доторкнутися. Тобто ніяких додаткових дій не потрібно;
- велика кількість різноманітних, але досить примітивних ворогів. Контакт з ними може призвести до смерті або зменшення здоров'я. Як способи впоратися з проблемою традиційно пропонується ухиляється, біг, стрибки, а також стрілянина (але далеко не завжди);

– поява пасток і перешкод, які потрібно обходити або долати за допомогою спеціальних сил;
– платформи, що рухаються, падіння з яких призводить до втрати здоров'я або життя;
– можливість збирати гроші чи інші ресурси, іноді просто окуляри.

Розвиток даного жанру, на відміну інших напрямів, набагато менше пов'язані з появою підтипів. Найбільшого значення тут набула вдосконалена графіка. Сучасні платформери, навіть якщо йдеться про двомірні, пішли далеко вперед. Їхня барвистість просто вражає.

У цих проектах рівні не позначаються виключно умовно. Навпаки, деталізація стала однією із сильних сторін. А "платформи" все частіше набувають вигляду реальних перешкод, які потрібно долати з тих чи інших причин. Причому нерідко у грі це обігрується так, щоб все мало сенс. Наприклад, герой не просто просувається вперед деякими умовними платформами, а дереться по скелях, перебирається через річку, розбирається з іншими перешкодами.

1.4 Розробка відеогри

Розробка відеоігор-це процес створення відеоігор, і розробниками відеоігор може бути як 1 особа, так і компанія з сотнями співробітників. Гра може бути розроблена кількома людьми з обмеженим бюджетом або з опорою на фінансування від видавців. Тривалість і вартість розробки залежать від складності проекту.

Комерційна розробка відеоігор почалася в 20-х роках 70-го століття, з появою перших аркадних автоматів і домашніх ігрових консолей. Завдяки низькому енергоспоживанню першого комп'ютерного обладнання гра була простою, не вимагала великих витрат на розробку і часу, і один програміст зміг успішно обробити і завершити її. Зі зростанням обчислювальних потужностей і вимог гравців складність проекту також зросла, і стала вимагатися робота всієї команди експертів.

1.4.1 Ролі і посади в розробці відеогри

Гейм-дизайнер. Чим більше компанія, тим вужчу спеціалізацію своїх геймдизайнерів вона може собі дозволити. В AAA-студіях сюжетом займатиметься сценарист, налаштуванням рівнів — левелл-дизайнер, боями — комбат-дизайнер, а балансом — фахівець, який збудує спеціальні математичні моделі. В інді-студіях всі ці обов'язки найчастіше виконує одна людина.

Проджект-менеджер. Менеджер проектів займається постановкою щоденних завдань всім відділів розробки і слідкує за термінами їх виконання. Він розставляє пріоритети, щоб оптимально розподілити навантаження на фахівців та оперативно приймає рішення, якщо щось йде не так, як було заплановано.

Програміст - одна з найбільш високооплачуваних професій в ігровій індустрії з високою планкою входу: вам доведеться зібрати портфоліо з декількох ігор. Підійдуть прототипи на Unreal Engine чи роботи в Unity.

Художник з текстур. Комп'ютерні ігри - копітка робота. Щоб усі предмети в грі набули кольору та фактури, художники повинні їх розфарбувати за допомогою текстур. Для деяких проектів потрібні фотореалістичні текстури, для інших стилізовані. Ці фахівці працюють у команді з моделлерами та технічним відділом, оскільки за допомогою текстур можна досягти більшої деталізації наявних ассетів та суттєво заощадити ресурси. VFX-художники створюють різні симуляції на базі ігрового двигуна: вибухи, руйнування, заклинання, дим, туман, краплі та інші ефекти, що запам'ятовуються, які створять «вау-ефект» у гравця. Хороші VFX-художники потрібні у будь-якій студії, яка створює комп'ютерні ігри.

Технічний дизайнер допомагає наповнити локації предметами та оптимізувати їх так, щоб при появі на екрані вони економно витрачали ресурси комп'ютера користувача. Якою б гарною не була локація, якщо вона сильно уповільнить гру, користувач навряд це оцінить.Звукорежисер – здійснює запис і обробку звуків, музики, мови. Продумує відповідність звуків та ігрових ситуацій.

Композитор – пише музику. Сама музика може виконуватися для запису оркестром або музичним гуртом.

Звукорежисери - технічні фахівці, які відповідають за звукові ефекти та позиціонування звуку. Іноді вони спостерігають за озвученням та створенням інших звукових матеріалів. Композитори, які створюють музику для гри, також входять до складу звукової команди гри, хоча ця робота часто передається на аутсорсинг.

Сценарист – складає загальны сюжети у окремих подіях. Пише діалоги, описує предмети та персонажі.

Актор – задіяний в захопленні руху для персонажів, зйомках відеороликів.

Актор озвучування – озвучує мову персонажів, закадровий голос.

Левел-дизайнер. Професії у геймдеві підбираються під ігри, над якими працює студія. Для одних ігор не потрібні програмісти (якщо це нескладні проекти, які збираються в двигунах-конструкторах), для інших – сценаристи (якщо у грі зовсім не буде тексту). Але рівні є у будь-якій грі, навіть у RPG з відкритим світом. Левел-дизайнери одержують від геймдизайнерів опис того, що має відбуватися на локації, і вигадують, які завдання має вирішувати гравець, щоб отримати максимально цікавий ігровий досвід.

Піар-менеджер – рекламує гру, поширює її рекламу, проводить виставки, презентації.

Тестувальник ігор. Перед тим, як гра потрапить до рук користувачів, потрібно переконатися, що в ній немає серйозних помилок, які зіпсують гравцям враження або змусять зробити рефанд. QA-Engineer займається ручною та автоматизованою перевіркою гри. Він готує тест-кейси і ретельно документує проблеми, щоб відділ розробки зрозумів у чому причина поломки і швидко усунув баг.

1.4.2 Етапи розробки відеогри

Загальний алгоритм розробки комп'ютерної гри [2] мало чим відрізняється від алгоритму розробки будь-якого іншого програмного продукту та включає в себе 3 великі етапи:

– проектування;

– розробка;
– видання та підтримка.

На етапі проектування визначаються мета гри та засоби її розробки. При визначенні мети виділяються ідея, жанр та сеттинг гри. Ідея – це те, що буде спонукати гравця грати в гру, що створюється, і вона дуже тісно пов'язана із жанром. Так, наприклад, основна ідея RPG – дозволити гравцеві прожити свою роль так, як захочеться, а основна ідея шутера –дозволити гравцеві взяти участь у реальних чи вигаданих бойових діях. Таким чином, визначивши основні ідеї гри, жанр буде підібраний майже відразу.

Визначившись із жанром та ідеєю гри, наступним кроком буде вибір мережі. Сеттинг – це середовище, в якому відбуватиметься основна дія гри. Він визначає місце, час та умови дії. Вибір сеттингу може сильно полегшити розробку сценарію для гри, тому його краще вибирати заздалегідь і ґрунтуючись на смаках цільової аудиторії. До засобів розробки в першу чергу відносять програмний код та ігровий двигун. Від їхнього грамотного вибору залежить як швидкість самої розробки, так і працездатність самого продукту надалі. Програмний код в першу чергу залежить від платформи, для якої створюватиметься комп'ютерна гра. Наприклад, якщо гра створюється для браузерів, то логічно буде використання мови Java або Flash, але якщо гра створюється для персонального комп'ютера, оптимальним вибором буде, наприклад, мова програмування C#, C++.

Ігровий двигун відповідає за низькорівневий опис фізики об'єктів, правил рендерингу графіки та ін. При виборі ігрового двигуна першим

справою дивляться з його доступність і вже зроблений вибір мови програмування. Наприклад, ігровий движок Unity дозволяє розробляти ігри мовою C# і C++ він є безкоштовним, якщо середній дохід кампанії не перевищує $100000 на рік.

Після вибору мети гри та засобів розробки, починається другий ета реалізації проекту – технологія. Розробка найбільший і найдовший етап реалізації проекту, він включає велику кількість кроків, без яких неможливо створити

працездатний продукт. Насамперед потрібно визначитися із сюжетом та ігровою механікою. Ігрова механіка ґрунтується на цілі гри, вона визначає всі об'єкти та правила, за якими гравець взаємодіятиме з ними. Зазвичай паралельно із розробкою ігрової механіки йде написання сюжет гри. Сюжет відіграє не маловажну роль, він визначає те, наскільки гравцеві буде цікаво грати у вашу гру. Сюжет представляють у двох варіантах: літературний та режисерський сценарії. Літературний сценарій описує основні події та персонажів гри, які беруть участь у грі. Режисерський же є докладним описом рівнів гри, подій, які на цих рівнях відбуваються.

Так само на даному етапі починається раннє опрацювання графічної складової та дизайну гри. На основі сюжету та заздалегідь обговореного дизайну, створюються ранні концепт-арти, на основі яких згодом буде опрацьовано основний вид гри та персонажів. Після розробки сюжету та ігрової механіки починається найважливіша частина – розробка самої гри. На основі сюжету та концепт-артів починається створення персонажів та об'єктів гри, паралельно із цим йде розробка ігрових рівнів. При розробці ігрових рівнів спочатку створюється його спрощений план, якому схематично зображено сам рівень, а так само зображені предмети, з якими згодом взаємодіятиме гравець.

Після цього створюється перша версія рівня. Зазвичай, вона є просто голою локацією, з мінімумом необхідних для проходження предметів. Ця версія рівня служить для того, щоб протестувати рівень на прохідність. Після тесту рівень починають поступово заповнювати рештою об'єктів.

Незабаром після створення перших рівнів відбувається складання першого прототипу. гри, яку називають альфа-версією гри. Вона необхідна для того, щоб розробник міг провести тестування (альфа-тестування) основний механіки гри, та перевірити наскільки вона відповідає заявленим вимогам. Часто в альфа-версії гри, об'єкти навіть не мають текстур або вони взагалі представлені у вигляді абстрактних об'єктів.

Якщо альфа-версія гри успішно проходить тестування, настає наступний етап розробки – опрацювання механіки та об'єктів гри. На даному етапі йде доопрацювання рівнів та механіки гри, і починають додавати перші сюжетні події

в гру, такі як відеоролики, сюжетні діалоги та кат-сцени. Також виправляються перші помилки та несправності в коді гри, які були виявлені при тестуванні альфаверсії гри. Після цього настає етап створення другого прототипу гри, або як заведено говорити, бета-версії. Бета-версія служить для того, що протестувати гру на несправності, фактично бета-версія є практично готову гру. У ній можуть бути відсутні якісь незначні елементи.

Висновки до першого розділу

Під час аналізу сучасних відеоігор було виявлено, що платформер – це один з найперспективніших жанрів ігор. Він один з жанрів які полюбляють гравці, та які видють самі знамениті студії-розробники відеоігор. Цей жанр може включати у собі велику кількість інших жанрів. Таким чином ми можемо зробити такі під жанри як платформер-шутер, платформер-пригоди, платформер-стратегію та навіть рольову гру-платформер.

Таблиця 1.3 – Планування розробки гри

Назва	Жанр	Під-жанр	Основні механіки
«Adventus»	Платформер	Пригоди	Керування ГГ
			Динамічна камера
			Ворожий штучний інтелект
			Головне меню
			Різновид платформ
			Зручний інтерфейс
			Музика
			Графіка

2 РОЗРОБКА ТЕХНІЧНОГО ДОДАТКУ ГРИ

2.1 Ігровий двигун Unity

Unity - це ігровий движок для розробки двох-або тривимірних додатків та ігор, призначений для операційних систем Windows та OS X. Створене за допомогою Unity ігрове програмне забезпечення працює на таких операційних системах, як Windows, OS X, Android, Apple iOS, Linux, а також на ігрових приставках Wii, PlayStation 3 та Xbox 360. Комп'ютерні ігри та інша продукція, створені за допомогою Unity, підтримують набір бібліотек DirectX та OpenGL.

Цей двигун дозволяє створити сцену, на якій будуть розміщені імпортовані з 3D Max об'єкти взаємодії. особливості:

- кілька сценарних мов програмування: C#, JavaScript (модифікація) та Boo;
- можливість миттєвого запуску гри;
- проста робота з ресурсами через Drag-and-Drop;
- широкі можливості імпорту;
- повністю налаштований і доступний більшості людей інтерфейс;
- кросплатформність;
- потужність, гнучкість і нескінченна розширюваність;
- наявність безкоштовної версії з деякими обмеженнями.

На даний момент Unity є одним з найпростіших, але також час і найефективніших, ігрових двигунів. Його можливості безмежні, а засоби розробки будуть зрозумілі навіть початківцю програмісту. Виходячи з цих параметрів, була обрана саме ця мультплатформне середовище розробки

2.2 Історія Unity

Unity – це один з найпопулярніших ігрових движків на сьогоднішній день, який використовується для створення ігор та програм у різних жанрах, включаючи аркади, шутери, головоломки, гонки та багато іншого. Цей двигун був створений в

2005 році компанією Unity Technologies і з тих пір став відомий завдяки своєму зручному середовищу розробки та безлічі можливостей.

Історія Unity почалася в 2002 році, коли два друзі-програмісти з Данії, Девід Хелгасон і Ніколас Френсіс, почали працювати над своїм власним ігровим двигуном. Їх метою було створити зручне та доступне середовище розробки для мобільних пристроїв. У 2005 році вони випустили першу версію Unity для Mac OS X і продовжили покращувати свій продукт, додаючи нові функції та можливості. Незважаючи на те, що перша версія була дуже простою, вона вже мала кілька ключових функцій, які зробили Unity популярним серед розробників.

До 2006 року Unity почав підтримувати платформу Windows, що розширило його потенційну аудиторію. Крім того, було додано підтримку веб-плеєра, що дозволило іграм, створеним на Unity, запускатися прямо в браузері без необхідності встановлення додаткового ПЗ.

За час своєї історії Unity Technologies випустила багато версій свого ігрового двигуна. Деякі з них були особливо значущими для індустрії ігор та розробки.

У 2009 році Unity випустила версію 2.5, яка додала підтримку iPhone та iPod Touch. Це дозволило розробникам створювати ігри для мобільних пристроїв, використовуючи одне середовище розробки для кількох платформ. Крім того, Unity продовжувала додавати нові функції, такі як підтримка віртуальної реальності, інструменти для створення анімації та нові типи атрибутів для більш точного настроювання поведінки ігрових об'єктів.

Одна з найважливіших версій - Unity 5, була випущена в 2015 році і включала безліч нових функцій, таких як підтримка фізичних матеріалів, багатопотокова підтримка та підтримка WebGL. Ці функції допомогли покращити продуктивність та можливості двигуна.

У 2018 році Unity випустила версію Unity 2018, яка включала нові інструменти для роботи з віртуальною і доповненою реальністю, а також підтримку пристроїв, що працюють на операційній системі Android. , покращений редактор інтерфейсу та нові функції для роботи з розрахованими на багато користувачів іграми.

Крім того, Unity Technologies продовжує працювати над покращенням свого продукту та розвиває нові технології. Наприклад, у 2020 році компанія представила Project MARS – платформу для створення доповненої реальності, яка дозволяє розробникам створювати більш реалістичні та взаємодіючі з довкіллям об'єкти.

Однією з головних переваг Unity є його зручне та інтуїтивно зрозуміле середовище розробки. Вона дозволяє швидко створювати ігрові об'єкти, додавати анімації та налаштовувати їхню поведінку, створювати свої власні шейдери та багато іншого. Крім того, Unity підтримує велику кількість платформ, включаючи PC, мобільні пристрої, консолі та віртуальну реальність.

2.3 Середа розробки Microsoft Visual Studio

Ця система програмування побудована у вигляді інтегрованого середовища розробки, що включає всі необхідні засоби для розробки результуючих програм, орієнтованих на виконання під управлінням ОС типу Microsoft Windows різних версій.

Microsoft Visual C# це сучасна та прогресивна мова програмування, яка включає можливості, доступні в найбільш поширених промислових та дослідницьких мовах. Дотримуючись філософії розробки C, Microsoft ввела в нього кілька потенційно нових можливостей, що збільшують продуктивність розробника за допомогою структурних компонентів мови. Фактично, C# довів, що є мовою, придатною створення високоякісного комерційного програмного забезпечення. Багато можливостей мови програмування C# були створені для чотирьох різних цілей:

Єдина система типів та спрощення способів використання мовою значень та типів посилань.

- розробка на основі компонентів, заснована на таких можливостях як коментарі XML, атрибути, властивості, події та делегати.

– практичні переваги ґрунтуються на унікальних можливостях мови C#, включаючи роботу з безпечними покажчиками, контроль переповнення тощо.

– зручні мовні конструкції, такі як оператори foreach та using, що підвищують продуктивність розробника.

На відміну від систем програмування компанії Borland, система програмування Microsoft Visual C# орієнтована використання стандартних засобів зберігання та обробки ресурсів інтерфейсу користувача в ОС Windows. Microsoft Visual C# забезпечує всі необхідні засоби для створення професійних програм WINDOWS.

2.4 Ескізний ігровий додаток

Ігровий додаток розрахован на його використання одним гравцем. Гра починається з ігрового поля, в лівій частині якого розташовується керований об'єкт (Головний Герой). Гравець управляє персонажем, який може переміщатися по горизонталі вліво та вправо, а також стрибати та розмовляти з неігровими персонажами (NPC).

Безпосередньо переможні умови присутні. Мета гравця полягає в тому, щоб пройти усі доступні рівні у грі. Гра завершується, якщо у головний герой помирає 3 рази. Вмерти ГГ (головний герой) може якщо впаде з платформи, наступить на небезпечний предмет або його вб’є ворог.

Гра містить 13 сцен з яких:

- 1 туторіал.
- 1 головне меню.
- 1 опції.
- 1 вибір рівня.
- 1 титри.
- 8 рівнів.

Гра також містить сюжет.

Ігрова камера не має фіксованого положення а плавно слідує за ГГ.

2.4.1 Діаграма варіантів використання

В ході аналізу вимог до додатка була розроблена UML-діаграма варіантів використання.

Гра розрахована на використання одним гравцем. Гравець управляє головним героєм.

Головний герой – об'єкт, керований гравцем, що переміщається по ігровому полю по горізонталі. Управління здійснюється за допомогою сенсорних кнопок на мобільному пристрої. А також клавіш AD (переміщення), Пробіл (стрибок), та миші (розмова).

У головному меню гравець може виконати наступні дії.

Запустити нову гру – після натискання на дану кнопку запуститься сцена-туторіал, у якій гравця навчать основним механікам даної гри.

Продовжити гру – після натискання на дану кнопку запуститься сцана-рівень який був відкритий останнім. Опції – у даній сцені будуть даступні налаштування гри (прик. «Гучність музики»)

Вибір рівня – кнопка яка запустить сцену на якій будуть показані всі рівні. Також гравець може вибрати будь-який рівень, якщо він відкрит.

Титри – гравець також має можливість дізнатися хто за що відповідав під час створення гри.

Вийти з гри – безпечне вимикання гри.

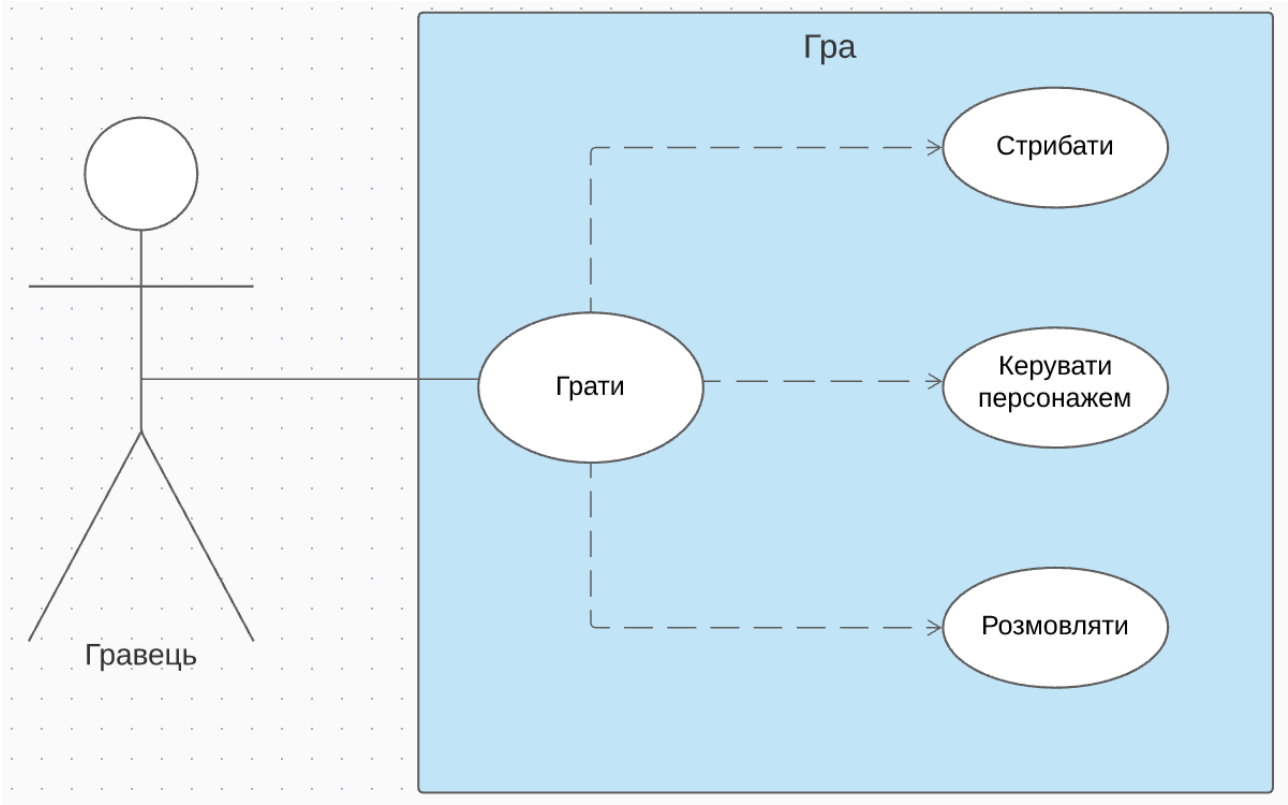


Рисунок 2.1 – UML-діаграма варіантів використання

Висновки до другого розділу

В цьому розділі були визначені вимоги до проєктування, розробки та кінцевого вигляду мобільного додатку. Ці вимоги достатньо описують функціонал та користувацький інтерфейс майбутнього додатку. Далі під час розробки, ці вимоги будуть спрямовувати рішення в правильному напрямку та допоможуть розробити цілісний продукт.

3 РОЗРОБКА ПРОГРАМНОГО КОДУ

3.1 Мова програмування C#

C# (вимовляється як «C Sharp») - це сучасна об'єктно-орієнтована і типобезпечна мова програмування. C# дозволяє розробникам створювати різні типи безпечних та надійних програм, що працюють у .NET. C# належить до добре відомого сімейства мов C і буде знайомий кожному, хто працював з C, C++, Java або JavaScript. Це огляд основних мовних компонентів C# 8 та попередніх версій. Якщо ви хочете вивчати мову за допомогою інтерактивних прикладів, ми рекомендуємо вам вивчити вступні посібники з C#.

Переваги:

Підтримка переважної більшості продуктів Microsoft;

Безкоштовність ряду інструментів для невеликих компаній та деяких індивідуальних розробників — Visual Studio, хмара Azure, Windows Server, Parallels Desktop для Mac Pro та ін.

Типи даних мають фіксований розмір (32-бітний int і 64-бітний long), що підвищує «мобільність» мови та спрощує програмування, тому що ви завжди знаєте точно, з чим ви маєте справу.

Автоматична «складання сміття» Це означає, що нам у більшості випадків не доведеться дбати про звільнення пам'яті. Вищезгадане загальномовне середовище CLR саме викличе збирач сміття та очистить пам'ять.

Велика кількість «синтаксичного «цукору» — спеціальних конструкцій, розроблених для розуміння та написання коду. Вони не мають значення при компіляції. Низький поріг входження. Синтаксис C# має багато схожого на інші мови програмування, завдяки чому полегшується перехід програмістів. Мова C# часто визнають найбільш зрозумілою і придатною для новачків.

За допомогою Xamarin на C # можна писати програми та програми для таких операційних систем, як iOS, Android, MacOS і Linux.

Недоліки:

Пріоритетна орієнтованість на платформу Windows;

Мова безкоштовна тільки для невеликих фірм, індивідуальних програмістів, стартапів та учнів. Великій компанії купівля ліцензійної версії цієї мови обійдеться в круглу суму.

3.1.1 Інші компілятори C#

Будь-який огляд IDE для C# потрібно починати саме з Visual Studio, це класика. Багато розробників, спробувавши програмувати в VS, так і залишаються вірними їй по життю. Її люблять за такі переваги:

- офіційна версія Microsoft не просто розробляє програмний продукт, а й стежить за подальшим його розвитком;
- безкоштовно. Є звичайно і платні продукти, але й те, що поставляється в безкоштовній версії, буде достатньо для зручної роботи;
- можливість використовувати Visual Studio як для C#, але й інших мов програмування. Для цього достатньо встановити спеціальні плагіни;
- надійне зберігання своїх проектів у хмарному просторі.

Можливість роботи та взаємодії для команди розробників. Незважаючи на безліч переваг VS не позбавлена і недоліків. Тож новачкові освоїти весь цей функціонал без сторонньої допомоги буде дуже складно. Також якщо раптом вирішите вибрати платний варіант цього програмного забезпечення, потрібно бути готовим до того, що можуть злетіти налаштування та можуть відбутися зміни в роботі корпоративного сервера.

Project Rider

Ця IDE від компанії JetBrains може і не настільки відома та популярна як Visual Studio, але вже встигла завоювати довіру багатьох розробників. Отже, основні переваги цього інтегрованого середовища розробки є наступними:

- можливість розробки програмного забезпечення від початку до кінця. Це і проектування, і розробка та супровід ПЗ;

– можливість підключити платформу збирання проекту MSBuild, розроблену Microsoft, або Xbuild. А ще це комплексний підхід до організації роботи із CLI-проектами.

Project Rider є кросплатформним програмним забезпеченням, яке добре працює під будь-яку операційну систему. Є можливість запуску одночасно одразу кількох програм. І, звичайно, не можна пропустити – наявність вбудованого функціонала з контролю версій.

Незважаючи на те, що Project Rider – це відмінне інтегроване середовище розробки, яке допомагає прискорити роботу, покращити процес кодування, але воно має й низку недоліків. По-перше, це ще досить новий програмний продукт, тому можливі різні баги, незважаючи на те, що Project Rider постійно допрацьовується. А по-друге, ціна цього програмного забезпечення не маленька. Найпростіша версія коштуватиме майже 140 доларів за рік користування. Хоча є виняток, є безкоштовні версії для студентів.

Eclipse

Багато хто скаже, що це інтегроване середовище розробки зайве в цьому списку, оскільки в основному воно орієнтоване для такої мови програмування як Java. Але все-таки і для розробників, що кодують на C#, тут знайдеться багато цікавого та корисного. Отже, основні переваги Eclipse такі:

– велика кількість плагінів. Яке б завдання не стояло перед розробником, завжди знайдеться відповідний плагін.
– можна сказати напевно, що у Eclipse найбільше число послідовників, що утворює активну спільноту.
– компілятор має високу швидкодію, навряд чи знайдеться гідний суперник у цій справі.
– відладник багатофункціональний, тут можна побачити і перетини, і потоки.
– можливість повного персонального налаштування під себе.

Найважливіше – розробникам використання Eclipse обійдеться безкоштовно. Так, ця IDE спочатку розроблялася під Java, але нині з її допомогою

можна організувати повний цикл розробки на C#. Але є й мінуси у цієї інтегрованої середовища розробки – вона досить складна, і особливо складно новачкам розібратися з таким величезним функціоналом. Також варто врахувати, що є недоробки у всій системі, які не завжди швидко виправляються.

3.2 Створення та використання скриптів

Поведінка ігрових об'єктів контролюється за допомогою компонентів, які приєднуються до них. Незважаючи на те, що вбудовані компоненти Unity можуть бути дуже різнобічними, незабаром ви виявите, що вам потрібно вийти за межі їх можливостей, щоб реалізувати ваші власні особливості геймплея. Unity дозволяє вам створювати свої компоненти, використовуючи скрипти. Вони дозволяють активувати ігрові події, змінювати параметри компонентів, і відповідати на введення користувача яким вам завгодно способом.

Unity спочатку підтримує дві мови програмування:

- C# (вимовляється як Сі-Шарп), стандартний в галузі мову подібний Java або C++
- UnityScript, мова, розроблена спеціально для використання в Unity за зразком JavaScript

На додаток до цих, з Unity можуть бути використані багато інших мов сімейства .NET, якщо вони можуть компілювати сумісні DLL.

3.2.1 Створення скриптів

На відміну від інших Ассет, скрипти зазвичай створюються безпосередньо в Unity. Ви можете створити скрипт використовуючи меню Create в лівому верхньому кутку панелі Project або вибравши Assets> Create> C # Script (або JavaScript / Boo скрипт) в головному меню. Новий скрипт буде створений в папці, яку ви обрали в панелі Project. Ім'я нового скрипта буде виділено, пропонуючи вам ввести нове ім'я.

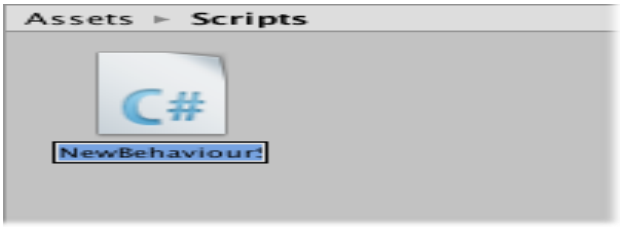


Рисунок 3.1 – Створення скрипта на Юніті

Краще ввести нове ім'я скрипта відразу після створення ніж змінювати його потім. Ім'я, яке ви введете буде використано, щоб створити початковий текст в скрипті, як описано нижче.

3.2.2 Структура файлу скрипта

Після подвійного клацання на скрипт в Unity, він буде відкритий в текстовому редакторі. За замовчуванням Unity буде використовувати MonoDevelop, але ви можете вибрати будь-який редактор з панелі External Tools в налаштуваннях Unity.

Вміст файлу буде виглядати приблизно так:

```
Assembly-CSharp                                   NewBehaviourScript
 1  using System.Collections;
 2  using System.Collections.Generic;
 3  using UnityEngine;
 4
        Скрипт Unity | Ссылок: 0
 5  public class NewBehaviourScript : MonoBehaviour
 6  {
 7      // Start is called before the first frame update
            Сообщение Unity | Ссылок: 0
 8      void Start()
 9      {
10
11      }
12
13      // Update is called once per frame
            Сообщение Unity | Ссылок: 0
14      void Update()
15      {
16
17      }
18  }
19
```

Рисунок 3.2 – Структура файлу скрипта

Скрипт починає взаємодіяти із внутрішніми механізмами Unity за рахунок створення класу, успадкованих від вбудованого класу, званого MonoBehaviour. Можна думати про клас, як про різні плани для створення іншого типу компонента, який може бути приєднаний до ігрового об'єкту. Кожен раз, коли ви приєднуєте скриптова компонент до ігрового об'єкту, створюється новий екземпляр об'єкта, визначений планом. Ім'я класу береться з імені, яке ви вказали при створенні файлу. Ім'я класу і ім'я файлу повинні бути однаковими, для того, щоб скриптова компонент міг бути приєднаний до ігрового об'єкту.

Основні речі, гідні уваги, це дві функції, визначені усередиns класу. Функція Update - це місце для розміщення коду, який буде обробляти оновлення кадру для ігрового об'єкта. Це може бути рух, спрацьовування дій і відповідна реакція на введення користувача, в основному все, що повинно бути оброблено з плином часу у ігровому процесі. Щоб дозволити функції Update виконувати свою роботу, часто буває корисно форматувати змінні, вважати властивості і здійснити зв'язок з іншими ігровими об'єктами до того, як будуть здійснені будь-які дії. Функція Start буде викликана Unity до початка ігрового процесу (тобто до першого виклику функції Update), і це ідеальне місце для виконання ініціалізації.

Замітка для досвідчених програмістів: ви можете бути здивовані, що ініціалізація об'єкта виконується не в функції-конструкторі. Це тому, що створення об'єктів обробляється редактором і відбувається не на початку ігрового процесу, як ви могли б очікувати. Якщо ви спробуєте визначити конструктор для скриптового компонента, він буде заважати нормальній роботі Unity і може викликати серйозні проблеми з додатком.

3.2.3 Управління ігровим об'єктом

Скрипт визначає тільки план компонента і, таким чином, ніякий його код не буде активований до тих пір, поки екземпляр скрипта не буде приєднаний до

ігрового об'єкту. Ви можете прикріпити скрипт перетягуванням Ассет скрипта на ігровий об'єкт в панелі Hierarchy або через вікно Inspector обраного ігрового об'єкта. Є також підміню Scripts в меню Component, яке містить всі скрипти, доступні в проекті, включаючи ті, які ви створили самі. Примірник скрипта виглядає так само, як і інші компоненти у вікні Inspector:

Рисунок 3.3 – Скрипт у вікні Inspector

Після приєднання скрипт почне працювати, коли ви натиснете Play і запустіть гру.

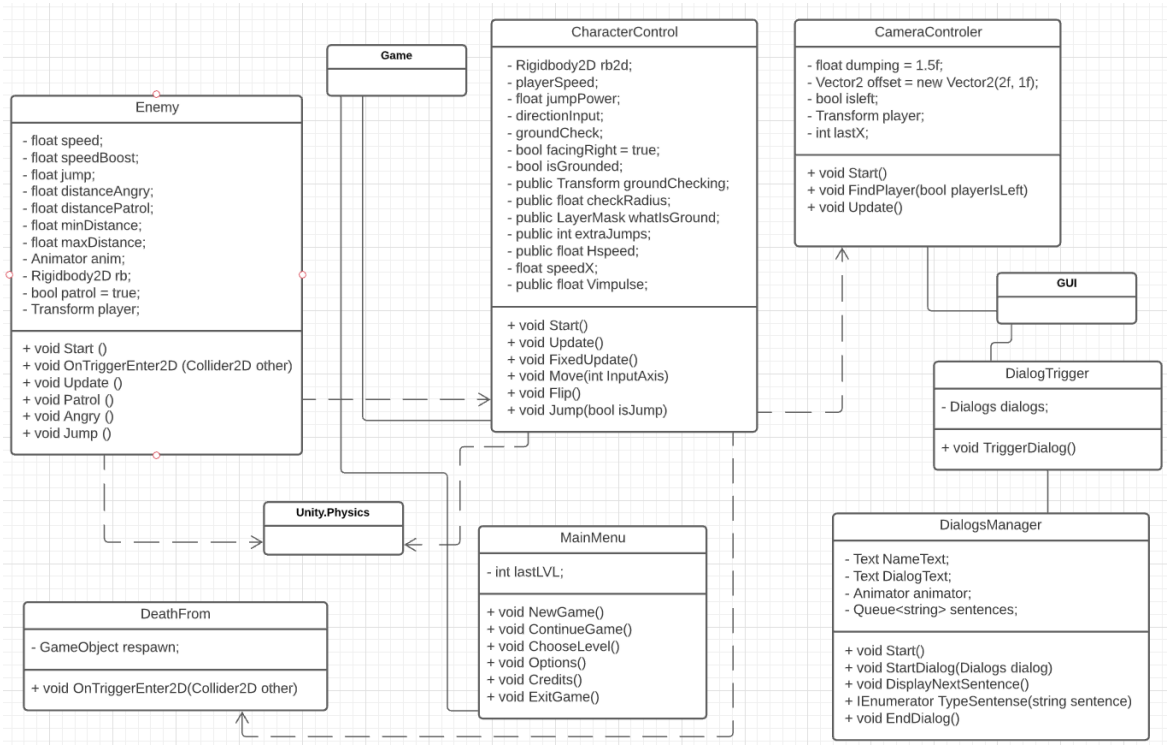


Рисунок 3.4 – UML-діаграма класів

3.3 Створення ігрової сцени

Сцена містить ігрові об'єкти. Її можна використовувати для головного меню,р ізних рівнів та інших цілей. Кожен файл сцени можна розглядати як окремий ігровий рівень. У кожній сцені ви можете розміщувати об'єкти оточення,

ландшафти, бар'єри і фрагменти по дизайну, а також фрагменти, які створюють саму гру.

Додавання компонентів і скриптів.

Коли виділено префаб або будь-GameObject, до нього можна додати додаткову функціональність, використовуючи компонети. Скрипт-це свого роду компонент. Для додавання компонента, виділіть ваш GameObject, та виберіть компонент з меню Component. Ви побачите, що компонент виник в інспектора GameObject'a. Скрипти також містяться в меню Component за замовчуванням.

Розміщення GameObject'ів.

Як тільки ваш GameObject виявився в сцені, ви можете використовувати інструменти Transform Tools для його розташування. Крім того, ви можете змінювати значення властивостей компонента Transform в інспектора для більш тонкої настройки.

Робота з камерами.

Камера-це око вашої гри. Гравець 1 переглядає одну або кілька камер і бачить, що відбувається. Можна переміщати, повертати камери, і прив'язувати до батькові чи матері, як і у випадку з іншими GameObject'амі. Камера - це об'єкт з прикріпленим до нього компонентом Camera, який і забезпечує специфічні для камери функції.

Джерела світла.

За рідкісним винятком, ви завжди повинні включати джерело світла у своюсцену. Існує три типа джерел свытла,і всі вони поводяться трохи по-різному. Важливо, щоб вони додавали атмосферу у вашу гру. Ефективне використання світла є важливим об'єктом вивчення, оскількирі з неосвітлення може повністю змінити настрій гри.

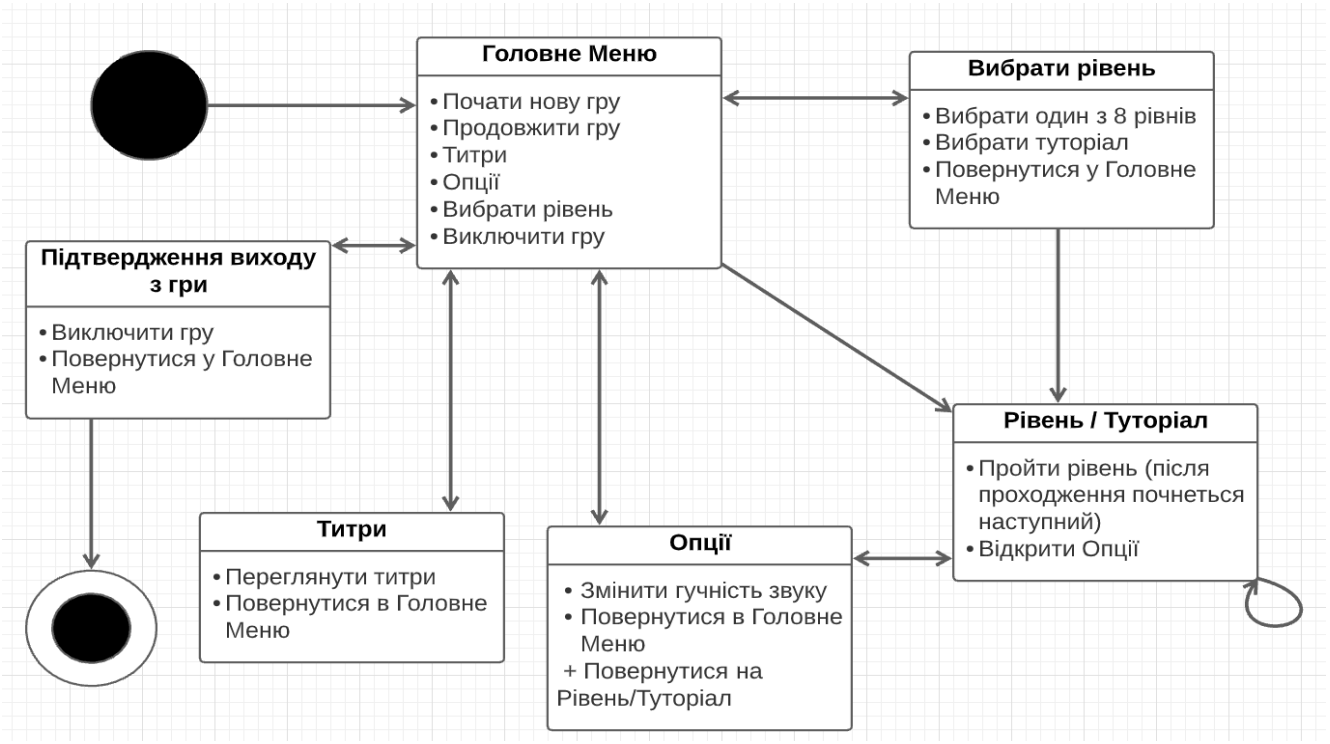


Рисунок 3.5 – UML-діаграма станів ігрового додатку

3.4 Розроблення ігрової сцени

Рисунок 3.6 – Концепт головного меню (Гра запущена)

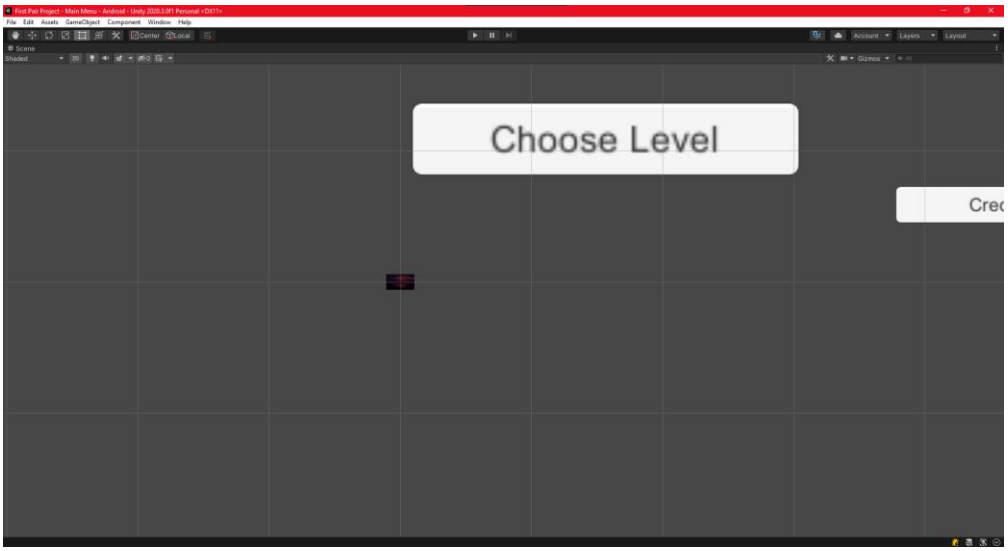


Рисунок 3.7.1 – Концепт головного меню гри (Запущено редактор сцен)

Рисунок 3.7.2 – Концепт головного меню гри (Запущено редактор сцен)

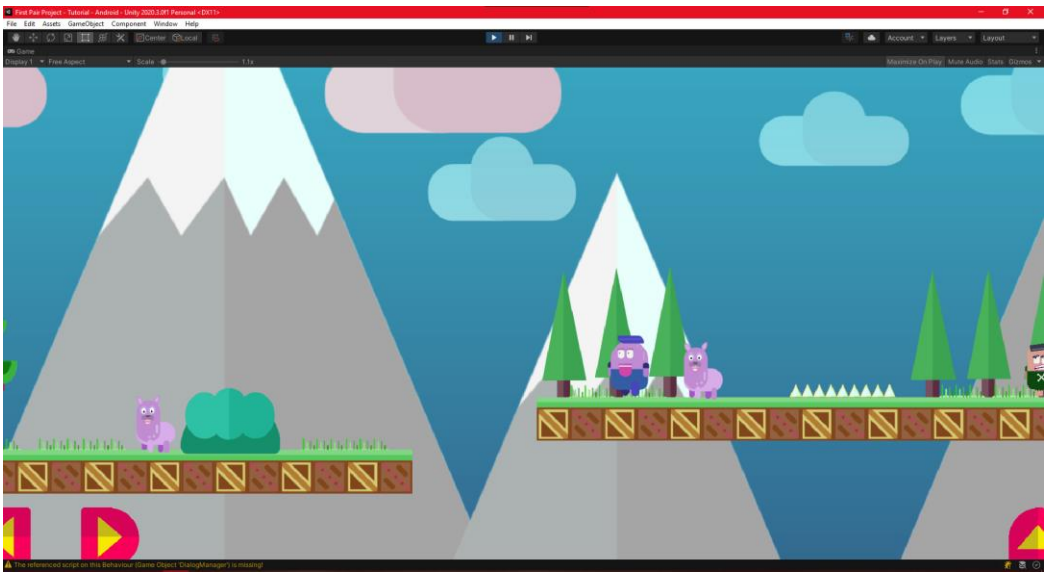


Рисунок 3.8 – Концепт навчального рівня (Гра запущена)

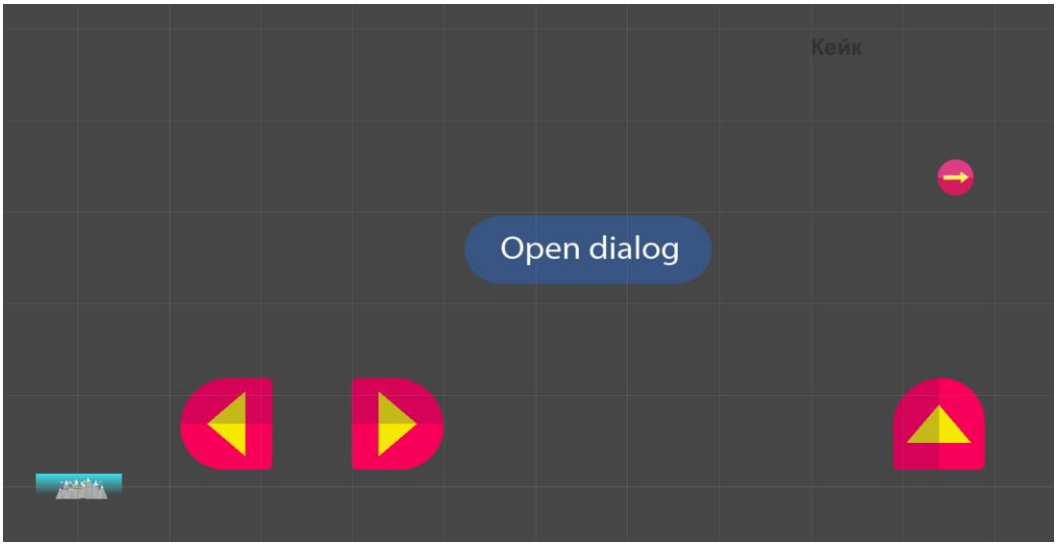


Рисунок 3.9.1 – Концепт навчального рівня (Гра запущена)

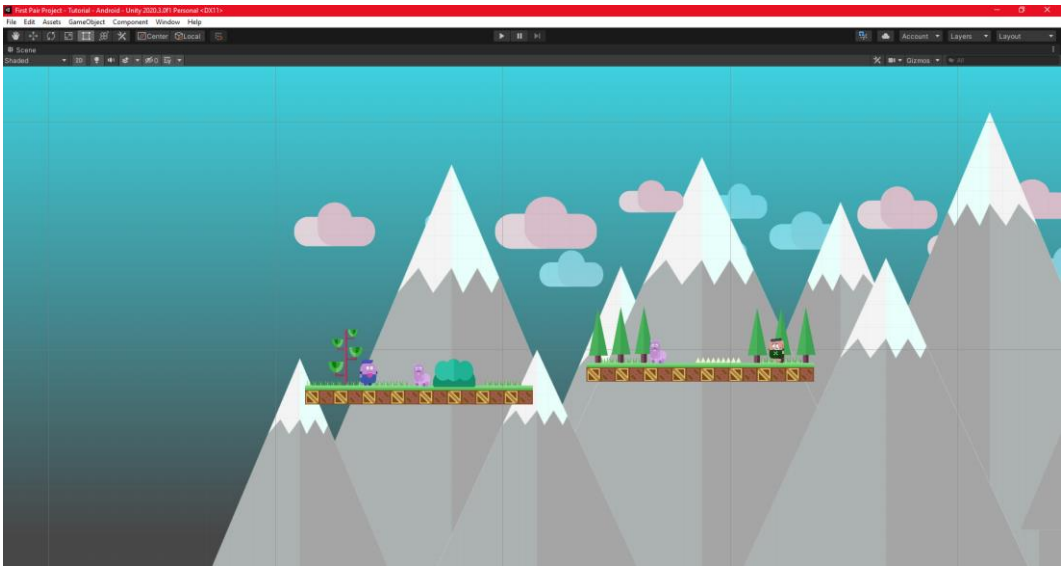


Рисунок 3.9.2 – Концепт навчального рівня (Гра запущена)

Висновки до третього розділу

У даному розділі були розглянуті всі технології, що будуть використовуватись при розробці даного проєкту. Розглянуті технології є безкоштовними. Та мають велику популярність. Обрані технології дозволяють швидко реалізувати додаток та дозволять підтримувати його. Було проведено аналіз з виявленням переваг та недоліків, для розробки додатків було обрано

платформу Unity та мову програмування C#. Саме ця платформа, і ця мова програмування буде найбільш практичною.

4 ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

4.1 Основи тестування

Під час аналізу доступних джерел було проведено дослідження поняття комп'ютерна гра, під час якого було проведено класифікацію комп'ютерних ігор за 4 критеріями, але через порівняльну молодість ігрової індустрії, а також того, що класифікація комп'ютерних ігор не була систематизована, скласти докладну класифікацію не вдалося. Додатково було складено алгоритм розробки відеоігор.

Було проаналізовано популярні засоби розробки. У ході аналізу було проведено їх порівняння та обрано найбільш актуальні засоби розробки для розробників-початківців. Вибір пріоритетних коштів розробки проходив за двома критеріями: доступність та функціональність. При аналізі існуючих розробок, було проведено їх порівняння та виділені їх переваги та недоліки. У ході аналізу стало ясно, що за розробці комп'ютерної гри з простою ігровою механікою варто звернути увагу на додаткові елементи гри, такі як сюжет і графічне оформлення. Це потрібно для того, щоб утримати потенційного гравця та продовжити життєвий цикл розробки. Грунтуючись на отриманій в ході дослідження інформації, було вирішено розробити прототип двовимірного платформера для одного гравця на ігровому движку Unity. Таке рішення було ухвалено з кількох причин:

- двовимірна графіка, на відміну від тривимірної легше у створенні;
- з ігрової механіки, гра жанру платформер простіше реалізується;
- ігровий движок Unity поширюється безкоштовно і дозволяє розробляти програми мовою програмування C#.

Після вибору засобів розробки було розпочато вивчення Unity, а також розробка самого проекту. У ході розробки було вивчено ігровий двигун Unity і були набуті необхідні знання та вміння, а саме:

- створення сцен;
- створення анімацій;
- створення та написання скіптів;

– налаштування об'єктів;
– створення UI;
– компіляція проекту.

Освоєння середовища розробки Unity несе не маловажний характер, так

як у світі індустрія розробки ігор дедалі більше поширюється у суспільстві. Ігри перестали бути лише предметом для розваг, і тепер використовуються і в інших областях, наприклад, у науці чи навчання користувачів. Тому розвиток у цьому напрямі можна вважати одним із найважливіших у сучасному суспільстві.

У ході реалізації проекту було виконано такі завдання:

– вивчено особливості та стан комп'ютерної індустрії Росії;
– обрані жанр, вид та платформа для комп'ютерної гри;
– розроблено сценарій та концепцію основних елементів;
– обрано та вивчено засіб реалізації;
– підготовлені необхідні гри анімації;
– реалізовано прототип гри.

Error – помилка користувача, тобто він намагається використовувати програму іншим способом. Приклад - вводить букви в поля, де потрібно вводити цифри (вік, кількість товару тощо). У якісної програмі передбачені такі ситуації і видаються повідомлення про помилку (error message), з червоним хрестиком які.

Bug (defect) – помилка програміста (або дизайнера або ще кого, хто бере участь в розробці), тобто коли в програмі, що щось іде не так як планувалося і програма виходить з-під контролю. Наприклад, якщо не контролюється введення користувача, в результаті невірні дані викликають краш чи інші «радощі» в роботі програми. Або всередині програма побудована так, що спочатку не відповідає тому, що від неї очікується.

Failure – збій (причому не обов'язково апаратний) в роботі компонента, всієї програми або системи. Тобто, існують такі дефекти, які призводять до збоїв і існують такі, які не призводять. UI-дефекти наприклад. Але апаратний збій, ніяк не пов'язаний з software, теж є failure.

Баг Репорт (Bug Report) – це документ, що описує ситуацію або послідовність дій які призвели до некоректної роботи об'єкта тестування, із зазначенням причин і очікуваного результату. Короткий опис (Summary) – явно вказує на причину і тип помилкової ситуації. Проект (Project) – назва тестованого проекту. Компонент додатка (Component) – назва частини або функції тестованого продукту. Номер версії (Version) – версія на якій була знайдена помилка. Серйозність (Severity) – найбільш поширена п'ятирівнева система градації серйозності дефекту:

- S1 Блокуючий (Blocker)
- S2 Критичний (Critical)
- S3 Значний (Major)
- S4 Незначний (Minor)
- S5 Тривіальний (Trivial)

Пріоритет (Priority) – пріоритет дефекту:

- P1 Високий (High)
- P2 Середній (Medium)
- P3 Низький (Low) Статус (Status)

Статус бага – залежить від процедури і життєвого циклу.

Автор (Author) Творець баг репорту. «Assignee» (Assigned To) – ім'я співробітника, призначеного на вирішення проблеми. Кроки відтворення (Steps to Reproduce) – кроки, за якими можна легко відтвоіити ситуацію, яка призвела до помилки. Прикріплений файл (Attachment) – файл з логами, скріншот або будь-який інший документ, який може допомогти прояснити причину помилки або вказати на спосіб вирішення проблеми.

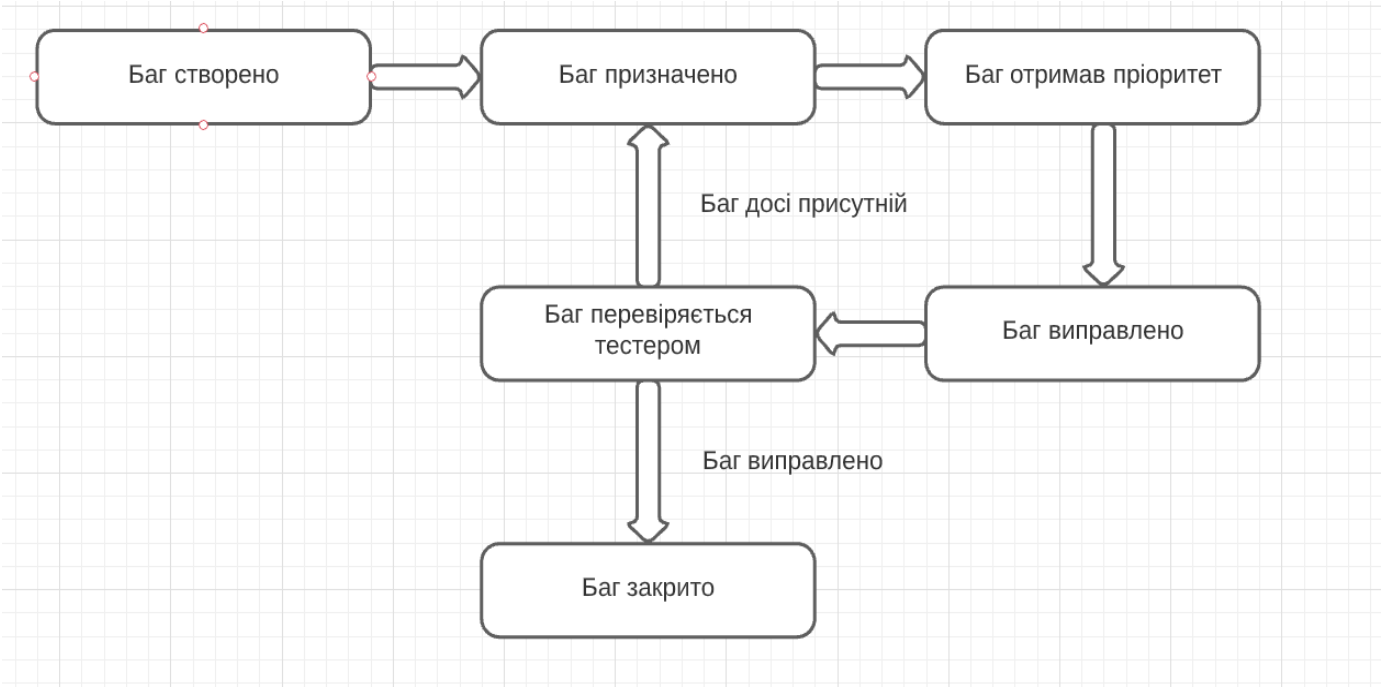


Рисунок 4.1 – Життєвий цикл бага

Діаграма зв'язків – це інструмент управління якістю, що базується на визначенні логічних взаємозв'язків між різними даними.

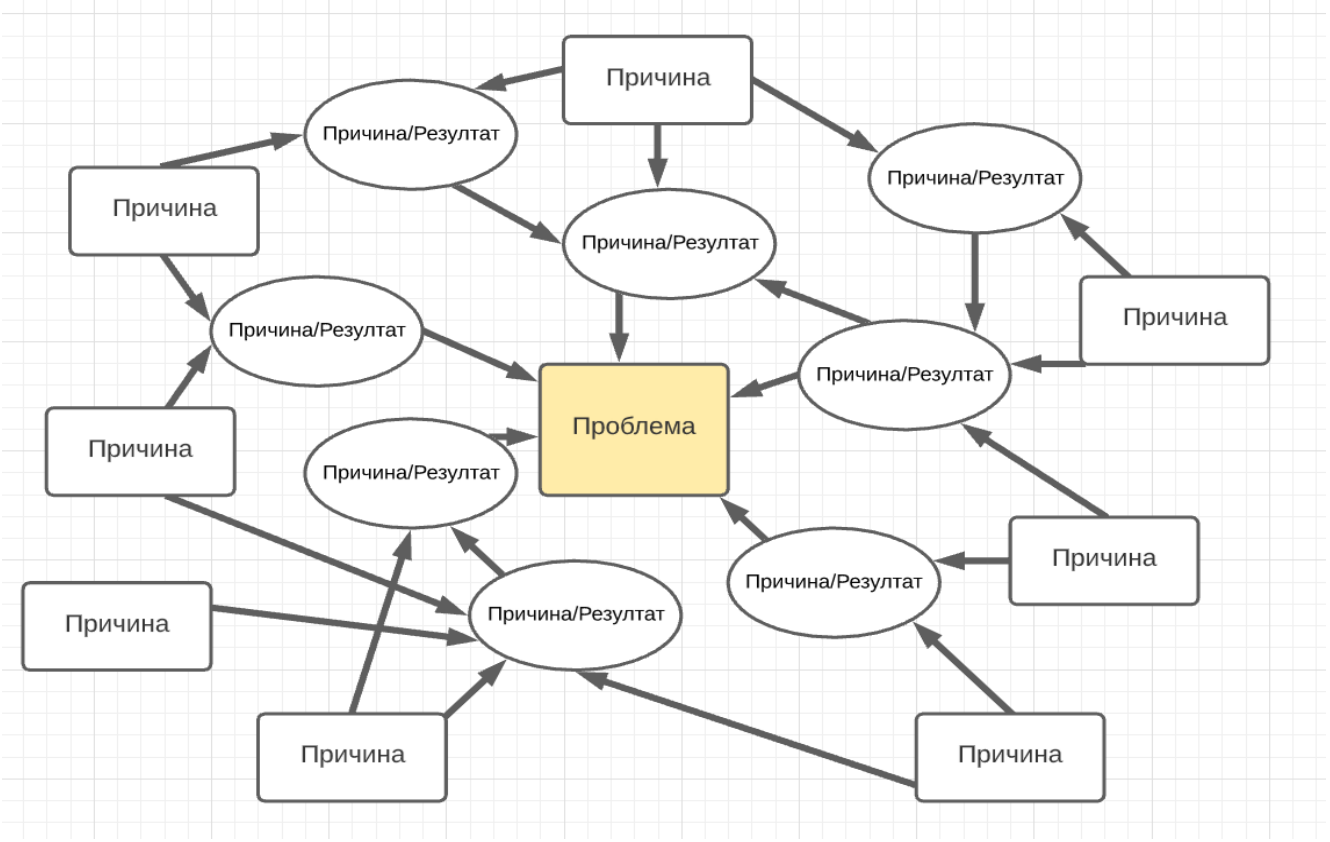


Рисунок 4.2 – Діаграма зв’язків

4.2 Види та типи тестування

Функціональне тестуваннязасноване аналізі специфікацій функціональності компонента або всієї системив цілому з урахуванням заздалегідь заданої поведінки.

Функціональні тести засновані на функціях, що виконуються системою, і можуть виконуватися на всіх рівняхтестування (компоненти, інтеграції, системи, приймання). Як правило, ці функції описуються у вигляді вимог, функціональних специфікацій або випадків використання системи.

Функціональне тестування може проводитися в 2 аспектах:

– попит;
– бізнес-процес.

Тестування з точки зору "вимог" використовує специфікацію функціональних вимог до системи, як основу для розробки тестового випадку. У цьому випадку вам потрібно скласти список того, що тестується, а що ні, розставити пріоритети вимог на основі ризику (якщо це не зроблено вдокументі вимог) і на основі цього розставити пріоритети сценаріїв тестування. Це дозволяє зосередитись на найважливіших функціях під час тестування та не пропустити їх.

Під частестування з точки зору "бізнес-процесів " більшість цих знань про бізнес-процеси використовується для опису сценаріїв, щоденного використання системи. З цієї точки зору сценарії тестування зазвичай базуються на варіантах використання системи[9].

Переваги функціонального тестування – імітує фактичне використання системи. Недоліки функціонального тестування можуть бути відсутні логічні помилки програмного забезпечення та ймовірність надмірного тестування [10].

Тести конфігурації дозволяють побачити, як програма працює з різною роздільною здатністю екрана, різними браузерами, різними операційними системами та різним програмним забезпеченням та обладнанням.

Навантажувальнийтест. Цей тип тестування дозволяє визначити рівень критичного навантаження при роботі з базами даних, інтернет-серверами, мережами та іншими ресурсами. За допомогоюа втоматизованого тестування ви

можете відтворити типовий сценарій дій користувача та багаторазово помножити це число, щодозволить вам протестувати системузі 100 або 10000 активних користувачів.

Тестування можливості.

Перевіряється зручність використання продукту. Наприкладі групи випробовуваних автор описує, як користувач сприймає продукт, як він уявляє собі, як продукт використовується, і як можна використовувати ту чи іншу програму.

Автоматиче тестувння

Автоматизоване тестування програмного забезпечення є частиною процесу тестування на етапі контролю якості, процесу розробки програмного забезпечення. Він використовує програмні засоби для запуску тестів і перевірки результатів виконання, що допомагає скоротити час тестування і спростити процес[6]. Ви можете автоматизувати процес, використовуючи програмні скрипти для тестування в автоматичному режимі. Це корисно, коли продукт часто змінюється, але в той же час має чудові функції. Автоматичне тестування дозволяє відстежувати стабільність всіх функцій продукту, після їх зміни. Автоматизовані тести можна інтегрувати в цикл розробки додатків, запускаючи їх щоразу, коли ви створюєте нову збірку, або створюєте нову версію. При цьому присутність людини необов'язково[3].

Автоматизоване тестування - це непросто виконання тестів. Автоматизація може бути представлена в більшості процесів тестування:

1. Планування та контроль. Тут необхідний обсяг тестування вибирається на основі ризику, або іншої методології. Серед співробітників; контролюйте їх продуктивність; визначайте кількістьт естів, їх статусів та властивості.
2. Аналіз та звітність. Це означає, що ви можете створювати звіти в різних форматах і потреб для всіх. Для робочих груп і для вищого керівництва.
3. Контроль помилок. Сюди входить система відстеження помилок в різних варіаціях.
4. Створення тестів на основі отримани хвимог. Ви можете створити його за допомогою методів запису тавідтворення, або протестувати на одній із

багатьох мовпрограмування, підтримуваних тестовою системою на ваш вибір.

5. Аналізпокриття/трасування. Запуск для оцінки покриття коду, деяка кількість вимог або специфічних функці різними типами тесту, створеного на попередньому етапі.
6. Ми запускаємо тести. Виконання тестових сценаріїв та аналіз результатів[1].

Автоматизоване тестування має переваги:

– більше покриття коду;
– мінімізація людського фактора, при повторній перевірці тестових прикладів;
– один раз закодавані кроки можна використовувати багато разів;
– тест не буде пропущений;
– тест включає в себету ж процедуру.
– убільшості випадків при написанні скрипта виникає помилка;
– скрипт також може містити помилки.

Тест інтерфейс у користувача-перевірте інтерфейс, щоб переконатися, що він відповідає таким вимогам, як розмір,шрифт та колір.

Тестування безпеки-це стратегія тестування, яка використовується для перевірки безпеки системи та використовується для аналізу ризиків, пов'язаних із забезпеченням комплексного підходу до захисту додатків, хакерських атак, вірусів та несанкціонованого доступу до конфіденційних даних.

Тестування насумісність-це функціональний тест, який перевіряє здатність програми взаємодіяти з 1 або більше компонентами або системами і є тестом на сумісність.

Таблиця 4.1 – Види тестування

Функціональні	Нефункціональні		Пов'язані зі змінами
Функціональне тестування	Тестування установки		Димове тестування
Тестування інтерфейсу	Тестування зручності		Регресійне тестування
Тестування безпеки	Тестування на відмову і відновлення		Повторне тестування
Тестування взаємодії	Конфігураційне тестування		Тестування збірки
-	Всі види тестування продуктивності		Санітарне тестування
-	Performance Testing	Stress Testing	-
-	Stability Testing	Volume Testing	-

Тестування стабільності або надійності. Завданням тестування стабільності є перевірка працездатності програми при тривалому тестуванні із середнім рівнем навантаження.

Тестування установки направлено на перевірку успішної інсталяції та настройки, а також поновлення або видалення програмного забезпечення.

Регресійне тестування – це вид тестування спрямований на перевірку змін, зроблених в додатку або навколишньому середовищу. Регресійний можуть бути як функціональні, так і нефункціональні тести.

Санітарне тестування – це вузьконаправлене тестування достатню для доказу того, що конкретна функція працює згідно із заявленими в специфікації вимогам. Є підмножиною регресійного тестування. Використовується для визначення працездатності певної частини програми після змін вироблених в ній або навколишньому середовищу. Зазвичай виконується вручну.

4.3 Виявлені баги під час тестування

Об’єкт має компонент CircleCollider2D та заважає проходженню гри (WTB).

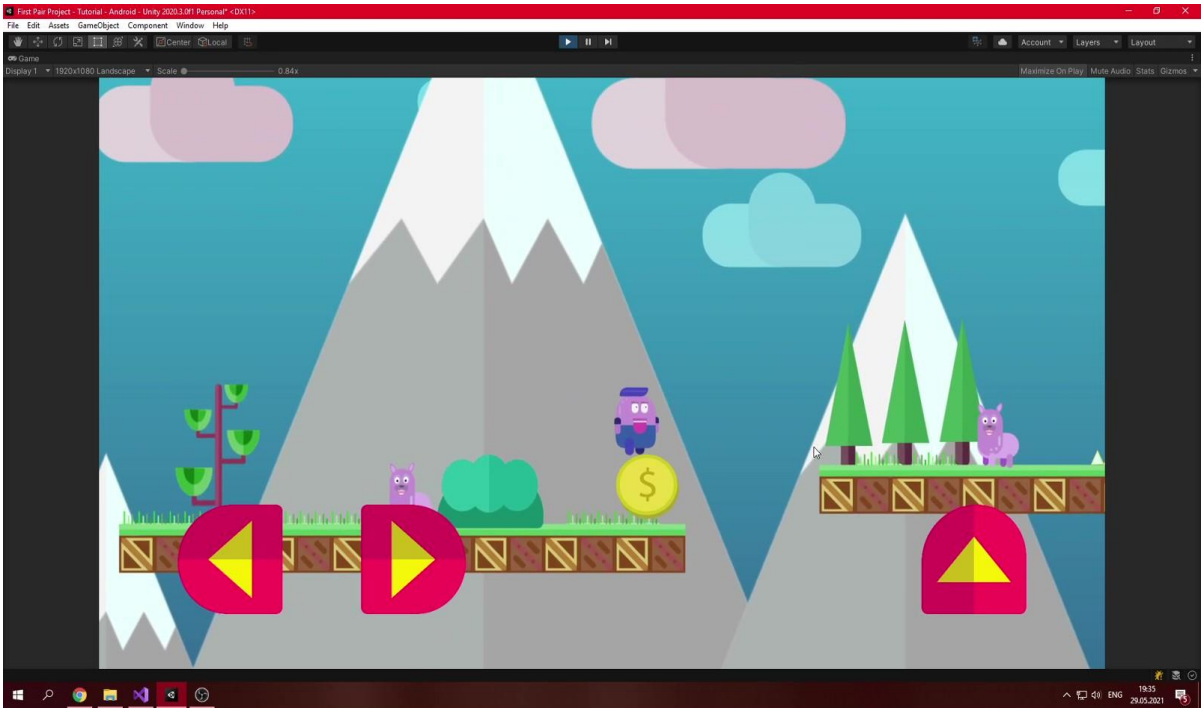


Рисунок 4.3 – Приклад WTB бага під час тестування

WTB (walkthrough blocker) – баг через який неможливо пройти гру. Зазвичай такі баги мають найбільший приортет та повинні бути виправлені одразу після знаходження.

Спрайт об’єкта не співпадає з компонентом BoxCollider 2D (Misplaced Texture).

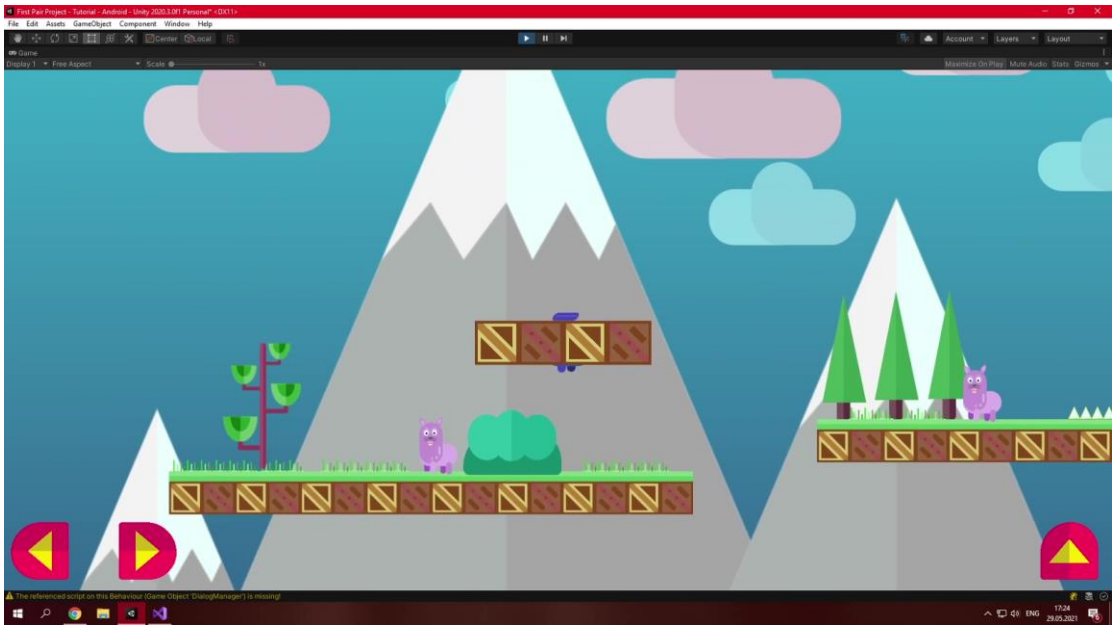


Рисунок 4.4 – Приклад Misplaced Texture бага під час тестування

Misplaced Texture – баг у якому текстура чи спрайт об’єкту не співпадає з його фізичними властивостями або комонентами. Такі типи багів також можуть бути WTB.

Під час гри камера встановлена не правильно (UI).

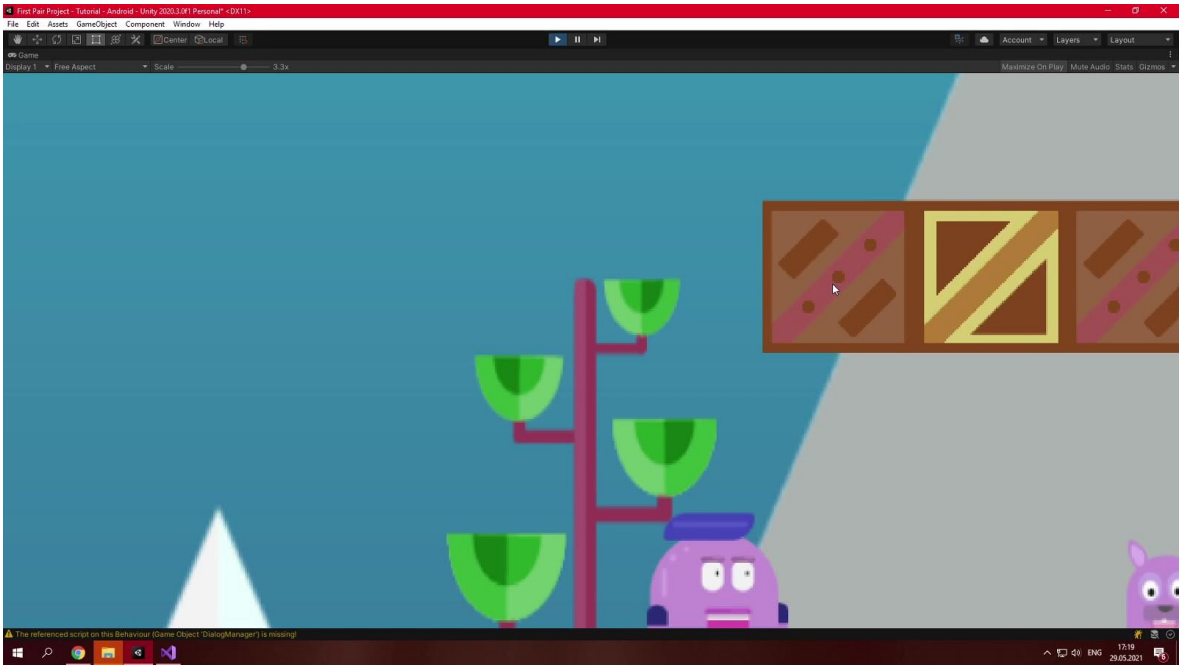


Рисунок 4.5 – Приклад UI бага під час тестування

UI (UserInterface Issue) – будь який баг пов’язаний з «Інтерфесом Користувача». Баги такого типу також можуть бути WTB.

Відсутній інтерфейс на страрті другого рівня.

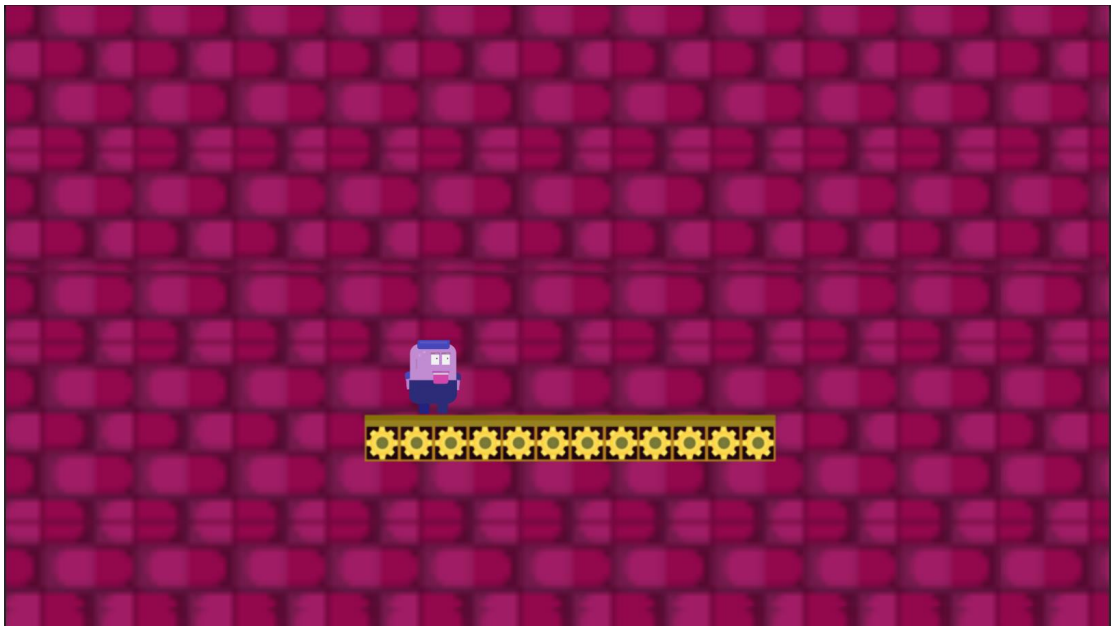


Рисунок 4.6 – Приклад UI та WTB бага під час тестування

Велика швидкість платформи, яка рухається по вертикалі (Missions-LD).

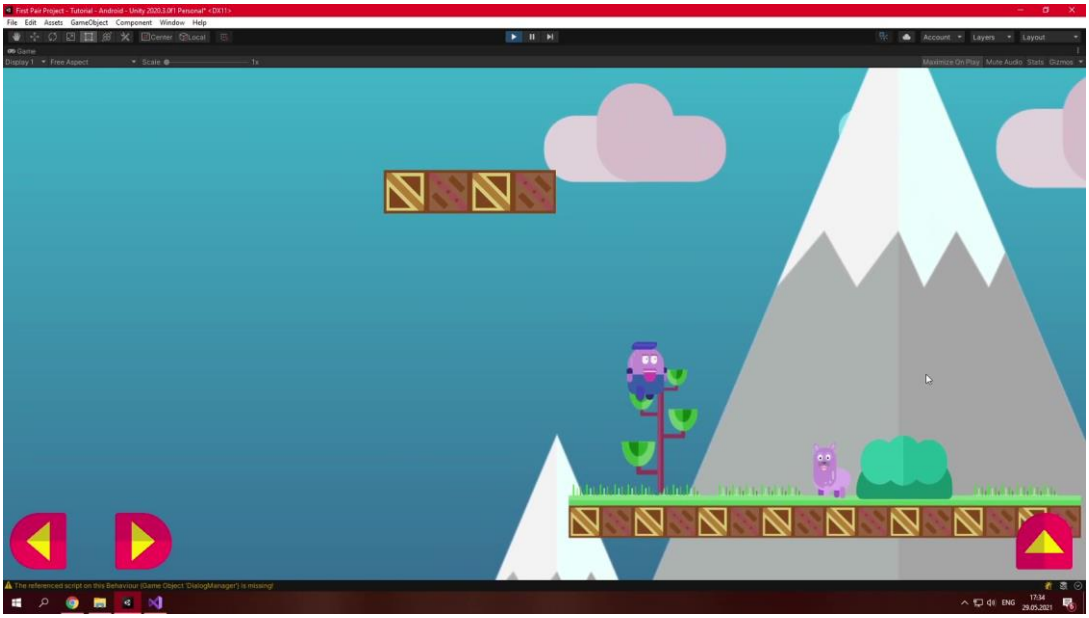


Рисунок 4.7 – Приклад Missions-LD бага під час тестування

Missions-LD (Level Design) – будь який баг пов’язаний з налаштуванням рівня. Баги такого типу також можуть бути WTB.

Порушено STP у ворожого NPC. Він наступає на шипи, які мають перешкоджати гравцю шлях (STP Issue).

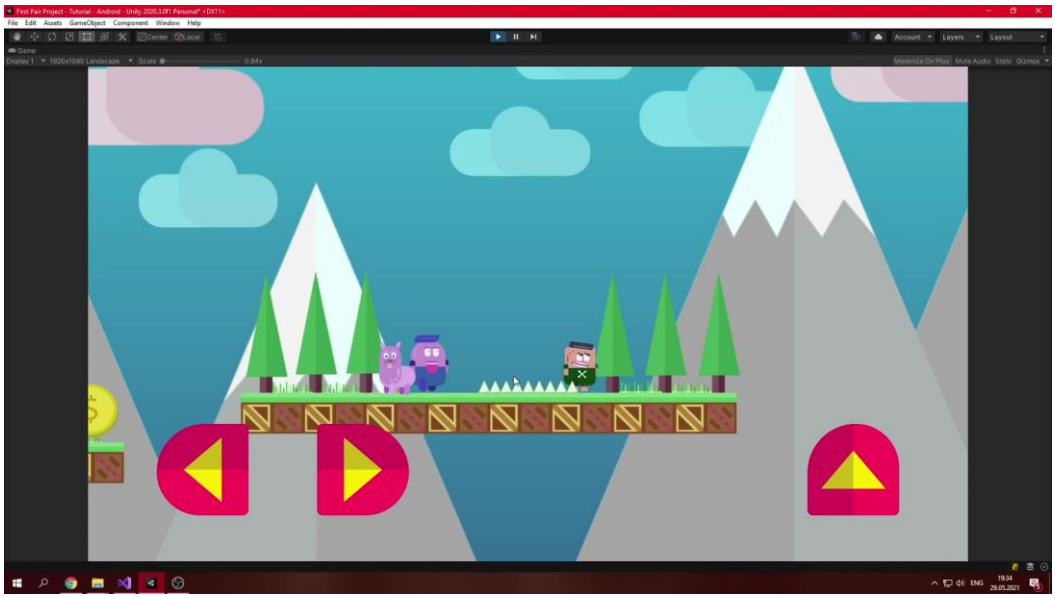


Рисунок 4.8 – Приклад STP бага під час тестування

STP Issue – баг пов’язаний з алгоритмом поведінки будь-якого NPC. Баги такого типу також можуть бути WTB.

Не правильно розташований ігровий об’єкт (World LA).

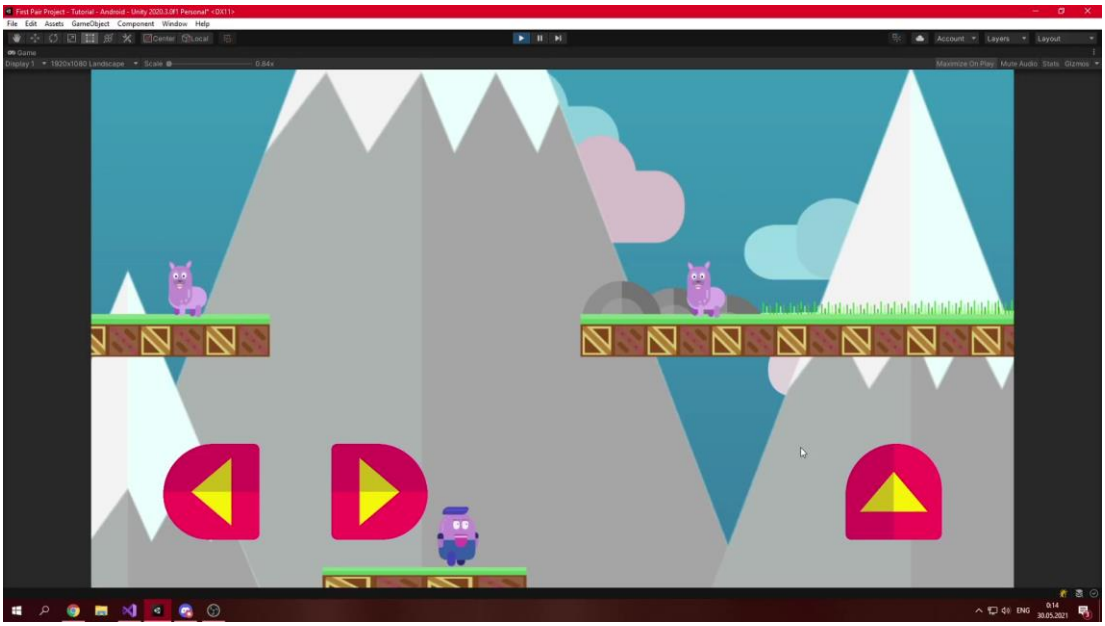


Рисунок 4.8 – Приклад World-LA бага під час тестування

World-LA (Level Art) – тип бага який пов’язан з предметами у грі. Під типів такого типу дуже багато, наприклад: floating, z-fighting, clipping, misplaced та інші.

Таблиця 4.2 – Баги знайдені під час тестування мобільного додатку «Adventus»

Summary	Issue’s Type	Severity	Priority	Attachments
Об’єкт має компонент CircleCollider2D та заважає проходженню гри.	WTB	S1	P1	Скріншот та відео
Спрайт об’єкта не співпадає з компонентом BoxCollider 2D.	Misplaced Texture	S2	P3	Скріншот та відео
Під час гри камера встановлена не правильно.	UI	S2	P2	Скріншот та відео
Відсутній інтерфейс на страрті другого рівня.	UI/WTB	S1	P1	Скріншот та відео
Велика швидкість платформи, яка рухається по вертикалі (Missions-LD).	Missions-LD	S1	P2	Скріншот та відео
Не правильно розташований ігровий об’єкт (World LA).	World-LA	S2	P1	Скріншот та відео
Порушено STP у ворожого NPC.	STP Issue	S3	P2	Скріншот та відео

Висновки до четвертого розділу

У даному розділі було проведення тестування всіх аспектів гри. Також був проведений тест інтересу на трьох мобільних девайсах. Всі тести пройшли успішно.

Дане тестування не гарантує відсутність проблем у кінцевого користувача, а лише зменшує вірогідність. Існує велика кількість девайсів з операційною системою Android, тому тести не можуть гарантувати коректність роботи програми на всіх девайсах.

5 АНАЛІЗ ТЕСТУВАННЯ ТА ВИПРАВЛЕННЯ ПОМИЛОК

5.1 Тестування відеоігор

QA означає «quality assurance», іншими словами «забезпечення якості» відеоігри. Ці слова описують мету роботи і відображають різницю між звичайним проходженням ігор та їх тестуванням. А суть полягає в пошуку багів.

Моє завдання при цьому тестуванні – зламати гру. Необхідно перебрати весь код, який працює невірно. Для цього треба проходити гру і перевіряти її на оточенням і міцність, це потрібно робити досить винахідливо. Взаємодії з предметами, персонажами відбуваються в різних порядках і різних поєднаннях. Тестувальник повинен перебрати якомога більше комбінацій, щоб перевірити, що вони працюють коректно. А для цього потрібно в тому числі нестандартно мислити. Ви повинні взаємодіяти зі світом несподіваним, навіть немислимим для розробників чином.

Таблиця 5.1 – Приклад варіантів тестування

Персонажі	Локація 1	Локація 2	Локація 3
Персонаж А	AvA	AvA	AvA
	AvB	AvB	AvB
	AvC	AvC	AvC
Персонаж В	BvA	BvA	BvA
	BvB	BvB	BvB
	BvC	BvC	BvC
Персонаж С	CvA	CvA	CvA
	CvB	CvB	CvB
	CvC	CvC	CvC

Щоб спіймати всі баги до купи, тестувальники перевіряють взагалі всі можливі комбінації ігрових елементів. Візьмемо файтинг: кожен з доступних персонажів повинен зустрітися з усіма іншими на всіх існуючих рівнях. Якщо персонажів в такій грі 10, бійка кожного з кожним виллється в 100 матчі. Однак

рівнів теж більше одного, а значить, 100 бійок повторяться на кожній з карт. Всього п'ять рівнів. Як бачите, навіть невеликі цифри і обмежений по функціоналу жанр припускають багато днів роботи тестувальника. Якщо перспектива зіграти в файтинг тисячу разів вас все ще надихає, то ви, напевно, уявляєте собі свою улюблену гру. Скажімо, Marvel vs Capcom, Dead or Alive або Mortal Kombat.

В роботі тестувальника дуже важлива уважність до деталей. Щоб можна успішно працювати в цій сфері, потрібно вміти помічати усі дрібниці. Гнучкість мислення теж буде доречною. Вигадувати все нові способи ламати гру допоможе творчий підхід.

5.2 Виправлення помилок

Баг №1. У об’єкт є компонент CircleCollider2D, він заважає проходженню гри (WTB). Щоб пройти рівень, гравець повинен перестрибнути з платформи на іншу, але об’єкт зі спрайтом «Coin» заважає йому це зробити.

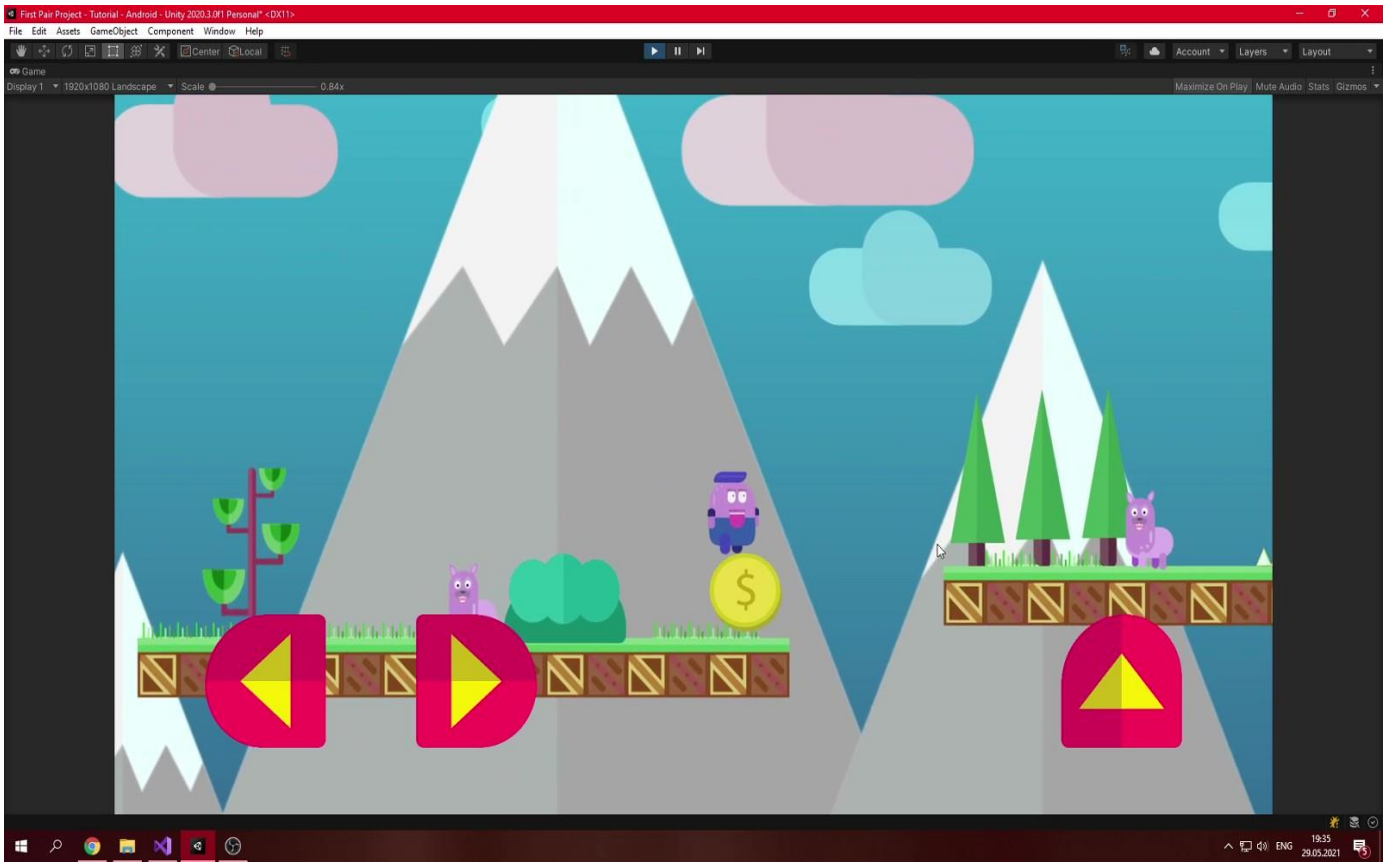


Рисунок 5.1.1 – Баг №1 до виправлення

Рішення 1 – видалити об’єкт який заважає проходженню гри.

Рішення 2 – видалити компонент CircleCollider 2D, для того щоб персонаж зміг пройти крізь об’єкт.

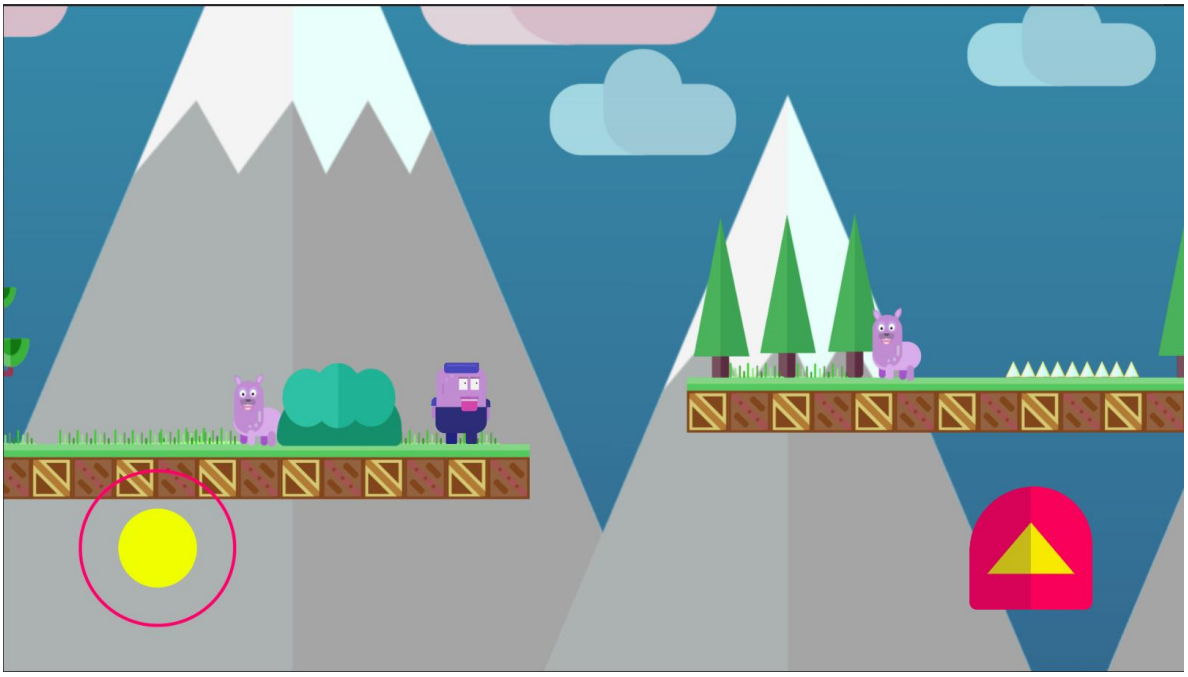


Рисунок 5.1.2 – Баг №1 виправлено

Баг 2. Порушено STP у ворожого NPC. Він наступає на шипи, які мають перешкоджати гравцю шлях (STP Issue).

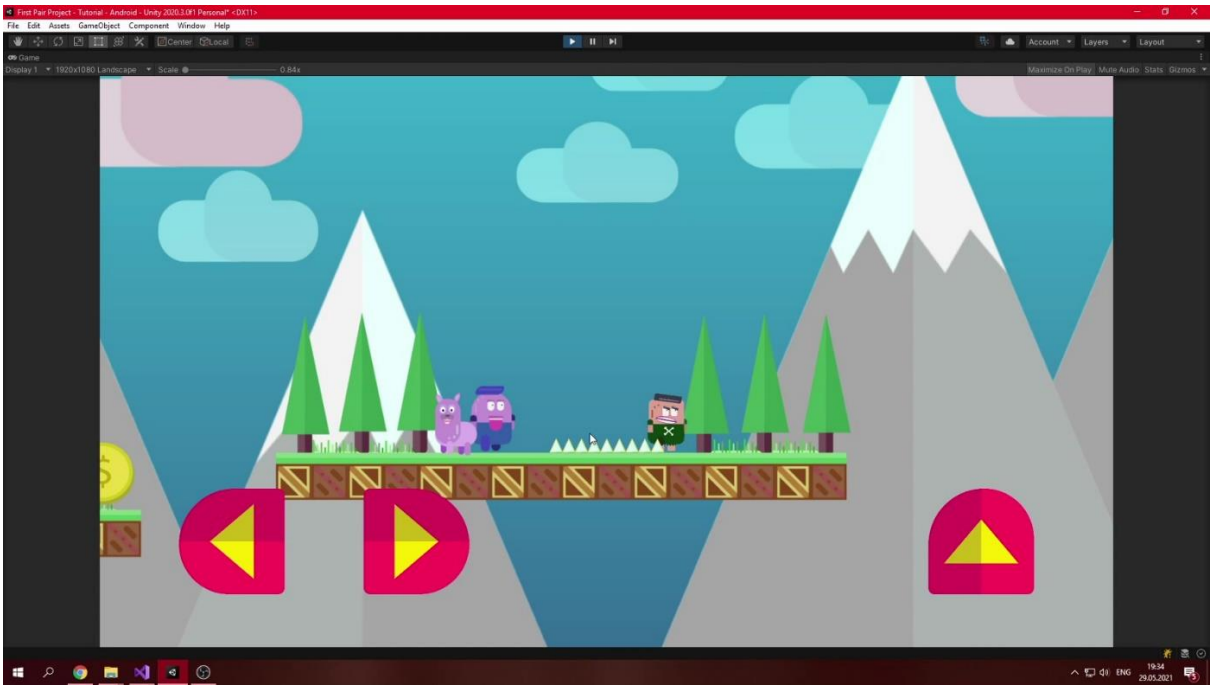


Рисунок 5.2.1 – Баг №2 до виправлення

Рішення 1 – змінити шлях у ворожого NPC таким чином, щоб він не наступав на шипи.

Рішення 2 – змінити стартову позицію у ворожого NPC.

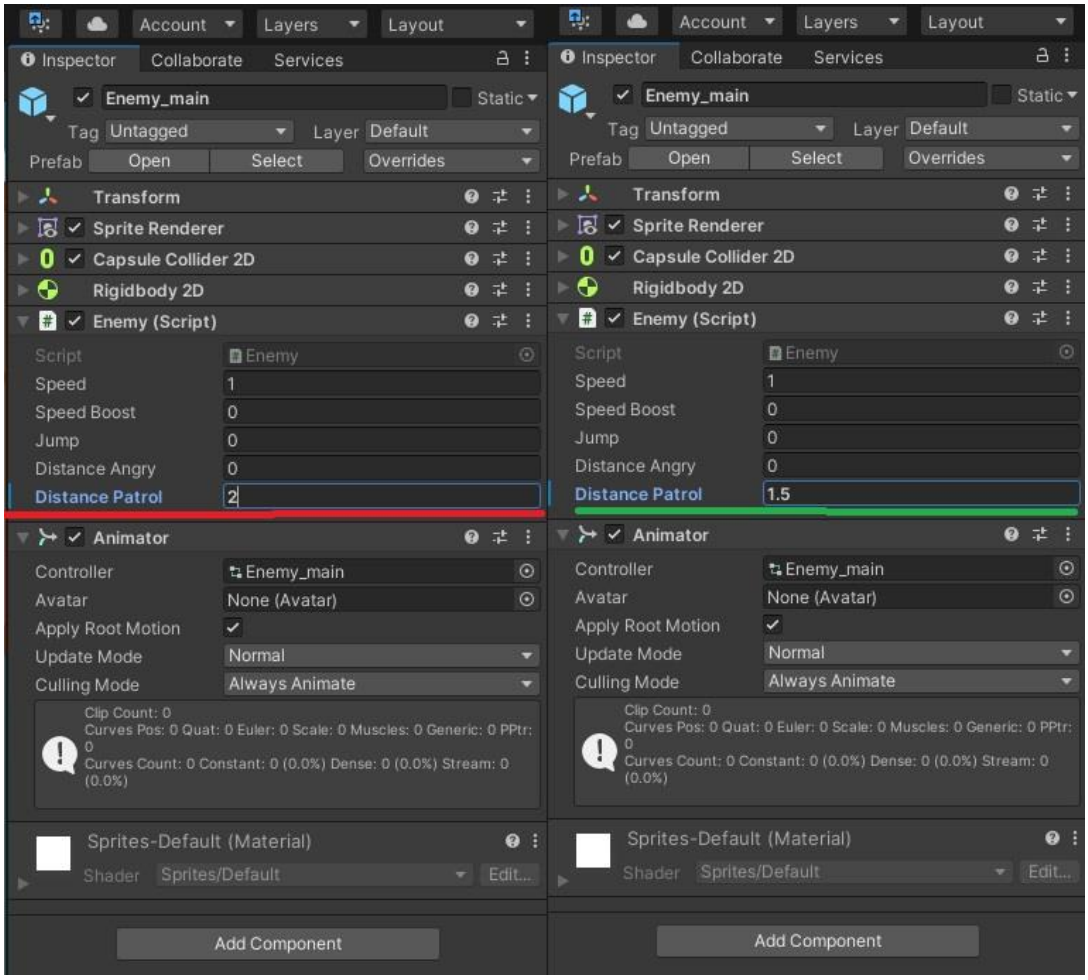


Рисунок 5.2.2 – Виправлення бага №2

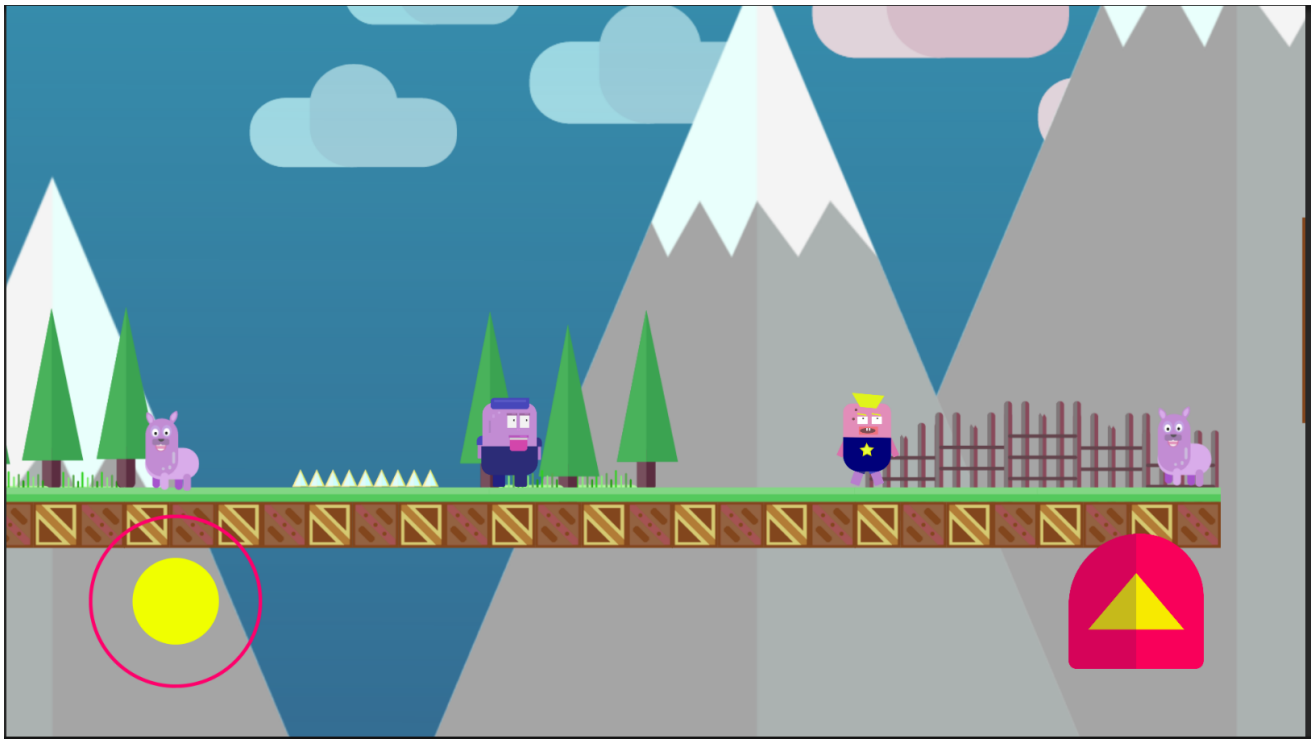


Рисунок 5.2.3 – Баг №2 виправлено

Баг 3. Неправильне розташування об’єкта.

Рішення – переставити об’єкт на сцені.

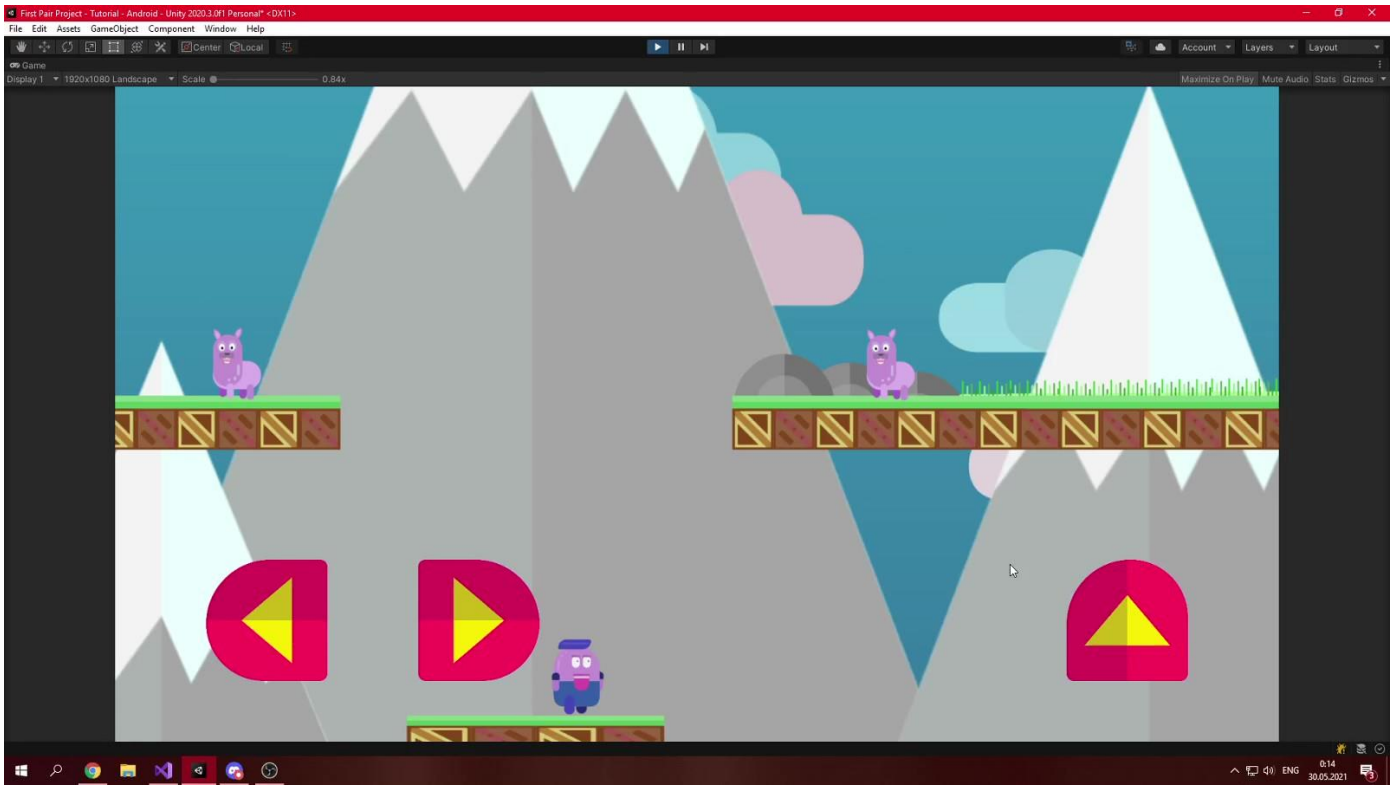


Рисунок 5.3.1 – Баг №3 до виправлення

Рисунок 5.3.2 – Баг №3 виправлено

Баг 4. Неправильно виставлені Z-координати для рук та ніг Головного Героя. Через це, на деяких ділянках гри вони відображаються перед об’єктом а не за ним.

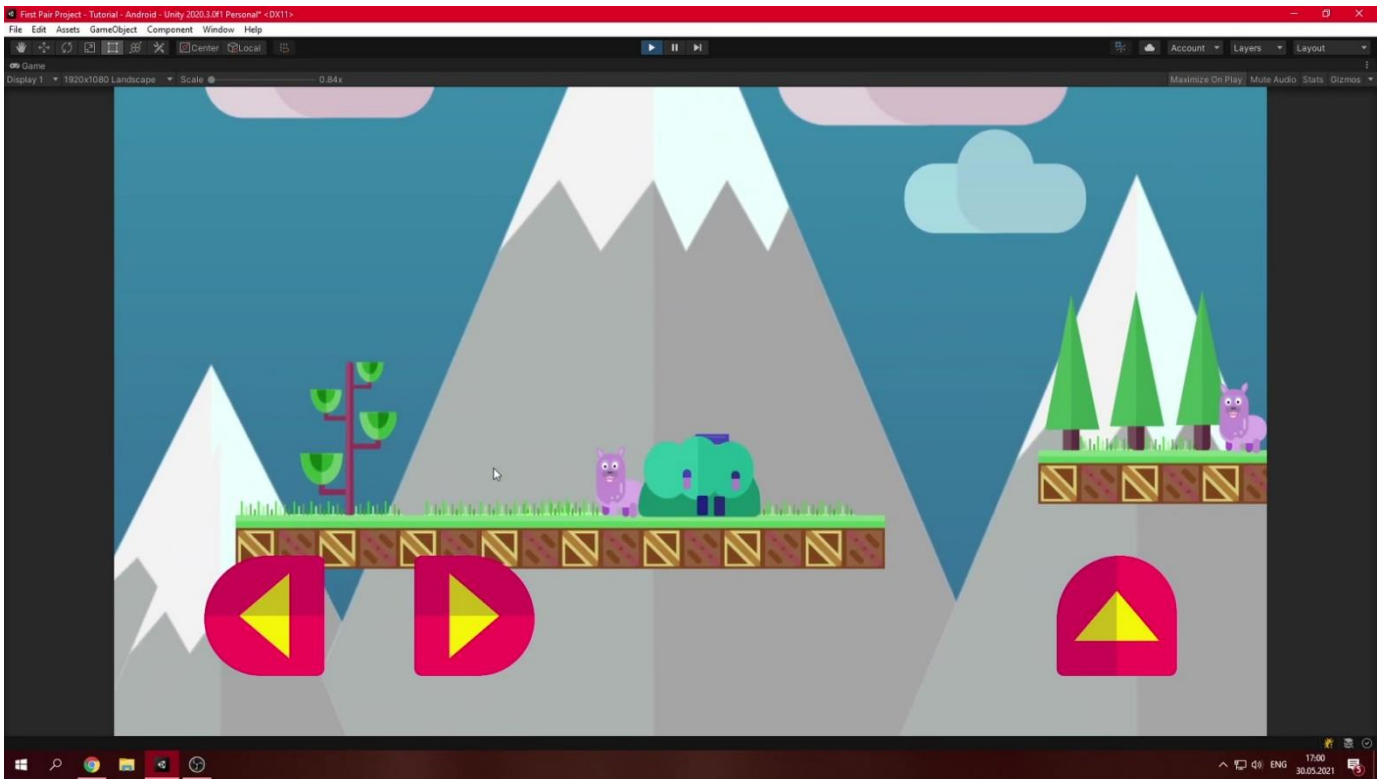


Рисунок 5.4.1 – Баг №4 до виправлення

Рішення – змінити Z-координати для рук та ніг Головного Героя.

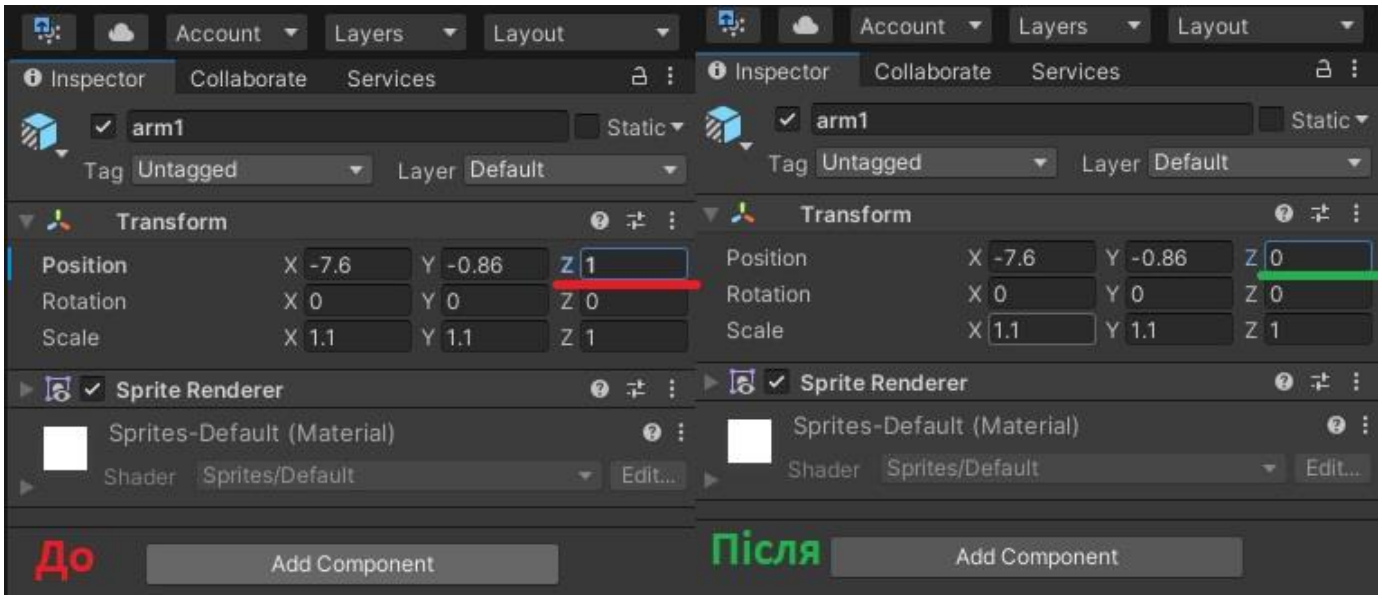


Рисунок 5.4.2 – Виправлення бага №4

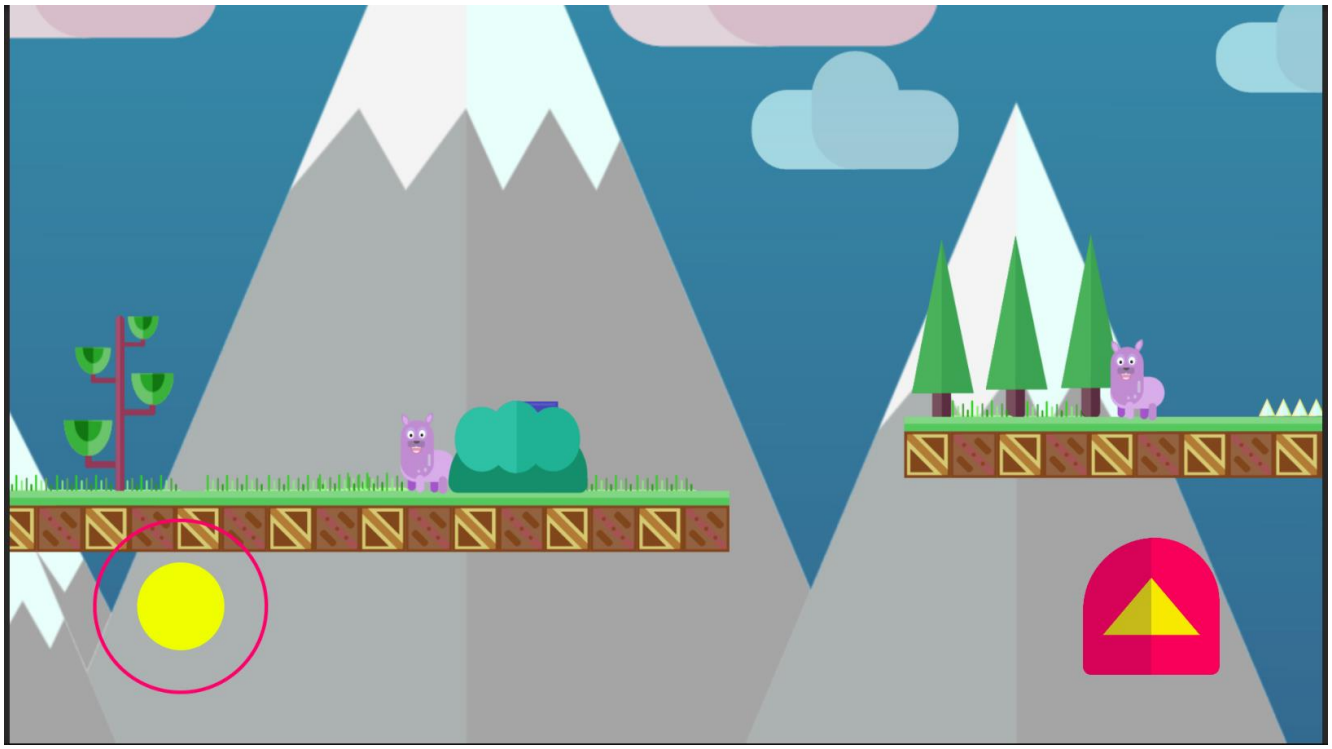


Рисунок 5.4.3 – Баг №4 виправлено

Баг 5. Встановлена дуже велика швидкість для платформи, яка має рухатися зверху до низу.

Рішення – встановити іншу швидкість у редакторі об’єкта за допомогою скрипта.

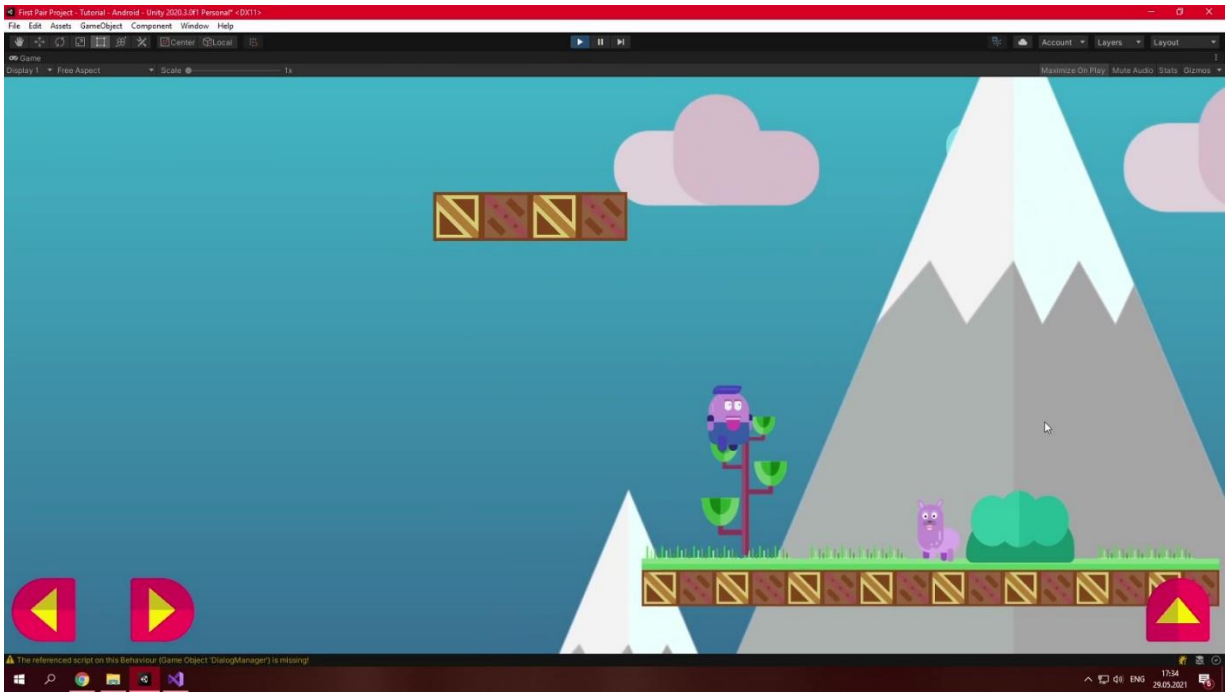


Рисунок 5.5.1 – Баг №5 до виправлення

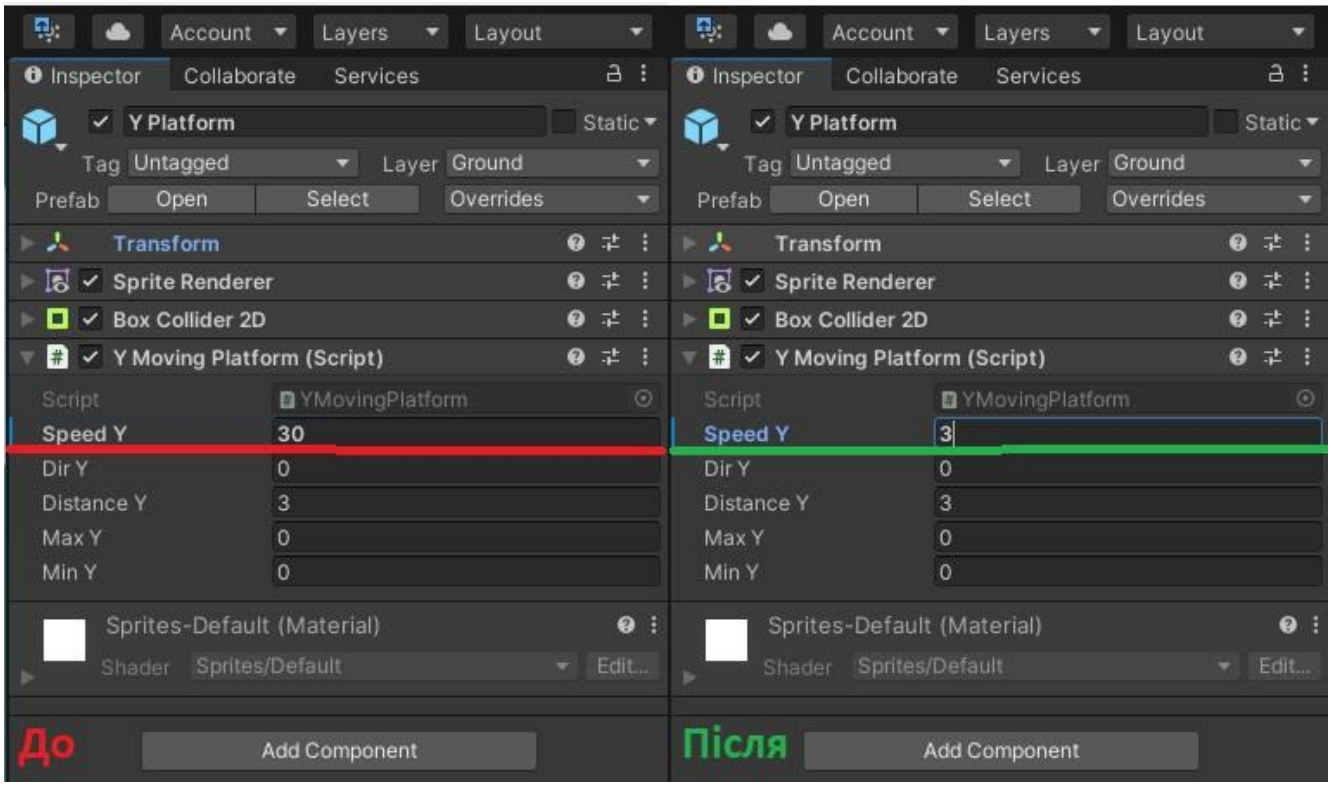


Рисунок 5.5.2 – Баг №5 виправлено

Баг 6. У настройках камери пункт «Scale» встановлено неправильно.

Рішення – зміни настройки камери.

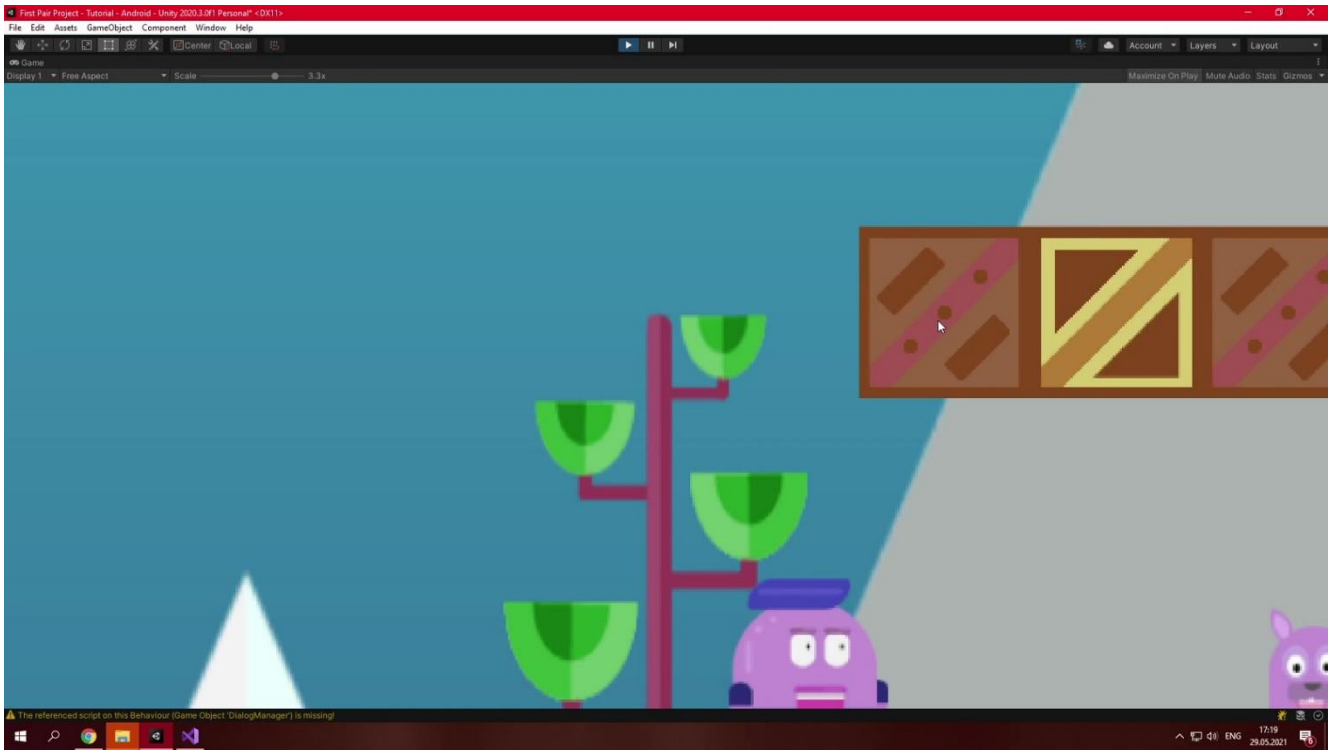


Рисунок 5.6.1 – Баг №6 до виправлення

Рисунок 5.6.2 – Баг №6 виправлено

Баг 7. У об’єкта відсутний компонент BoxCollider2D. Тому персонаж може проходити крізь об’єкт, та не може з ним контактувати.

Рішення – додати компонент «BoxCollider2D» у об’єкт.

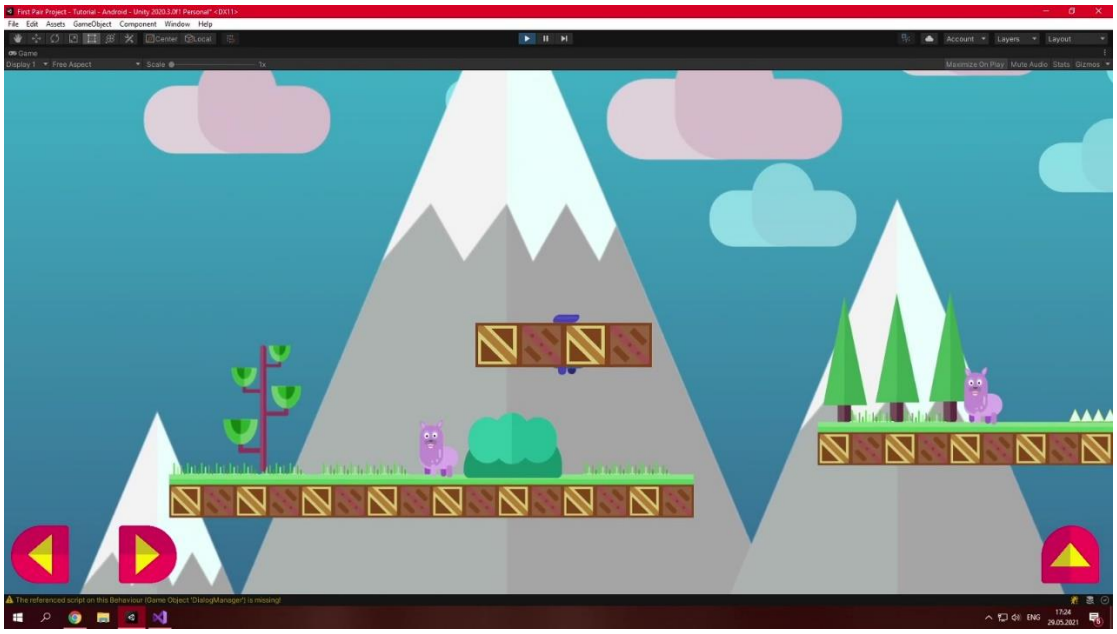


Рисунок 5.7.1 – Баг №7 до виправлення

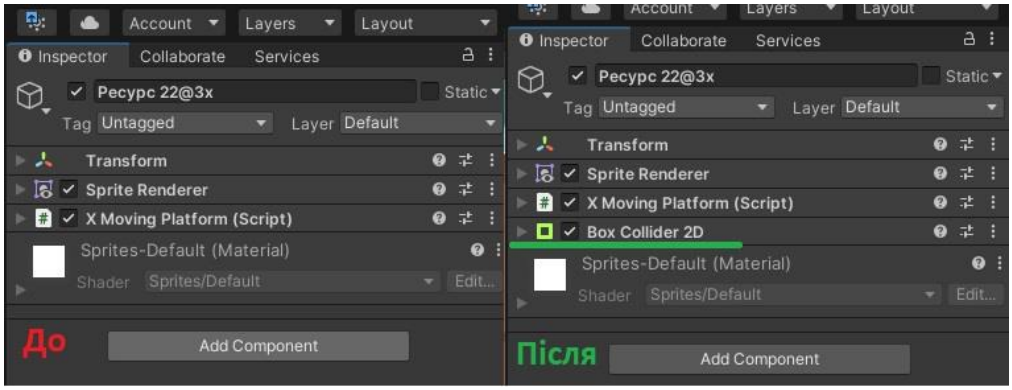


Рисунок 5.7.2 – Виправлення бага №7

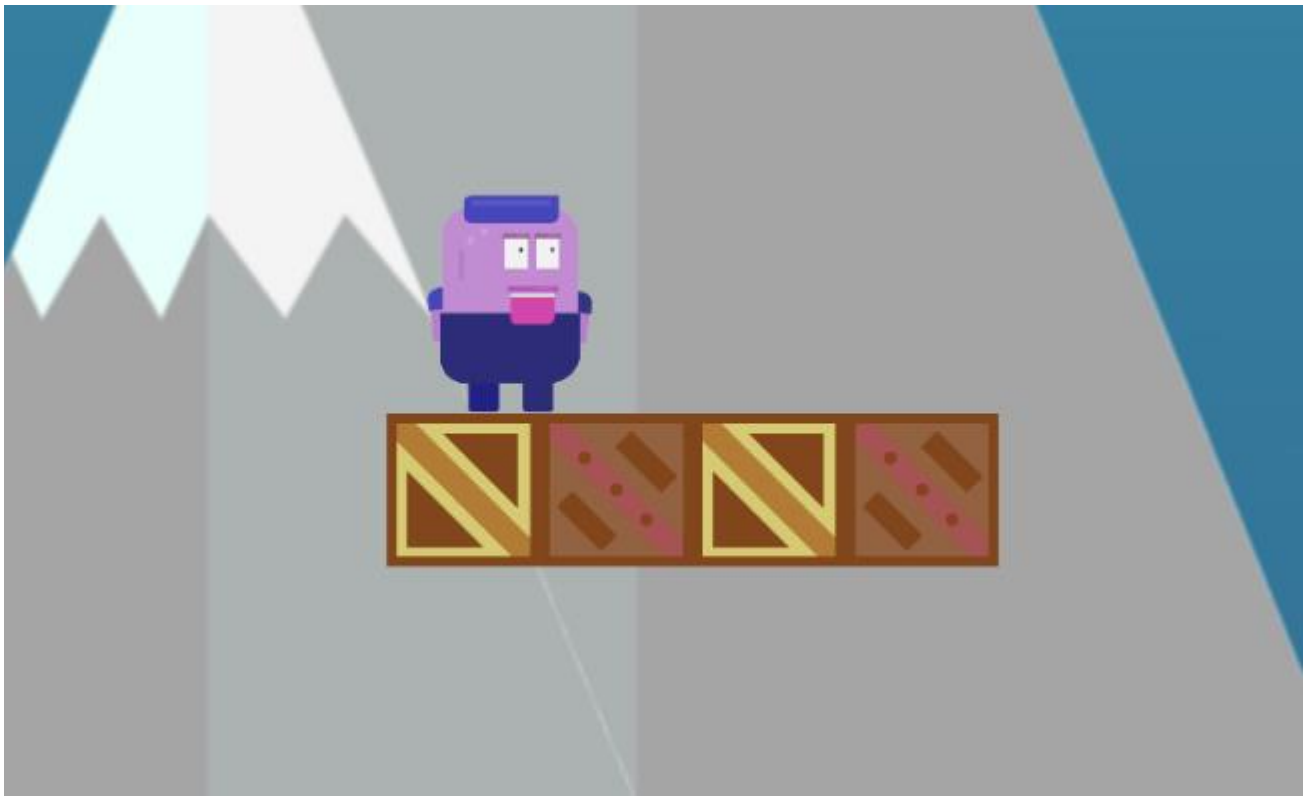


Рисунок 5.7.3 – Баг №7 виправлено

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

1. Проведено аналіз існуючих ігрових Android-додатків.
2. Розроблено нову гру в жанрі платформер під назвою «Adventus». Створено групу скриптів, сцени та зручний користувальницький інтерфейс.
3. Обрані інструменти розробки - Unity та Visual Studio, які мають деякі переваги, що забезпечують швидку та ефективну розробку Android-додатку з ігровим акцентом.
4. Практичне значення роботи складається в тому, що результати розробки може бути завантажено в магазин додатків для подальшого встановлення на різні мобільні пристрої з операційною системою Android.
5. Сучасні технології надають можливості для розвитку і подальшого вдосконалення розробленного ігрового Android-додатку, що говорить про актуальність та перспективність даної роботи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Unitiy Learning [Електронний ресурс] – Режим доступу: https://unity.com/learn

2. Microsoft Build [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/ru-ru/documentation/

3. Рейнбоу В. «Комп'ютерні ігри». Енциклопедія – С .: «Пітер», 2005. – 732с.

4. Миколаєва О. «Епідемія XXI століття: телебачення, інтернет і коп'ютерні ігри». – Ростов н/Д: Фелікс, 2018. – 254с.

5. Роллінгз, Ендрю. «Проектування та архітектура ігор»: пров. з англ. / Е. Роллінгз, Д. Морріс. – М.: Вільямс, 2016. – 1040с.

6. Дашко Ю. В., Заїка А. А. «Основи розробки комп'ютерних ігор» – М.: «Форум», 2009. – 350с.

7. Шреєр Дж. «Кров піт і пікселі. Зворотний бік індустрії відеоігор» – Р.: «Форс», 2019. – 368с.

8. Дороті Грем «Foundations of Software Testing: ISTQB Certification» – Р.: інш. 2016. – 208с.

9. Шевченко С. В. Гра як інструмент формування сприятливого соціально психологічного клімату та джерело розвитку усіх учасників освітнього процесу. VIII Всеукраїнська науково-практична конференція студентів, аспірантів та молодих учених «Гуманітарний і інноваційний ракурс професійної майстерності: Пошуки молодих вчених»: матеріали конф., 18 листопада 2022 р.: тези доп. – Одеса: МГУ, 2022. – С. 357-361. DOI https://doi.org/10.36059/978-966-397-266-4/105

10. Завербний А.С., Хома Т.В., Роль цифровізації, фінтеху, ІТ-інструментів у гармонійному розвитку економіки України // Сучасні тенденції розвитку фінансових та інноваційно-інвестиційних процесів в Україні. Матеріали VI Міжнародної науково-практичної конференції 2-3 березня 2023 року: збірник наукових праць [Електронний ресурс]. – Вінниця: ВНТУ, 2023. – С. 359-362.

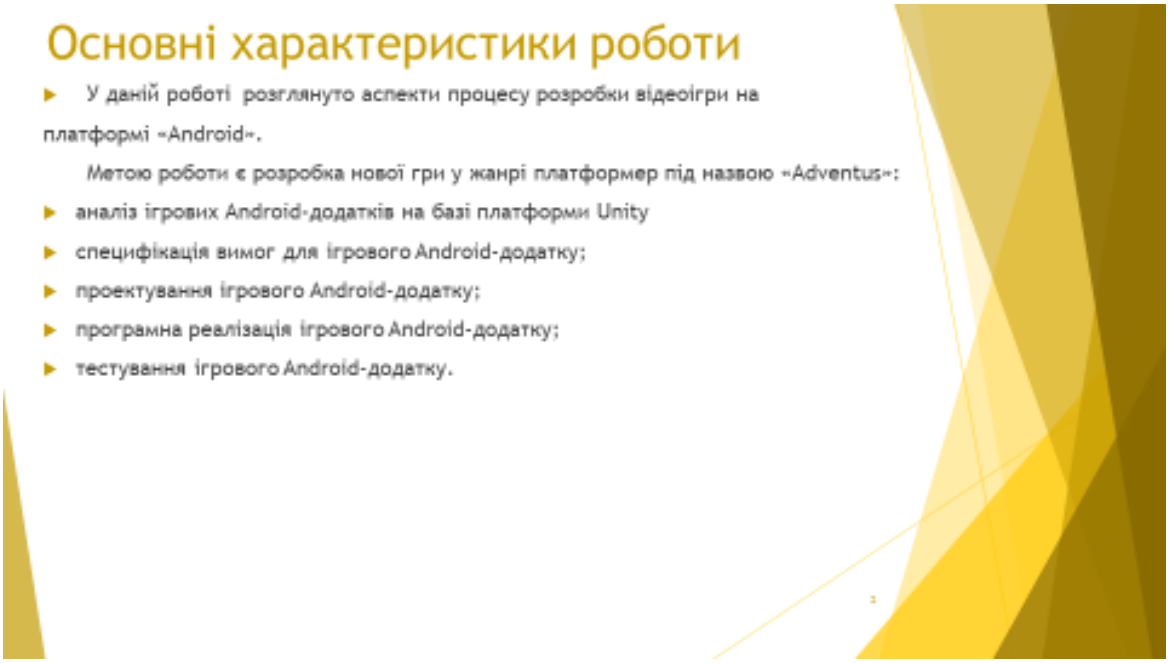
11. Сайт з інформацією про створення гри Unity з допомогою Visual Studio [Електронний ресурс] - URL: https://visualstudio.microsoft.com/ru/vs/unity-tools/

12. How to Download and Install Visual Studio for C# [Електронний ресурс] – Режим доступу: https://www.guru99.com/download-install-visual-studio.html
13. Гречко А. В., Захаров Н. В., Фалько М. О. Аналіз динаміки розвитку ринку відеоігор, джерел його фінансування та особливостей монетизації продукції в даній сфері. Ефективна економіка. 2021. № 5. – URL: http://www.economy.nayka.com.ua/?op=1&z=8871 DOI: 10.32702/2307-2105-2021.5.2

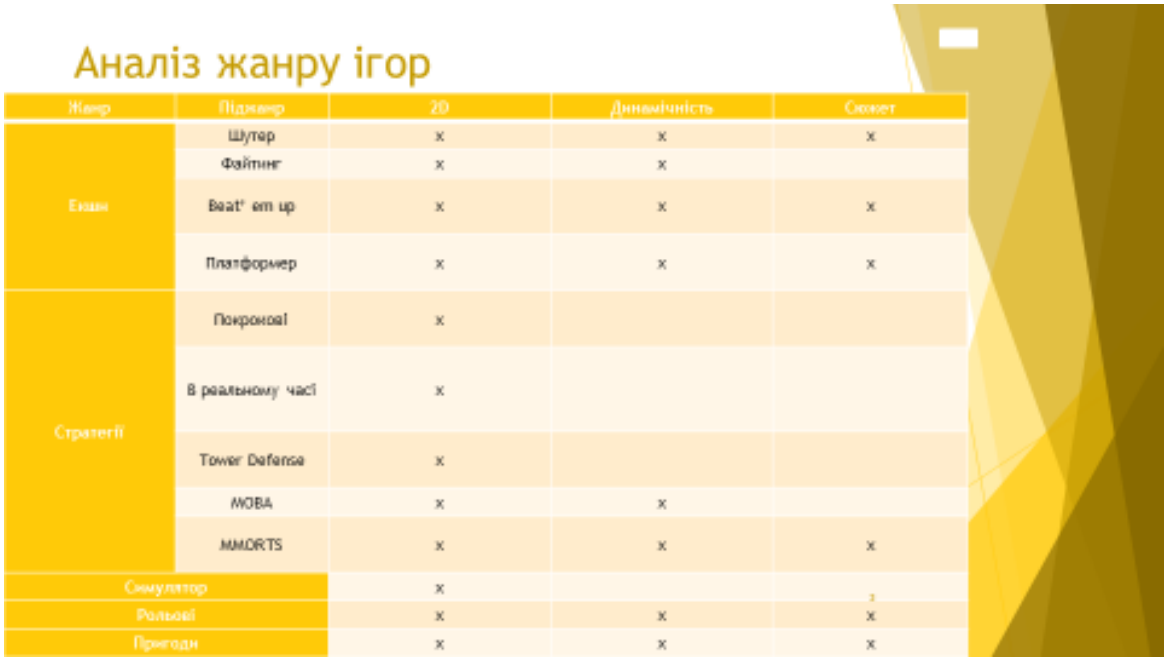
ДОДАТОК А

Перелік копій демонстраційного матеріалу

Слайд 1 – Тема магістерської роботи

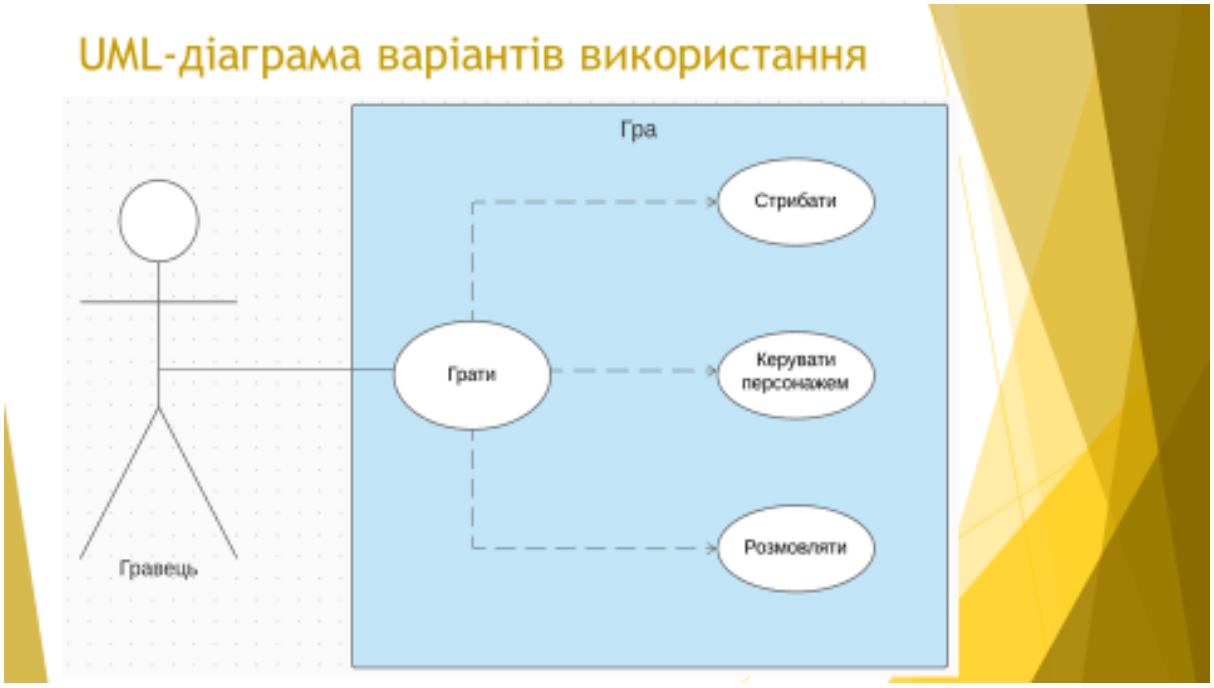


Слайд 2 – Основні характеристики роботи



Слайд 3 – Аналіз жанру ігор

Слайд 4 – Аналіз студій розробників ігор



Слайд 5 – UML-діаграма варіантів використання

Види тестування

Функціональні	Нефункціональні		Пов'язані зі змінами
Функціональне тестування	Тестування установки		Димове тестування
Тестування інтерфейсу	Тестування зручності		Регресійне тестування
Тестування безпеки	Тестування на відмову і відновлення		Повторне тестування
Тестування взаємодії	Конфігураційне тестування		Тестування збірки
-	Всі види тестування продуктивності		Санітарне тестування
-	Performance Testing	Stress Testing	-
-	Stability Testing	Volume Testing	-

ДОДАТОК Б

Розроблені скрипти для мобільного додатку

Скрипт YMovingPlatform:

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

public class YMovingPlatform : MonoBehaviour
{

    public float speedY;
    public float dirY;
    public float distanceY;

    public float maxY;
    public float minY;

    bool moveRight = true;

    void Start()
    {
        minY = transform.position.y - distanceY;
        maxY = transform.position.y + distanceY;
    }
    void Update()
    {
        transform.Translate(transform.up * speedY * Time.deltaTime);
        if (transform.position.y > maxY)
        {
            speedY = -speedY;
        }
        else if (transform.position.y < minY)
        {
            speedY = -speedY;
        }
    }
       }
```

Скрипт XMovingPlatform:

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

public class XMovingPlatform : MonoBehaviour
{
    public float dirX;
    public float speedX;
    public float distanceX;
    public float speedY;
    public float dirY;
    public float distanceY;

    public float maxX;
    public float minX;

    bool moveRight = true;

    void Start()
    {
        minX = transform.position.x - distanceX;
        maxX = transform.position.x + distanceX;
    }
    void Update()
```

```
    {
        transform.Translate(transform.right * speedX* Time.deltaTime);
        if (transform.position.x > maxX)
        {
            speedX = -speedX;
        }
        else if (transform.position.x < minX)
        {
            speedX = -speedX;
        }
    }
}
```

Скрипт FallingPlatform:

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

public class FallingPlatform : MonoBehaviour
{
    Rigidbody2D rb;

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

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.name.Equals("Character"))
        {
            Invoke("FallPlatform", 1f);
            Destroy(gameObject,2f);
        }
    }

    void FallPlatform()
    {
        rb.isKinematic = false;
    }
}
```

Скрипт DeadFromFalling:

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

public class DeadFromFalling : MonoBehaviour
{
    public GameObject respawn;
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            other.transform.position = respawn.transform.position;
        }
    }
}
```

Скрипт NextLevel:

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

public class NextLevel : MonoBehaviour
{
```

```
    public int level;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            SceneManager.LoadScene(level);
        }
    }
}
```

Скрипт Dialog:

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

public class Dialog : MonoBehaviour
{
    public GameObject text;

    void Start()
    {
        text.SetActive(false);
    }
    private void OnTriggerEnter2D(Collider2D col)
    {
        if (col.tag == "Player")
        {
            text.SetActive(true);
        }
    }
    private void OnTriggerExit2D(Collider2D col)
    {
        text.SetActive(false);
    }
}
```

Скрипт CameraControler:

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

public class CameraControler : MonoBehaviour
{
    public float dumping = 1.5f;
    public Vector2 offset = new Vector2(2f, 1f);
    public bool isleft;
    private Transform player;
    private int lastX;

    void Start()
    {
        offset = new Vector2(Mathf.Abs(offset.x), offset.y);
        FindPlayer(isleft);
    }

    public void FindPlayer(bool playerIsLeft)
    {
        player = GameObject.FindGameObjectWithTag("Player").transform;
        lastX = Mathf.RoundToInt(player.position.x);
        if(playerIsLeft)
        {
            transform.position = new Vector3(player.position.x - offset.x, player.position.y +
offset.y,transform.position.z);
        }
        else
        {
```

```
            transform.position = new Vector3(player.position.x + offset.x, player.position.y +
offset.y, transform.position.z);
        }
    }

    private void Update()
    {
        if(player)
        {
            int currentX = Mathf.RoundToInt(player.position.x);
            if(currentX > lastX)
            {
                isleft = false;
            }
            else if(currentX < lastX)
            {
                isleft = true;
            }
            lastX = Mathf.RoundToInt(player.position.x);
            Vector3 target;
            if (isleft)
            {
                target = new Vector3(player.position.x - offset.x, player.position.y +
offset.y, transform.position.z);
            }
            else
            {
                target = new Vector3(player.position.x + offset.x, player.position.y +
offset.y, transform.position.z);
            }

            Vector3 currentPosition = Vector3.Lerp(transform.position, target, dumping *
Time.deltaTime);
            transform.position = currentPosition;
        }
    }
}
```

Скрипт CharacterControl:

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

public class CharacterControl : MonoBehaviour
{
    public float speed;
    public float jumpForce;
    private float moveInput;
    private bool facingRight = true;
    private Rigidbody2D rb;

    private bool isGrounded;
    public Transform feetPos;
    public float checkRadius;
    public LayerMask whatIsGround;

    private Animator anim;

    public Joystick js;

    public float extrajumps = 2;

    private void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
    }
```

```
private void FixedUpdate()
{
    moveInput = js.Horizontal;
    rb.velocity = new Vector2(moveInput * speed, rb.velocity.y);

    if (facingRight == false && moveInput > 0)
    {
        Flip();
    }
    else if (facingRight == true && moveInput < 0)
    {
        Flip();
    }

    if (moveInput==0)
    {
        anim.SetBool("isRun", false);
    }
    else
    {
        anim.SetBool("isRun", true);
    }
}

private void Update()
{
    if (isGrounded == true)
    {
        extrajumps = 1;
    }


    isGrounded = Physics2D.OverlapCircle(feetPos.position, checkRadius, whatIsGround);

    if (Input.GetKeyDown(KeyCode.Space) && extrajumps > 0)
    {
        rb.velocity = Vector2.up * jumpForce;
        anim.SetTrigger("takeOff");
        extrajumps--;
    }

    if (isGrounded == true)
    {
        anim.SetBool("isJump", false);
    }
    else
    {
        anim.SetBool("isJump", true);
    }
}

public void OnJumpButtonDown()
{
    if (isGrounded == true)
    {
        extrajumps = 1;
    }

    if (extrajumps > 0)
    {
        rb.velocity = Vector2.up * jumpForce;
        anim.SetTrigger("takeOff");
        extrajumps--;
    }
}
```

```
    void Flip()
    {
        facingRight = !facingRight;
        Vector3 Scaler = transform.localScale;
        Scaler.x *= 1;
        transform.localScale = Scaler;
        if (moveInput < 0)
        {
            transform.eulerAngles = new Vector3(0, 180, 0);
        }
        else if (moveInput > 0)
        {
            transform.eulerAngles = new Vector3(0, 0, 0);
        }
    }
}
```

Скрипт Enemy:

```
using UnityEngine;

public class Enemy : MonoBehaviour {
    [SerializeField] private float speed;
    [SerializeField] private float speedBoost;
    [SerializeField] private float jump;
    [SerializeField] private float distanceAngry;
    [SerializeField] private float distancePatrol;
    private float minDistance;
    private float maxDistance;
    private Animator anim;
    private Rigidbody2D rb;
    private bool patrol = true;
    private Transform player;

    private void Start () {
        player = GameObject.Find ("Character").transform;
        anim = GetComponent<Animator> ();
        rb = GetComponent<Rigidbody2D> ();
        minDistance = transform.position.x - distancePatrol;
        maxDistance = transform.position.x + distancePatrol;
    }
    private void Update () {
        if (patrol == true)
            Patrol ();
        else
            Angry ();
        if (Vector2.Distance (transform.position, player.position) < distanceAngry) {
            Mathf.Abs (speed);
            patrol = false;
        }
    }
    private void Patrol () {
        transform.Translate (transform.right * speed * Time.deltaTime);
        if (transform.position.x > maxDistance) {
            speed = -speed;
            transform.rotation = Quaternion.Euler (0, 0, 0);
        } else if (transform.position.x < minDistance) {
            speed = -speed;
            transform.rotation = Quaternion.Euler (0, 180, 0);
        }
    }
    private void Angry () {
        if (patrol == false) {
            Vector2 moveVector = Vector2.MoveTowards (transform.position, player.position,
speedBoost * speed * Time.deltaTime);
            transform.position = new Vector2 (moveVector.x, transform.position.y);
            if (transform.position.x > player.position.x) {
                transform.rotation = Quaternion.Euler (0, 0, 0);
```

```
            } else if (transform.position.x < player.position.x) {
                transform.rotation = Quaternion.Euler (0, 180, 0);
            }
        }
    }
    private void Jump () {
        rb.AddForce (transform.up * jump, ForceMode2D.Impulse);
        anim.SetTrigger ("Jump");
    }
    private void OnTriggerEnter2D (Collider2D other) {
        if (other.tag == "ground")
            Jump ();
    }
}
```

Скрипт MainMenu:

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

public class MainMenu : MonoBehaviour
{
    public void NewGame()
    {
        SceneManager.LoadScene(1);
    }

    public void ChooseLevel()
    {
        SceneManager.LoadScene(4);
    }

    public void Options()
    {
        SceneManager.LoadScene(5);
    }

    public void Credits()
    {
        SceneManager.LoadScene(6);
    }

    public void ExitGame()
    {
        Application.Quit();
    }
}
```

Скрипт BackToMeny:

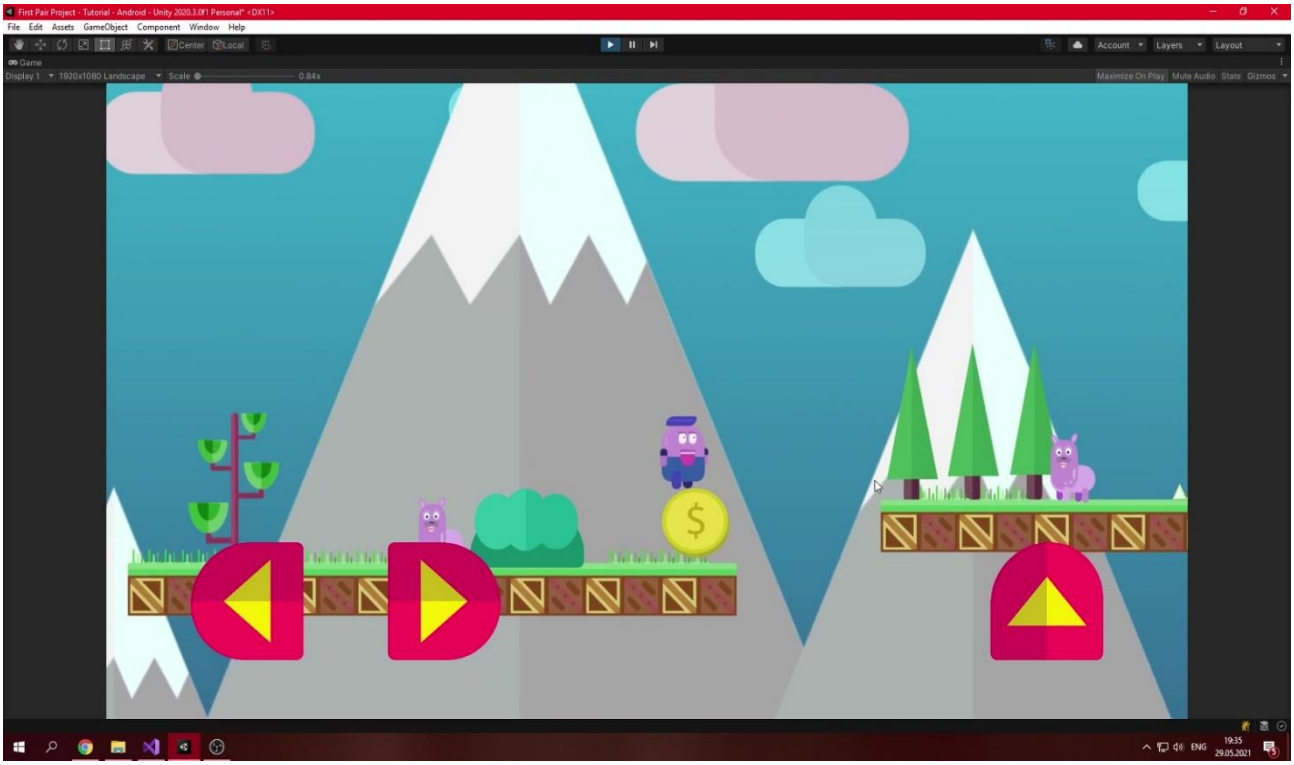
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;


public class BackToMeny : MonoBehaviour
{
    public void Back()
    {
        SceneManager.LoadScene(0);
    }

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

Скрипт ChooseLevel:

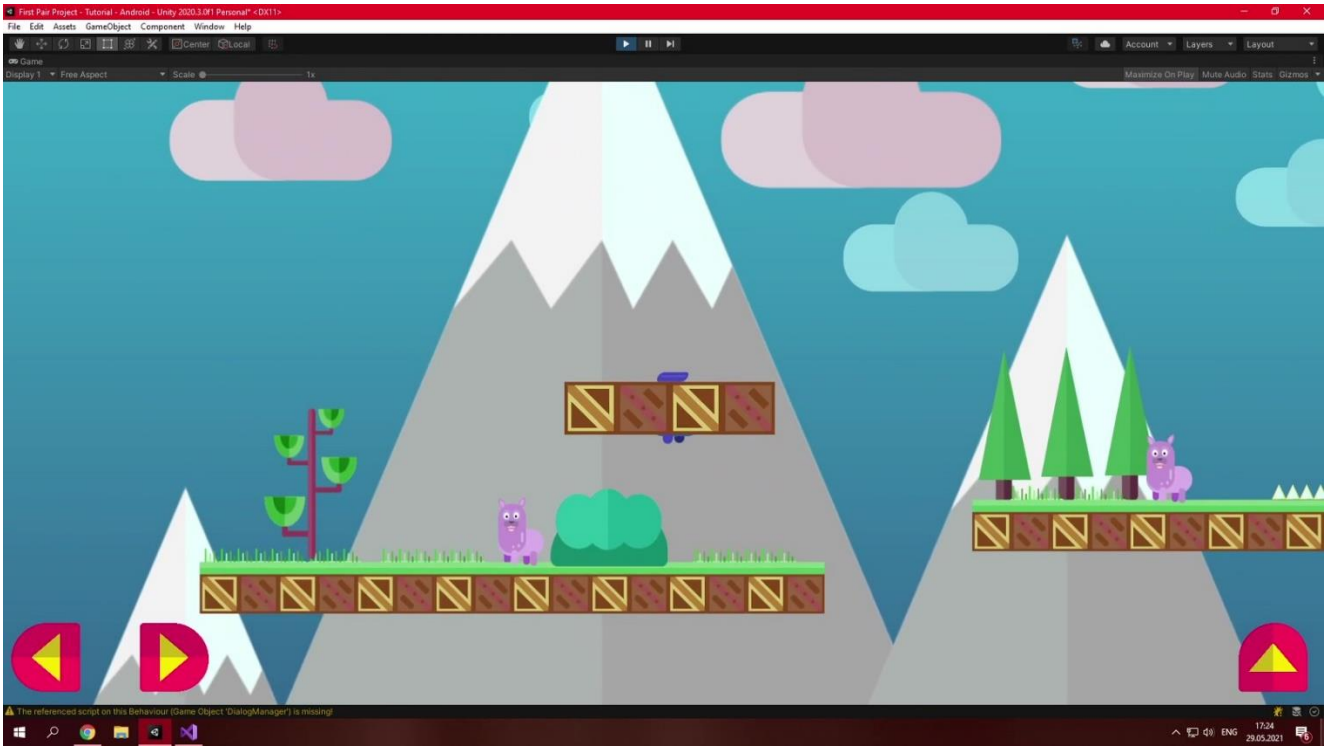
```
public class ChooseLevel : MonoBehaviour
{
    public void First()
    {
        SceneManager.LoadScene(1);
    }
    public void Second()
    {
        SceneManager.LoadScene(2);
    }
    public void Third()
    {
        SceneManager.LoadScene(3);
    }
}
```

ДОДАТОК В
Виявлені баги

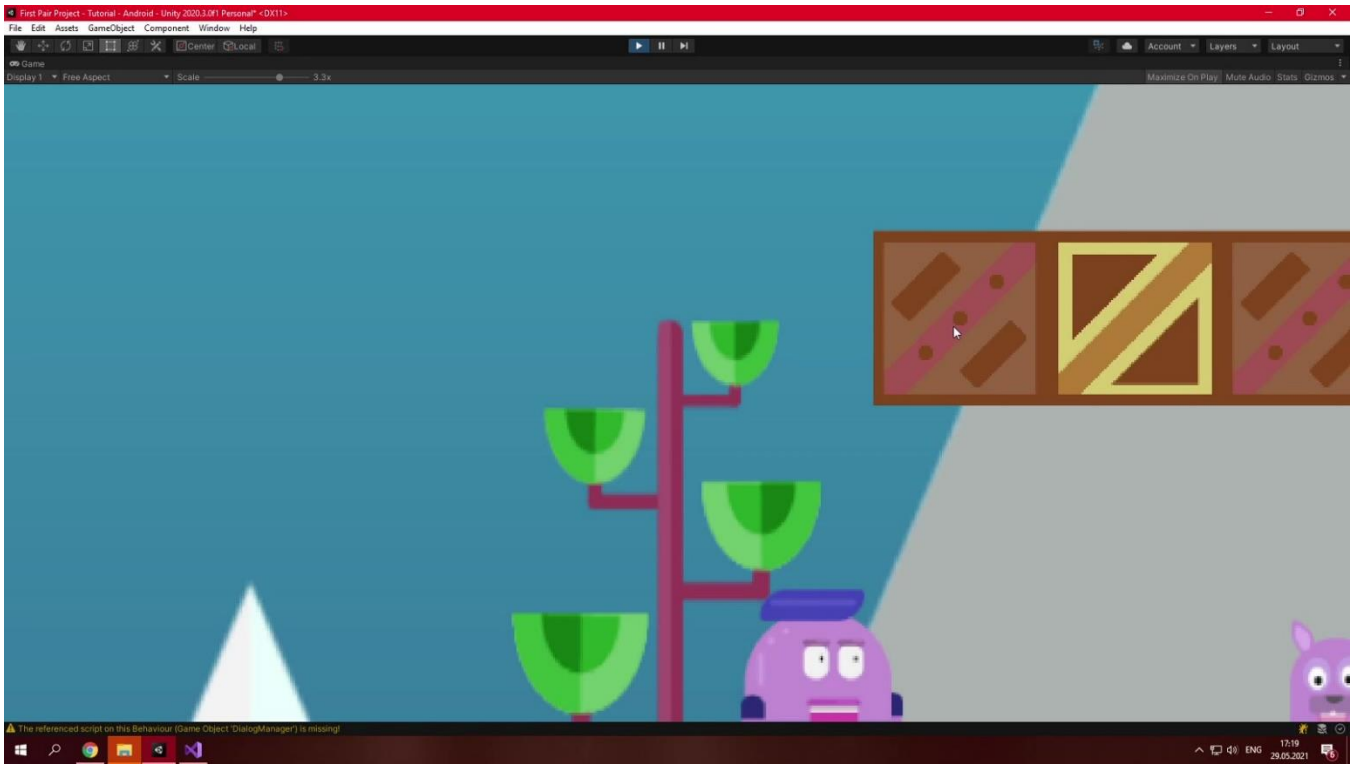
Приклад WTB бага під час тестування



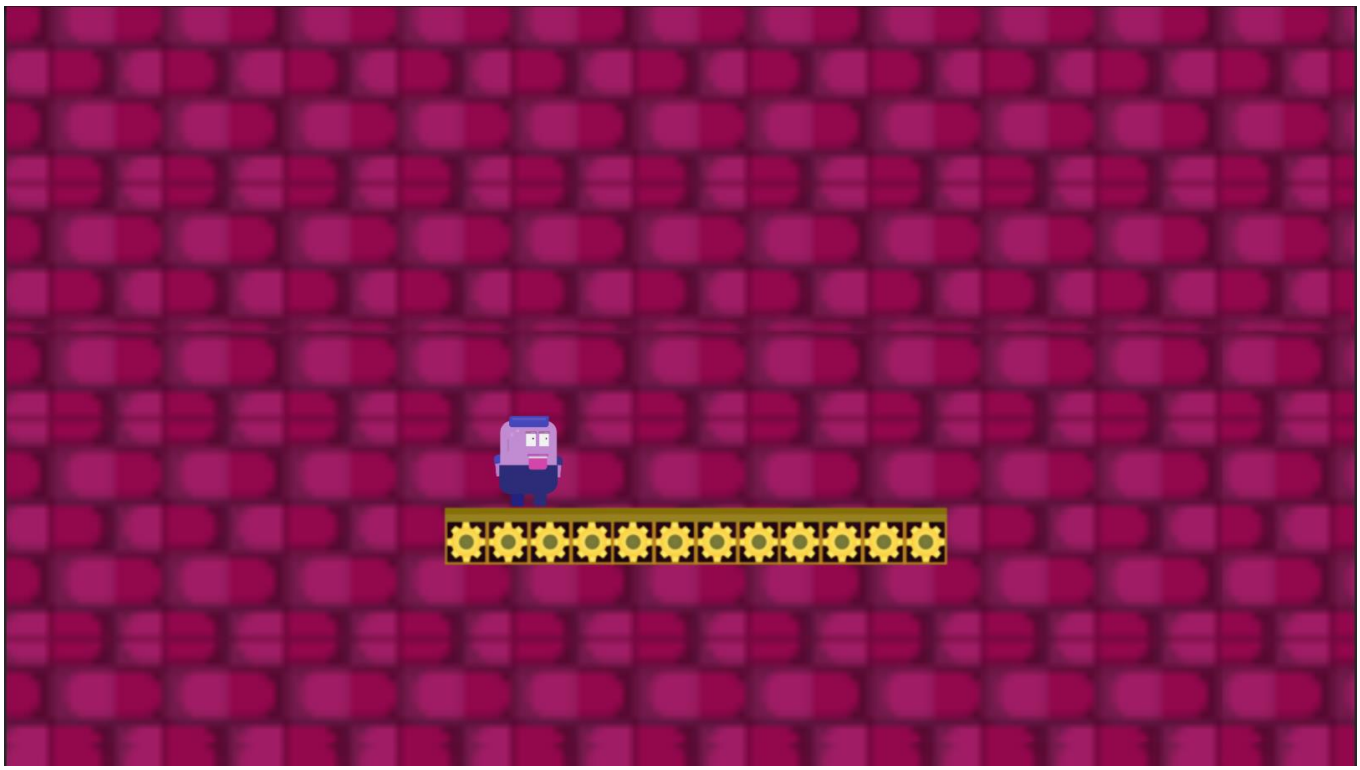
Приклад Misplaced Texture бага під час тестування



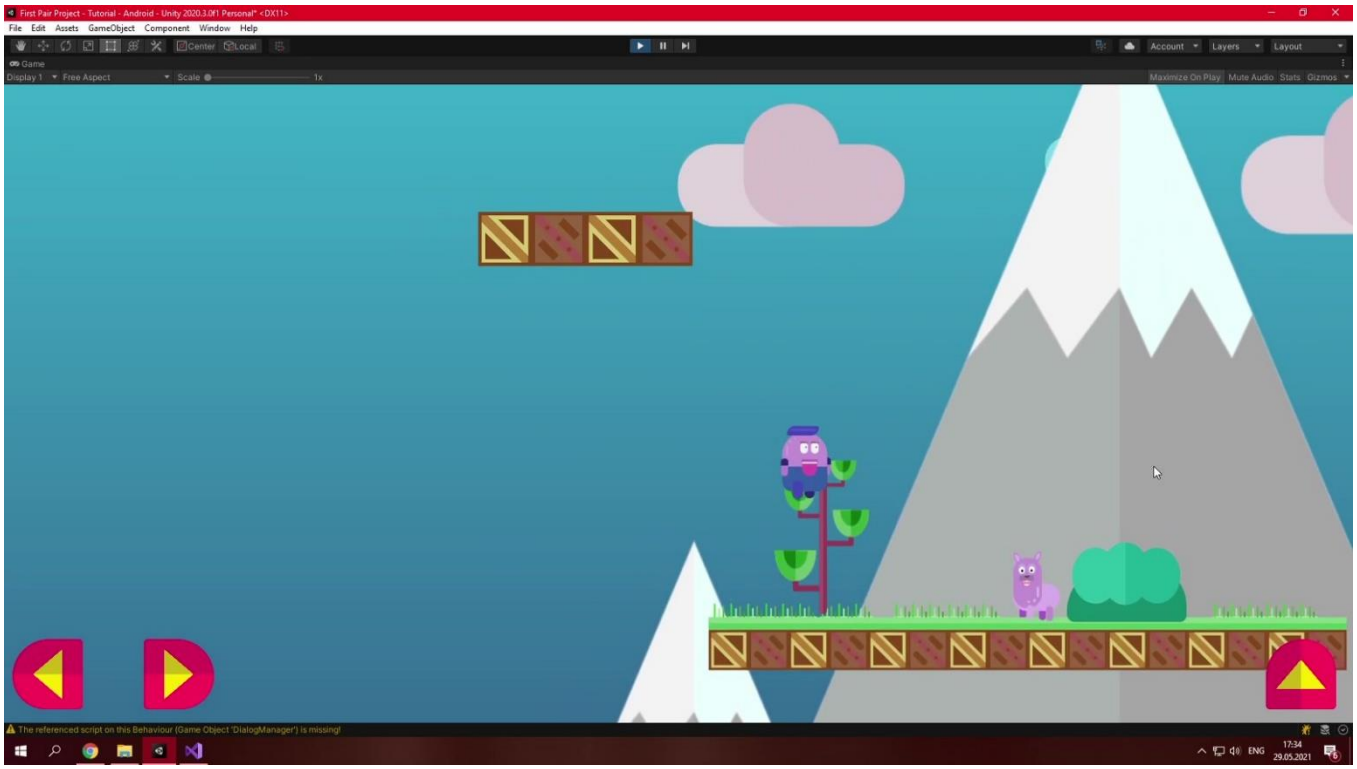
Приклад UI бага під час тестування



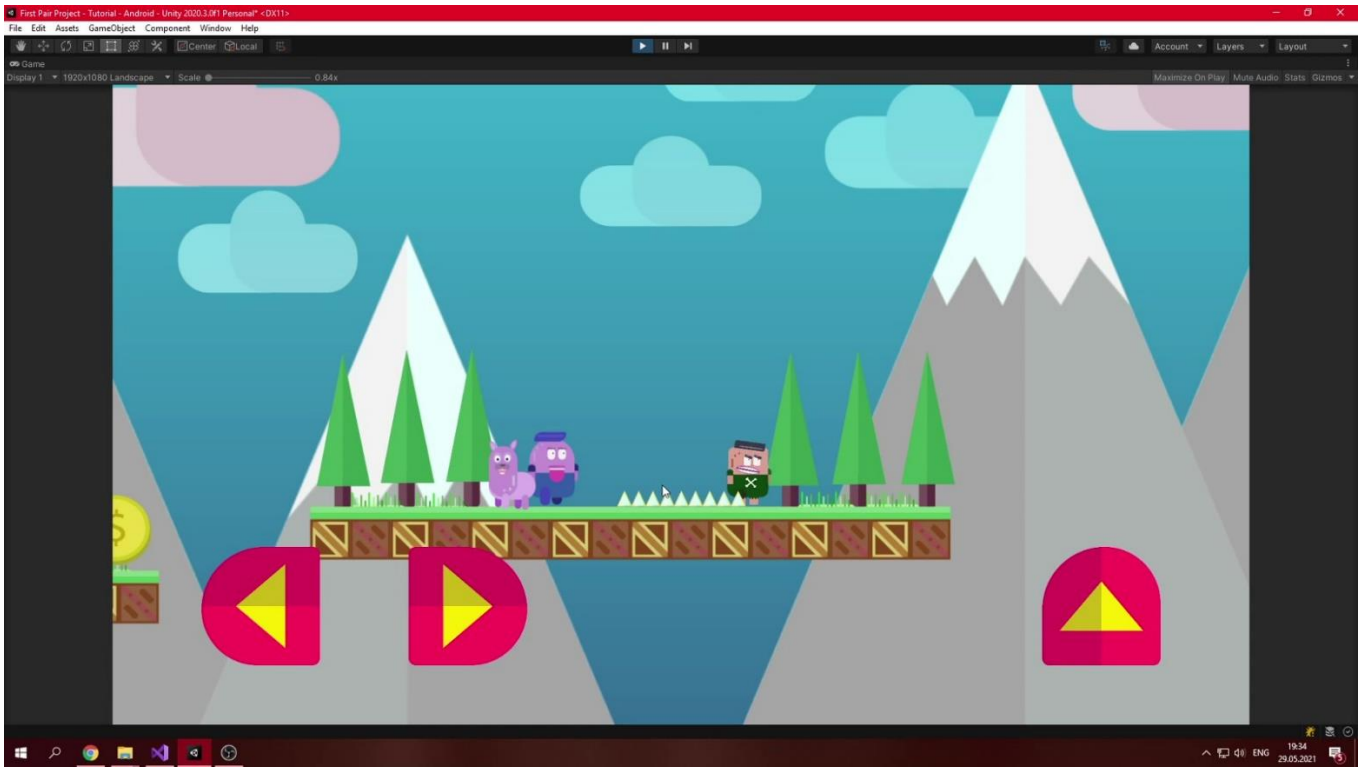
Приклад UI та WTB бага під час тестування



Приклад Missions-LD бага під час тестування



Приклад STP бага під час тестування



Приклад World-LA бага під час тестування