Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка
Кафедра програмних систем і технологій

УДК

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: ''Розробка комп'ютерної гри 3D платформера з елементами квесту"

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ-31.00.00.00

Студент групи ІПЗ-43
Павло ЮРКО
(підпис) (розшифровка підпису)(дата)

Науковий керівник
к. т. н., доц. Катерина МЕРКУЛОВА
(посада) (підпис) (дата) (розшифровка підпису)

Консультант питань з нормоконтролю
фахівець. Тамара ЧАПОВСЬКА
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту
з питань нормоконтролю
Завідувач кафедри
д.т.н., проф. Олексій БИЧКОВ
(посада) (підпис) (дата) (розшифровка підпису)

Київ – 2021

Рішенням Екзаменаційної комісії
випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова ЕК№1

Андрій БОНДАРЧУК

(посада) (підпис) (дата) (розшифровка підпису)

Київський національний університет імені Тараса Шевченка Факультет інформаційних технологій Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:

Завідувач кафедри програмних
систем і технологій
_____ (Олексій БИЧКОВ)
„___”_____ 20___р.

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Юрку Павлу Петровичу

1. Тема випускної кваліфікаційної бакалаврської роботи ‘’Розробка комп'ютерної гри 3D платформера з елементами квесту’’ затверджена наказом вищого навчального закладу від „__”__20__р. №___
2. Строк здачі студентом закінченої роботи _____
3. Вихідні дані до роботи _____
_____
_____
_____
_____
_____
4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
_____
_____
_____
_____
_____
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)
_____
_____
_____
_____
_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Огляд теоретичного матеріалу по завданню роботи	22.09.2020	
2	Аналіз методів та алгоритмів	25.10.2020	
3	Аналіз аналогів, прототипів та перегляд вже готових продуктів, схожих на розроблювальний продукт	10.11.2020	
4	Проектування сюжету та вибір як реалізувати продукт	11.02.2021	
5	Проектування гри та розробка графічної частини гри	09.03.2021	
6	Розробка рівнів гри та скриптової частини гри	02.04.2021	
7	Аналіз готового продукту і оформлення дипломної роботи	07.05.2021	
8	Затвердження пояснювальної записки роботи завідувачем кафедри	3.06.2021	

Студент – бакалавр _____(Павло ЮРКО)

Керівник роботи _____(Катерина МЕРКУЛОВА)

АННОТАЦІЯ

У даній дипломній роботі було розглянуто різні види, жанри та додаткові комп’ютерних ігор, а також різноманітні движки, які можна використовувати для створення та розробки ігор. Для цього був обраний конкретний жанр та елементи для створення гри і обрано движок за допомогою якого можна реалізувати гру.

Тема: Розробка комп'ютерної гри 3D платформера з елементами квесту;

Об’єкт дослідження: об’єктом дослідження були комп'ютерні відеоігри, їх жанри, елементи, функції та движки на яких їх створюють;

Мета роботи: Розробити комп'ютерну гру жанру платформера, протестувати її на можливість проходження та при наявності, виправити помилки та баги;

Предмет дослідження: Предметом дослідження були ігри виду платформеру, його елементи, движок Unity та 3D графіка;

Результати дослідження: За результатами дослідження було створенно платформер який має основні елементи подібного жанру та освоєнно навички роботи з движком Unity;

Висновок: Було побудовано, реалізованно та опубліковано 3D гру платформера з елементами квесту.

Дипломна робота складається з : 42 сторінки друкованого тексту, 1 таблиці, 4 скрипти коду та 24 рисунки, 16 джерел та 3 додатків обсягом 6 сторінок.

АННОТАЦИЯ

В данной дипломной работе были рассмотрены различные виды, жанры и дополнительные компьютерных игр, а также различные двигатели, которые можно использовать для создания и разработки игр. Для этого был выбран конкретный жанр и элементы для создания игры и избран движок с помощью которого можно реализовать игру.

Тема: Разработка компьютерной игры 3D платформера с элементами квеста;

Объект исследования: объектом исследования были компьютерные видеоигры, их жанры, элементы, функции и движки на которых их создают;

Цель работы: Разработать компьютерную игру жанра платформера, протестировать ее на возможность прохождения и при наличии, исправить ошибки и баги;

Предмет исследования: Предметом исследования были игры вида платформер, его элементы, движок Unity и 3D графика;

Результаты исследования: По результатам исследования было создание платформер который имеет основные элементы подобного жанра и освоению навыки работы с движком Unity;

Вывод: Было построено, реализованных и опубликовано 3D декабря платформера с элементами квеста.

Дипломная работа состоит из: 42 сторінки друкованого тексту, 1 таблиці, 4 скрипти коду та 24 рисунки, 16 джерел та 3 додатків обсягом 6 сторінок.

SUMMARY

This thesis discusses the different types, genres and additional computer games, as well as various engines that can be used to create and develop games. To do this, a specific genre and elements were chosen to create the game and an engine was selected with which you can implement the game.

Topic: Development of a computer game 3D platformer with elements of the quest;

Object of research: the object of research were computer video games, their genres, elements, functions and engines on which they are created;

Purpose: To develop a computer game of the platformer genre, to test it for the possibility of passage and, if available, to correct errors and bugs;

Subject of research: The subject of research were platformer type games, its elements, Unity engine and 3D graphics;

The results of the study: The results of the study created a platformer that has the basic elements of this genre and mastered the skills of working with the Unity engine;

Conclusion: A 3D platformer game with quest elements was built, implemented and published.

Thesis consists of: 42 pages of printed text, 1 table, 4 code scripts and 24 figures, 16 sources and 3 appendices of 6 pages.

ЗМІСТ

Перелік основних позначень, символів, скорочень

Рис. – рисунок;
Табл. – таблиця;
Код: – приклад коду для певних дій;
Баг – помилка;
Геймплей – ігрове керування;
Шутер – гра у якій герой стріляє;
Хоррор – гра з моментами жахів;
Глюк – помилка, яка може спияти погрозі усій грі;
Пофіксити – полагодити;
Мерчандайзинг – товари;
Стікер – наклейка;
Пікапс – предмети, які може підбирати герой;

ВСТУП

Актуальність – іншими словами значущість досліджуваної теми й проблематики. В актуальності найголовніше обґрунтувати, чому ця тема гідна вивчення, наскільки важливі результати дослідження, де і ким вони можуть використовуватися (теоретична і практична вагомість).

Актуальністю моєї роботи є те, що у нинішньому суспільстві є багато людей, особливо підлітків та дітей, які обожнюють грати у відеоігри.

Відеоігри також можуть навчити важливим навичкам або вирішити серйозні проблеми, такі організації, як "Ігри за зміни", пропагують використання ігор для освіти та соціальних дій, і часто залучають молодих людей до створення ігор, які вони можуть використовувати для висловлювання важливих питань події. Деякі медичні працівники, такі як доктор Курош Діні, також вважають, що «багатокористувацькі відеоігри, що відповідають віку, можуть дозволити дітям дізнатися, як думають інші люди - ключовий аспект співпереживання. Ігри також можуть допомогти дитині почуватися комфортніше за допомогою нових і постійно прогресуючих технологій ".

Недавні дослідження показали, що відеоігри можуть мати позитивний чи негативний вплив на поведінку гравців, залежно від вмісту. Просоціальні ігри можуть посилити співпереживання, співпрацю, допомогу та емоційну обізнаність, тоді як жорстокі ігри можуть зменшити ці риси.

У согоднішній час, складно знайти людей, які ніколи не грали у відеоігри.

Метою даної роботи є створення та реалізування компютерної гри жанру платформер з елементами квесту. Для цього були дослідженні різновиди платформерів, їхні види графіки, геймплей та які є квести і які з них більше підходять до платформерів. Огляд та аналіз популярних представників, розбиття засобів на класи. Постановка задач для тесових проектів ігор-прототипів, у розрізі спрощених вимоги до сучасних ігрових проектів. Детальний аналіз найбільш характерних представників кожного класу, розробка з їх допомогою ігор прототипів. Визначення оптимальних сфер використання для інструментальних засобів спираючись на аналіз процесу розробки прототипів.

Протягом багатьох років відеоігри критикуються за те, щоб зробити людей більш антисоціальними, надмірними вагами або пригніченими. Але тепер дослідники знаходять, що ігри можуть насправді змінити нас на краще і покращити як наше тіло, так і розум.

Ігри можуть допомогти розвивати фізичні навички. Показано, що дошкільні діти, які відігравали інтерактивні ігри, такі як доступні в Wii, мають покращені моторні навички, наприклад, вони можуть вдарити, ловити та кинути м'яч краще, ніж діти, які не грають у відеоігри. Вивчення хірургів, які роблять мікрохірургію в Бостоні, виявили, що ті, хто грав у відеоіграх, склав 27 відсотків швидше, і склали 37 відсотків менше помилок, ніж ті, хто цього не зробив. Бачення також вдосконалюється, особливо розповідаючи різницю між відтінками

сірого. Це корисно для водіння вночі, пілотування площини або читання рентгенівських променів.

Ігри також користуються різноманітними функціями мозку, включаючи прийняття рішень. Люди, які грають у грі на основі дій, приймають у рішенні 25 відсотків швидше, ніж інші, і не є менш точними, відповідно до одного дослідження. Було також виявлено, що найкращі геймери можуть зробити вибір і діяти на них до шести разів на секунду, чотири рази швидше, ніж більшість людей. В іншому дослідженні дослідників з Університету Рочестера в Нью-Йорку, досвідчені геймери мали можливість звернути увагу на більше шести речей відразу, не заплутаними, у порівнянні з чотирма, ніж більшість людей можуть нормально мати на увазі. Крім того, відеоігри також можуть зменшити гендерні відмінності. Вчені виявили, що жінки, які грають у іграх, краще здатні розумово маніпулювати 3D-об'єктами.

Існує також доказ того, що ігри можуть допомогти з психологічними проблемами. У університеті Окленда в Новій Зеландії, дослідники попросили 94 молодих людей, діагностовано депресію, щоб грати в 3D-фентезійну гру під назвою Sparx, і в багатьох випадках гра зменшила симптоми депресії більш ніж звичайного лікування. Ще однією дослідницькою командою Оксфордського університету виявила, що граючи в Тетріс незабаром після впливу на щось дуже засмучене - в експерименті використовувалася плівка травматичних сцен пошкодження та смерті - це може не заважати людям, що тримають порушення спортсменів.

Однак ефекти не завжди настільки позитивні. Дослідники університету Індіана здійснили сканування мозку на молодих чоловіків, і виявили докази того, що насильницькі ігри можуть змінювати функцію мозку після того, як за тиждень відтворення, що впливають на регіони, пов'язані з емоційним контролем та викликаючи більш агресивну поведінку у гравці. Але Дафна Бавевера, один з найсувовіших дослідників у цій галузі, каже, що насильницькі дії, які часто хвилюються батьками, насправді можуть мати найсильніший благотворний вплив на мозок. У майбутньому можно побачити багато процедур для фізичних та неврологічних проблем, які включають відтворення відеоігор.

Відеоігри були навколо протягом десятиліть, забезпечуючи розваги для дітей та дорослих. Вони істотно розвивалися з перших днів комп'ютерних ігор та перших версій Nintendo та Atari. Дні періодичних екранів та обмежених звуків є далекою пам'ять, оскільки відеоігри стали більш реалістичними, ніж будь-коли. Оскільки технологія продовжує вдосконалюватися, так і відеоігри.

Створення відеоігор стало все більш складним, а вартість створення гри для запуску на одній з основних консолей зросла з цією більшою складністю. Це було колись немислимо, щоб занурити мільйони до витрат на розвиток, але ігри сьогодні можуть коштувати десятки та навіть сотні мільйонів. Це підштовхнуло розвиток гри в кінотеатр Голлівуду з точки зору виробничих та маркетингових витрат.

Сектор відеоігор надзвичайно великий. Фактично, це більше, ніж фільм та музична промисловість, і вона зростає. Хоча це не розуміє, що робить фільм та музичну індустрію, у всьому світі є понад два мільярди геймер. Це 26% світового населення.

Не дивно, що компанії хочуть шматок пирога. Аналітики прогнозують, що до 2022 року ігрова промисловість отримає 196 мільярдів доларів у доході. Таким чином, технічні компанії прагнуть залучити до цього потоку доходів.

Як і Голлівуд, індустрія відеоігор повинна обертатися більш доходом від своєї інтелектуальної власності, оскільки продукт коштує багато чого зробити. Мерчандайзинг вже навколо, з футболками, фігурами, капелюхами, кружками тощо. Серія Halo на Microsoft Xbox поширилася на інші форми вмісту через романи та комічні книги на додаток до майбутніх телевізійних серій та довгострокового фільму. Це може стати підходом до всіх успішних серій відеоігор, щоб слідувати.

Насправді, популярна відеоігри, Assassin's Creed, гра, розроблена Ubisoft, була зроблена у фільмі у 2016 році з видатними акторами. Популярна гра Sonic Hedgehog була зроблена у успішному фільмі в 2020 році, головним чином відомі актори, а також встановив рекорд для найбільших вихідних для відеоігор.

РОЗДІЛ 1
ОСНОВНІ ПОНЯТТЯ ТА ПРЕДМЕТНА ОБЛАСТЬ

1.1. Вивчення предметної області

Відеоігри мають великий позитивний потенціал, крім своєї розважальної цінності, і досяг значного успіху, коли ігри розроблені для вирішення конкретної проблеми або для навчання певним навичкам. Відеоігри можуть чітко споживати увагу уваги дітей та підлітків. Однак це важливо оцінити, наскільки технологія відеоігор мала вплив на освіту дитинства. Оскільки відеоігри здатні залучати дітей до навчального досвіду, це призвело до зростання засобів масової інформації. Просто спостерігаючи за дітьми, стає абсолютно зрозумілим, що вони віддають перевагу саме такому підходу до навчання. Однак виявляється, що дуже мало ігор на комерційному ринку мають освітню цінність. Деякі дані свідчать про те, що важливі навички можуть бути створені або підсилені відеоіграми. Наприклад, здатність до просторової візуалізації (тобто, помітно, обертання та маніпулювання дво- та тривимірними об’єктами) покращується із відтворенням відеоігор. Відеоігри також були ефективнішими для дітей, які починали з відносно низькою кваліфікацією. Також висловлюється припущення, що відеоігри можуть бути корисними для вирівнювання індивідуальних відмінностей у просторових показниках навичок. Понад 20 років дослідники використовують відеоігри як засіб дослідження окремих людей. Багато з цих причин також дають уявлення про те, чому вони можуть бути корисними в освіті. Наприклад:

- Відеоігри можуть бути використані як інструменти дослідження та / або вимірювання. Крім того, як інструменти дослідження вони мають велику різноманітність
- Відеоігри залучають людей до участі у багатьох демографічних межах (наприклад, вік, стать, етнічна приналежність, освітній статус)
- Відеоігри можуть допомогти дітям у постановці цілей, забезпеченні репетиції цілей, забезпеченні зворотного зв'язку, зміцненні та веденні записів про зміни поведінки
- Відеоігри можуть бути корисними, оскільки вони дозволяють досліднику виміряти ефективність виконання найрізноманітніших завдань, і їх можна легко змінити, стандартизувати та зрозуміти
- Відеоігри можуть бути використані при вивченні індивідуальних характеристик, таких як самооцінка, Я-концепція, постановка цілей та індивідуальні відмінності
- Відеоігри є цікавими та стимулюючими для учасників.

Незважаючи на недоліки, здається, що відеоігри (у правильному контексті) можуть бути сприяючою освітньою допомогою. Отже, легше досягти і

підтримувати нерозділену увагу людини протягом тривалих періодів часу. Через веселощі та хвилювання вони також можуть забезпечити інноваційний спосіб навчання. Відеоігри можуть забезпечити елементи інтерактивності, які можуть стимулювати навчання. Відеоігри також дозволяють учасникам відчути новизну, цікавість та виклики. Це може стимулювати навчання. Відеоігри оснащують дітей найсучаснішими технологіями. Це може допомогти подолати технофобію (стан, добре відомий серед багатьох дорослих). З часом це також може допомогти усунути гендерний дисбаланс у використанні IT (оскільки чоловіки, як правило, більш завзяті користувачі IT) Відеоігри можуть допомогти у розвитку передавальних IT-навичок. Відеоігри можуть виступати в ролі симуляції. Вони дозволяють учасникам займатися надзвичайною діяльністю та знищувати або навіть померти без реальних наслідків Відеоігри можуть допомогти підліткам регресувати до дитячих ігор (через можливість зупинити реальність у грі відеоігор).

Звичайно, є деякі недоліки у дослідженні відеоігор в освітньому контексті. Наприклад :

- Відеоігри викликають у учасників захоплення, і тому вони створюють цілу низку незрозумілих змінних, таких як мотивація та індивідуальні навички11
- Технологія відеоігор швидко змінюється з часом. Тому відеоігри постійно вдосконалюються, що ускладнює оцінку освітнього впливу в ході досліджень
- Досвід і практика відеоігор можуть покращити ефективність учасника в певних іграх, що може призвести до погіршення результатів.

1.2. Обраний движок для реалізації продукту

Незважаючи на те, що більшість усього популярних 3D ігор було створено на движку Unreal Engine, для реалізації проекту було використано Unity.

Unity є другим по популярності та використанні движком, хоча це твердження є спірним. Обидва движка є достатньо популярними, але однією із головних різниць мож ними є графіка, та память, на яку вона припадає. Unreal Engine потребує багато памяті на графіку, тому використовується лище у великих проектах. Unity натомісь, можливо використати як для обох графік 2D і 3D.

Таблиця 1.1

Порівняння Unity та Unreal

Параметри	**Unreal Engine**	**Unity**
Визначення	Джерело доступного ігрового движка.	Крос-платформний ігровий движок.

Розроблено	Epic Games	Unity Technologies
Мови програмування	Він використовує мову C ++ або Javascript для розробки.	Для розробки використовується C#.
Використання	Використовується для розробки ігор для ПК, мобільних телефонів. консолі та багато іншого.	Використовується для розробки ігор для ПК, мобільних телефонів. консолі та багато іншого.
Особливості	Надійна багатокористувацька структура, VFX та моделювання частинок.	2D-вдосконалення, анімація, створення знімків.
Вихідний код	Вихідний код є відкритим.	Вихідний код не є відкритим.
Нагороди	Премія «Новачок року» у 2018 році.	Премія "Unity" у 2018 році.
Ціноутворення	Це безкоштовно.	Базова версія безкоштовна.
Крива навчання	Важко вчитися.	Легко та весело навчатися, оскільки він має інтуїтивно зрозумілий інтерфейс.
Графіка	Якість AAA хороша графіка.	Графіка хороша, але Unreal тут краща.

Unity також є одним із движків, з якими можно легко працювати і швидко вчитися. Додатково використовуючи особливий Unity сайт https://assetstore.unity.com, де можно викачати безкоштовні пакети, для реалізації гри.

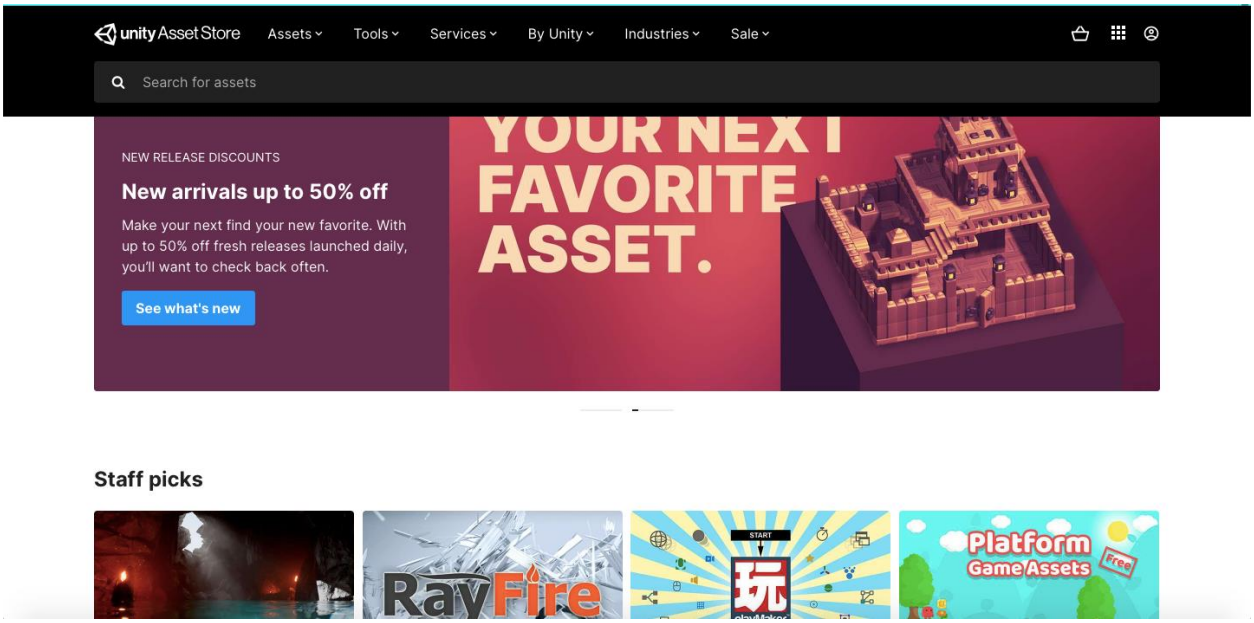


Рис 1.1. Unity assetstore де можна придбати додаткові моделі та різні предмети та ефекти.

Можливо викачати моделі, матеріали, шейдери, аудіо, анімацію, предмети для обох видів графік, інструменти такі як:

- Штучний інтелект;
- Анімація;
- Візуальний скиптинг;
- Земля (На якій персонаж вже стоїть і яка вже має колайдер, без функції додавання тв скриптингу).

Плюси та мінуси Unity:

Плюси:

1. Розгортання - Unity експортує майже на всі можливі платформи: мобільні, консолі, ПК, VR, є, з незначним перенесенням та необхідними особливими функціями платформи.
2. Гнучка - Єдність не обмежується певним типом гри. Незважаючи на те, що 2D-ігри складніше і менш інтуїтивно зрозумілі за допомогою двигуна, все можливо, якщо є технічні можливості.
   Простота використання - порівняно з іншими движками, такими як Unreal та CryEngine, крива навчання Unity досить низька, і движок може підхопити кожен, хоча рекомендується мати деякі попередні знання з програмування та / або створення ігор.
3. Спільнота - Unity має одне з найбільших - якщо не найбільше - співтовариство, коли йдеться про ігрові двигуни. В Інтернеті є ресурси, починаючи від простих підручників для початківців і закінчуючи більш

досконалими та повними підручниками. Натрапивши на проблему, яку не можно вирішити, є безліч місць в Інтернеті, де є можливість запитати.

Мінуси:

1. Якщо створюється невеличка мобільна гра, Unity, мабуть, не підійде для цього - її розміри (додаток) величезні. Порожня сцена в Unity дає 10 Мб мобільного додатка.
2. Вихідний код Unity недоступний для загального користування. Виявивши помилку двигуна, єдине, що можливо зробити, - це подати звіт про помилку і молитися, щоб її виправити. Можна ліцензувати вихідний код Unity, але, швидше за все, це непотрібно для потреб і надзвичайно важко.
3. Якщо заробляти більше 100 тис. Доларів на рік, тоді потрібно придбати ліцензію Unity Plus (до 200 тис. Доларів брутто) або ліцензію Unity Pro (необмежена брутто).
4. На відміну від подібних двигунів, таких як Unreal Engine, є набагато вдосконаленіші функції, які не виходять з коробки, переважно щодо графіки та post-fx. Потрібно зробити їх самостійно, або можно перейти до магазину активів і шукати вже зроблене рішення.
5. Якщо експортувати в iOS або OSX, потрібно xCode, щоб скомпілювати гру, яка може працювати лише на Mac.

1.3. Аналіз вже існуючих продуктів

Платформер – є одним із популярних жанрів ігор, які обожнюють грати люди будь-якого віку, а також цей жанр можно змішувати з іншими для отримання ще кращого продукту, який сподобається ігрокам. Є багато старих платформерів, які до сих пір є популярними такі як Super Mario Bros, Сонік, Crash Bandicoot.

Багато яких пратформерів є зараз популярними і кожен із них заслуговує мати місце у списку найкращих. Кожен із таких був побудований по різному, із різною графікою, движками, геймплеями та сюжетами. Прикладами для аналізу вже існуючих продуктів було обранно такі чотири гри: Ape Escape 3, Sonic Adventure 2, Little Nightmares та LEGO Star Wars: The complete saga.

Ape Escape 3.

Рис 1.2. Герой використовую форму Jeene Dancer, щоб заставити мавпу танцювати.

Ape Escape 3 - це гра на платформі, але стратегія та вирішення головоломок набагато більше, ніж є у більшості ігор цього жанру. Розташування втеклих мавп і шлях до них - це така ж частина ігрового процесу, як насправді їх захоплення. У грі також є стелс-елемент: підкрадання мавп і здивування їх робить їх набагато простішими для лову, ніж якщо вони помітять героя і почнуть бігати, як божевільні чи гірше, почнуть атакувати. Кожна мавпа в грі носить на голові маленьку поліцейську сирену, яка перемикається на жовту, якщо їм стає підозріло, що хтось поруч, і червоніє, якщо помічають гравця. Це хороша візуальна допомога при спробі підкрастися до мавпи, але це також додає відчуття пандемонії, яке виникає, коли кілька мавп бігають навколо з блимаючими червоними сиренами, що спрацьовують повним ходом.

Одним з найкращіх плюсів цієї гри є кольорова графіка та більша різноманітність у ворогах та в мавпах. У серіях гри 1 і 2, вороги були однакові і мавпи також не мали якихось цікавинок. У цій весії вороги різні, на кожному рівні вороги міняють тематику, а також зброю. Мавпи та боси також одягеуті під тематику рівнів, мають різноманітні здібності, атакож через додаткові костюми складно зрозуміти якого кольору у них штани; колір штанів відповідає якого типу

є мавпа. Наприклад мавпи із темно-синім кольром бігають дуже швидко, а з червоним кольором атакують гравця частіше.

Рис 1.3. Мавпи із різнокольоровими штанами, кожен з яких має особливі здібності.

Як і кожна гра, завжди будуть мінуси. Першою є невелика проблема у графіці, оскільки не будо створено анімацію на обличчя героів, окрім фільмових сцен, але і там анімація слабка. Другою проблемою є те що неможливо спіймати усіх мавп які є на рівні за один раз, оскільки деяких можна спіймати лише за допомогою спеціальних гаджетів і особливої трансформації, яка дається доступною наприкінці історії, тому рівень треба проходити ще раз.

Для проекту дипломної роботи було використано такі плюси: графіка гри є кольоровою, для дітей та дорослих, також додано різних ворогів у грі разом із іншими моделями. Також є мінус, герой не має анімації обличчя. Рівні можно пройти з одного разу, не повертаючись до них.

Little Nightmares.

Little Nightmares - це пригодницька гра від третьої особи з елементами стелс та розвідки. Маленьку дівчинку на ім'я Шість викрадають з дому та вивозять на роботу в сюрреалістичний підводний курорт Моу. Коли їй вдається врятуватися, Шість здійснює подорож дивним і непередбачуваним світом.

Рис 1.4. Шість ховається від кухаря.

Кожного разу потрібно ховатися від ворогів, оскільки можливісь боротися відсутня і герой занадто малий. Ця гра створенна у жанрі хоррора, щоб надати більше адреналіну, а в такі ігри багато людей грають. Керування як і в більшості платформерах слабке: є можливість бігати, ходити і взаємодіяти із предметами. Однак головною ідеєю цього проекту не пройти гру швидко, а обережно, оскільки, щоб пройти певний рівень потрібно продумати що робити і при цьому не попасти на очі ворогам.

Додатковими також є елемнети квестів, такі як пошук ключів, перетягування ящиків, натискання кнопок. Це є важливою частиною, без якої гравець не зможе продовжувати гру.

Недоліками у грі є баги, які не були фіксовані і до сих пір є у грі. Самим головним і проблемним багом є те, що модети персонажів можуть глючити, як наприклад обертатися невірно, через що в кінці страждає проходження гри; гравець не зможе пройти гру до кінця.

Для проекту використані елементи квесту, які є у грі, при цьому гра реагує на дії та функції моментально, без ніяких багів.

LEGO Star Wars: The complete saga.

Рис 1.5. Приклад лего гри, сцена “Прибуття на Каміно”.

LEGO Star Wars: The Complete Saga надає жартівливий погляд на фільми як трилогії приквелу, так і оригінальної трилогії. Гра була розроблена Traveler's Tales, яка також створила інші ігри серії. LucasArts опублікували гру, як і в LEGO Star Wars II: The Original Trilogy. Він містить рівні та персонажів перших двох ігор, але має додатковий вміст для рівнів. Всього доступно 128 персонажів, а також додатковий та розширений рівні. Гра не надто глибоко заглиблюється в сюжет саги; скоріше, він дає короткий зміст з певною увагою до важливих подій, але в жартівливій формі. Персонажі також не говорять, лише ропочуть і вокалізують.

Рівні, спочатку знайдені в LEGO Star Wars: The Video Game, мають вміст, подібний до рівня LEGO Star Wars II: The Original Trilogy, такий як Power Bricks та транспортні засоби. Нові функції гри включають розширені сили Force, нові бонуси та новий режим виклику. Рівень швидкості швидкості, спочатку вилучений з першої гри, є ігровим рівнем, а також Зам Везеллом. Велика частина гри обертається навколо збирання «монеток», невеликих частин LEGO, які використовуються як ігрова валюта.

Одним із плюсів було те, що у грі є багато різноманітних персонажів за яких можно грати, однак це також є і мінусом, оскільки є багато персонажів, які взагалі є даремними і не грають взагалі ніякої ролі у проходженні гри.
Також плюсом є те що у грі існують бонусні рівні, які весело проходити, а також є рівні, які прохядяться за допомогою транспортів. Є багато бонусів і квестів які містять жартівливий характер, що підходить для гравців усього віку.

Рис 1.6. Приклад рівня з транспортом.

Було додано до проекту моменти жартівливого характеру, а також створено додатковий рівень і багато різних квестів, щоб зробити рівень складніше. Також у проекті є можливість грати лише одним героєм і можливо пройти гру з одного разу.

Sonic Adventure 2.

Рис 1.7. Перший рівень у грі Sonic Adventure 2.

Більшу частину гри потрібно проходити на час, оскільк чим швидше тим краще. Під кінець рівня завжди дається оцінка за проходження кожного рівня, тобто як швидко було пройденно і як багато кілець було зібранно. Це також є і проблемою, оскільки текстури іноді не встигають за персонажем і може трапитися глюк, через який вже пройти повністю гру вже буде неможливо. Також у самій графіці є багато багів у текстурі та дизайні рівнів, які до сих пір не були полагодженні.

Хоча не зважаючи на велику кількість проблем, одним із плюсів є бонусні рівні та секрети, які можна знайти на кожному рівні, при цьому вони навіть можуть допомогти гравцеві дістатися до кінця рівня навіть швидше, ніж нормальним шляхом.

Це було одним із елементів, які були додані до проекту, створення бонусного рівня та секретів. При цьому жоден рівень не настроєн на проходження їх якомога швидше, головна ціль проекту – пройти гру, обережно, не загинувши, оскільки потім потрібно буде проходити рівень знову.

РОЗДІЛ 2
РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Постановка задачі

Незважаючи на те, що розвиток відеоігор є хаотичним характером, існують ще структури та рамки, які мають бути ефективні студії та ефективні та проекти для завершення роботи. Етапи розвитку гри йдуть, таким чином:

Рис 2.1. Схема планування розробки відеоігор.

1. Планування відеоігор

Перед письменниками починають писати, дизайнери починають проектувати, а розробники починають розвиватися, ідея для відеоігор повинна повертати. Це перша частина стадії планування та коріння, з яких кожна відеоігри зростатимуть.

На етапі планування, найбільш основні питання потрібно відповісти, як:

- Який тип відеоігри ми виробляємо?
- Це буде 2D або 3D?
- Які деякі з ключових функцій вона повинна мати?
- Хто його персонажі?
- Коли і де це відбувається?

- Хто наша цільова аудиторія?
- Яку платформу ми будуємо це?

Це, можливо, не схоже на це, але здивування відеоігор є однією з найважчих частин розвитку гри. Ідея ігрової студії виникає з боку всієї гри. Це те, що встановлює стандарт для кожного працівника, який бере участь у будівництві гри, але також дає видавці видобуток, що очікувати. Це приносить нам до наступної частини розвитку - перевірка концепції. Доказом концепції приймає всі ідеї, які були створені, і бачить, наскільки життєздатними для ігрової студії для виробництва. Звідти, потрібно відповісти додаткові питання, як:

- Яка наша оціночна вартість розробити цю гру?
- Чи є у нас технологічні можливості для його побудови?
- Чи потрібна нам новий ігровий двигун?
- Наскільки велика буде наша команда?
- Ми наймаємо зовнішніх голосових акторів і письменників?
- Що таке наш розрахунковий час для запуску?
- Як ми його монетизуємо?

Для студій, які будують гру під парасолькою видавця, необхідна перевірка концепції, перш ніж рухатися вперед з попередньою продукцією, і навіть може вимагати вертикального скибочка. Це тому, що видавець повинен буде схвалити поле за часом, бюджет та маркетингу. Для незалежних студій без нагляду видавця, на цьому етапі існує трохи гнучкість. Недоліком незалежного видавництва встановлює розвиток та маркетинговий бюджет, хоча, у всьому світі, як задачі, такі, як KickStarter та Fig. По суті, успішні ігри, такі як Pillars of Eternity та Shovel Knight, були повністю зараховані.

Який маршрут буде обранний, доказ концепції є життєво важливим для успіху гри, оскільки він ставить ідеї в перспективі того, що здатне.

2. Попереднє виробництво

Наступний етап розвитку гри, який називається попередньою продукцією, мозкові штурми, як дати життя багатьом ідеям, викладеним у етапі планування. Саме тут письменники, художники, дизайнери, розробники, інженери, проектне керівництво та інші важливі департаменти співпрацюють на сферу відеоігор і де кожен шматок головоломки підходить. Кілька прикладів цієї співпраці можуть виглядати:

- Письменники зустрічаються з проектом, що веде до плоті розповіді про історію. Хто є головними героями в цій казці? Які їхні підрізи? Як кожен персонаж відноситься один до одного? Чи є вільні кінці, нам доведеться зв'язати пізніше?
- Інженери, що зустрічаються з письменниками, дозволяючи їм знати, що за сучасними технологічними обмеженнями ми не можемо заповнити це середовище з 100 символами або гра.

- Художники зустрічаються з дизайнерами, щоб забезпечити візуальні, кольорові палітри, а стилі мистецтва узгоджуються та вирівняні з тим, що було викладено на етапі планування.
- Розробники зустрічаються з інженерами, щоб вийти з усієї механіки, фізики в грі, фізику, і як об'єкти відтворюють на екрані гравця.
- Проект призводить до зустрічі з декількома підрозділами, щоб з'ясувати "Fun Factor,", не легко визначити, до етапу тестування.

3. Виробництво

Більшу частину часу, зусиль, витрачених ресурсів, витрачених на розробку відеоігор, знаходяться під час виробничої стадії. Це також відбувається одним з найбільш складних етапів розвитку відеоігор. Під час цього процесу:

- Моделі персонажів розробляються, надаються, і витісняються, щоб виглядати точно, як вони повинні в історії.
- Аудіо-дизайн працює невтомно, щоб гарантувати кожен раз, коли персонаж наступить на пісок, гравій, або цемент, звучить автентичний.
- Рівень дизайнерів ремісничих середовищ, які є динамічними, зануренням і підходять для багатьох видів PlayStyles.
- Голосові актори читають великі стеки сценаріїв, роблячи після того, як взяти, щоб отримати правильну емоцію, час та тон.
- Розробники пишуть тисячі-від ліній-вихідного коду, щоб принести кожну частину вмісту в грі до життя.

Проект призводить до встановлених етапів та розкладів спринтів, що забезпечують кожного відділу та її членів команди. Це особливо важливо, якщо видавець регулярно перевіряє оновлення статусу.

4. Тестування

Кожна особливість та механік у грі повинні бути перевірені на контроль якості. Гра, яка не була ретельно перевірена, - це гра, яка навіть не готова до альфа-випуску. Ось деякі речі, які можна вказати на цьому етапі Playtertester:

- Чи є баггі райони або рівні?
- Чи все надає на екрані?
- Чи можу я пройти цю стіну або заблоковане середовище?
- Чи є функції, я можу використовувати для експлуатації гри?
- Чи є мій персонаж постійно застряг у цьому місці?
- Чи є діалог символів та нудним?

Є навіть різні типи плейтерів. Деякі програмні тестування проводять стрес-тести, бігаючи в стіни сотні, якщо не тисячі разів у спробі "зламати" гру. Інші PlayTesters проводять тестування "Fun Factor", щоб побачити, якщо гра занадто жорстка або занадто легко, або завершити всю гру, щоб побачити, чи

задовольняється достатньо. Без "веселого фактора", гра не буде продавати багато копій.

5. Пред-запуск

Етап попереднього запуску - це напружений час для ігрових студій. Питання самовинних сумнівів можуть побачити, як з'являється питання, як громадськість реагує на перший функціональний продукт.

"Чи будуть вони думають, що наша гра весело? Вони збираються знайти нові помилки? Яке висвітлення медіа ми збираємося отримати від цього?"

Але до того, як формальна бета-копія випущена, гра вимагатиме певного маркетингу. Зрештою, як ще люди дізнаються про це?

Видавництво майже завжди очікують відео-відео з поєднанням кінематографічних та вибіркового геймплея, щоб привернути увагу. Вони також можуть запланувати місце в одному з основних ігрових конвенцій, як E3 або PAX, для ексклюзивного попереднього перегляду гри.

6. Запуск

Фінішна лінія близька. Світло знаходиться в кінці тунелю. День запуску знаходиться на горизонті.

Місяць, що ведуть до очікуваної дати запуску гри, в основному витрачають журнальні великі відстані від помилок - деякі старі, деякі нові знайдені на етапі тестування. Для ігор з багатьма помилками студія створить ієрархію помилок до сквошу. Ця ієрархія включатиме "ігрові" помилки біля верхньої та незначної помилок біля дна.

На додаток до жукування журналу, розробники, як правило, польська гра якомога більше, перш ніж вона запускає. Можливо, що гірський хребет може мати більше глибини. Можливо, шкіряні ремені персонажа можуть бути більш текстурованими. Давайте нарешті обійдеться, щоб зробити ці дерева під час вітру. Ці типи змін, хоча незначні, можуть бути важливими для створення відеоігор більш занурено.

7. Пост-запуск

Пост-запуск - це один з найцікавіших часів для будь-якої ігрової студії. Роки важкої роботи, нарешті, виплатилася, а продаж відеоігор, розливуються. Але навіть зараз, все ще працює.

Це не рідкість для відеоігор для запуску з партіями незначних помилок. Перші кілька місяців на етапі пост-запуску, як правило, витрачаються на ідентифікацію та розгортання цих помилок. Gaming Studios також покладаються на гравців, щоб подати звіти про помилки або говорити про помилки в онлайн-форумах. Це всі частини підтримки пост-запуску.

Інша частина пост-запуску полягає в тому, щоб забезпечити регулярні оновлення програмного забезпечення для гри. Ці оновлення варіюються від балансувальних патчів до нового завантаженого вмісту або DLC.
Випуску свіжого вмісту є загальним у сучасній ігровій галузі, оскільки це збільшує значення відтворення та привабливість гри. Нові рівні, сюжетні лінії та багатокористувацькі режими - це лише деякі з багатьох варіантів DLC, а ігрова студія могла б вивчити.

2.2. Специфікація та вимоги до продукту

Часто простіше визначити гра платформи основними компонентами, які є загальними для багатьох ігор.
Що спільного у успішних платформах?

- Триваючий головний герой з масовим зверненням. Як правило, він милий / крутий, барвистий, і, як правило, "хороший хлопець", який намагається "врятувати світ";
- Історія керованої гри, яка часто використовує багато гумору, щоб розважати;
- Чіткий ворог, який повинен бути переможений в кінці, як правило, з нижчими лейтенатами, які повинні бути поразки першими в кінці кожного рівня;
- Здебільшого, ці ігри прив'язані до однієї консольної платформи;
- Високо інтерактивні середовища та цікаві символи;
- Гладкий геймплей, який досить простий і високо інтуїтивно зрозумілий;
- Додано нещодавно зйомки та водіння автомобіля, але виконання все ще просто і весело;
- Повторення, що поставляється з суб-місіями, і розблокованим або прихованим рівнем;

Змішані жанри

Оскільки жанри сильно поєднуються протягом останніх кількох років, додаткові визначення стали необхідними. І, по суті, змішування вирощувалося лише на наступних системах. Оскільки поєднання настільки поширене і переломе, я не можу надати приклади кожного змішаного жанру на ринку, але я дам приклади деяких значних тут. З метою обговорення змішаних жанрів, таких як платформа / пригоди, пригоди / акція / платформа, або платформа / стратегія / пригоди, у нас немає керівних принципів, які допоможуть визначити жанри. Платформа / пригода: На даний момент подумайте про платформу / пригод, як гра платформи з сильними пригодницькими елементами гри. Поточний приклад цього є особняк Луїджі для Gamecube. Пригоди / Дія / Платформа: Поточний приклад пригод / дії / платформи, це диявол може плакати за Capcom. Платформа

/ Стратегія / Пригоди: Поточний приклад платформи / стратегії / пригод - це мовчазний див.

Що є в платформері?

Питання, що будь-яка платформа повинна задати, - скільки відхиляється від решти. Чи добре бути різним для того, щоб бути різним, або це добре, щоб бути іншим? У минулому існувало платформні ігри, такі як Escape Escape, які мали дуже стандартний загальний геймплей, але він відрізнявся, маючи дуже оригінальну бойову та контрольну схему, яка використовує як аналогові палички, мабуть, вперше в будь-якій грі. Munch's Oddsee поєднує в собі платформні елементи, груповий бойовий, елементи RTS, головоломки та інші типи геймплея, і не має міні-ігор у ньому, а ігри, такі як Voodoo Vince та Pitfen, втрачена експедиція досить загальні у своїй геймплеї та дотримуються стандартів платформ.

Стандартні функції, знайдені в платформі, включають:

- Унікальний і сильний головний герой
- Дії і речі
- Матеріал для збору
- Стрибки та головоломки спритність
- Переміщення платформ та інших перешкод
- Вороги до поразки
- Міні-ігри
- Головоломки
- Квести або завдання для виконання
- Нові здібності по всій грі
- Нові завдання по всій грі

Це, здається, є елементами, які є найбільш поширеними в платформі. Це не означає, що кожна гра повинна мати всі ці функції, але визначити як платформу, яку потрібно мати декілька з цих функцій. Поруч із розробкою сильного головного героя, механіка гра платформи є критичною. Як грає гра, як вона відчуває, що робить це весело, і те, що відрізняє його, можна критично. Механіка та контроль є критичними, щоб зберегти гру від розчарування. Важко знайти баланс між збереженням стрибків, головоломок і бойових завдань, але не жорстким. Якщо занадто важко завершити стрибки, або виконувати інші кроки, або якщо це занадто важко боротися, втратите гравця.

Головоломки - це те, що, як правило, зберігають багато платформних ігор цікаві, і більшість ігор платформи мають деякі типи головоломок у них. Головоломки можуть сильно відрізнятися від гри до гри. Є дві великі групи, в які падають головоломки: на даний момент ми будемо посилатися на них, як заснована на основі думки та заснованої.У головоломок на основі думки, головоломка може центру, як дістатися до елемента або виконати завдання, а в

інших іграх вони можуть бути більш традиційними головоломками, які потрібно вирішити..Основні головоломки на основі Dexterity - більше орієнтованих на дії. Для цих головоломок і веселощів приходить від того, як можна швидко перемістити за рівнем, де потрібно стрибати, і що для виконання рухів на рівні в.

Більшість головоломки на основі спритності також можна вважати головоломки, в цьому Гравець також повинен виконувати дію в потрібний момент або послідовність дій у правих інтервалах часу, щоб бути успішним. Багато головоломки, заснованих на декстерісті, стосуються лише отримання певного місця. Потрібно переконатися, що головоломки в грі не надто складні. Потрібно повільно впадати гравця на складні головоломки, щоб не переповнювати їх. Якщо розроблються надмірно складні головоломки, спробуйте і дизайн альтернативних шляхів або способів уникнути головоломки або обхід, якщо це необхідно, так що шлях гравців не заблокований, а гравець стає розчарованим. Потрібно пам'ятати, що не всі любить головоломки, і не кожен збирається цінувати головоломку або зможете виконати його. Не блокуйте програвач. Також добре думати про те, що вбудовані підказки або довідники, які можуть отримати вас через складні головоломки.

Ось деякі додаткові типи головоломок:

- Інформаційні головоломки. Це головоломки, в яких гравець повинен поставити відсутній частину інформації. Це може бути настільки ж просто, як постачання пароля, або як складна, як вивовано правильну послідовність чисел, які будуть знешкоджувати бомбу.
- Виключили середину. Це один з найважчих типів загадок. Він спирається на налагодження надійних причинно-наслідкових зв'язків, а потім, що вимагає від гравця визнати, що одна конкретна дія відбудеться ланцюг подій, які будуть завербуватись у бажаних заходах. Викладені з точки зору логіки; "A" завжди викликає "B", а "C" завжди викликає "D". Отже, коли гравець опиняється в ситуації, яка вимагає "D", у місці, де він має підстави вірити "B", і "C" буде пов'язано, то, сподіваюся, він виконує "A".
- Люди головоломки. Найбільш задовольняючі головоломки є ті, які залучають людей. Це тому, що, намагаючись їх вирішити, гравець неминуче дізнається більше про героїв гри, а хороші персонажі є основою хороших історій. Ці головоломки зазвичай включають людину, яка блокує прогрес, або хто має потрібну частину.
- Головоломки на час. Це важкий клас головоломки, який вимагає від гравця, який повинен визнати, він повинен прийняти дію, яка не дасть жодного миттєвого ефекту, але замість цього буде щось відбудеться в певному сенсі в майбутньому. Це стає навіть важче, якщо беруть участь більше одного місця.
- Послідовності головоломки. Це головоломки, які покладаються на виконання серії дій лише в правильному порядку. В основному вони

комедійні, як у ведмежаній рибній головоломці в керівництві Хітчікера до Галактики Стів Мерецький. Зазвичай гравець представлений простим засобом досягнення простих цілей. Коли він виконує цю дію, однак, щось раптово з'являється, щоб запобігти цьому. Ситуація потім скидає, і гравець повинен покласти щось на місце, щоб вирішити нову проблему, перш ніж знову вийти з послідовності. Це може бути досить складним.

- Судові та помилки головоломки. Гравець зіткнувся з масивом вибору, і без інформації, щоб продовжувати, він повинен спробувати одну річ, знайти це не працює, і продовжуйте намагатися нові речі, поки раптом один з них натискає.

Міні-ігри

Для багатьох ігор платформи це вимога мати серію менших ігор у них. Міні-ігри зробили справжній широкий спектр шляхів. Більшість міні-ігор дозволяють гравцеві використовувати свій характер в іншій короткій послідовності геймплея, який відрізняється від того, що вони зазвичай роблять. Деякі ігри, як Crack 2 & 3, фактично зробили міні-ігри, стали величезною частиною звичайної гри. Ідея міні-гри полягає в тому, щоб розбити темп гри для гравця, і дати їм щось нове, щоб зробити і додати певний інтерес назад у деякі області гри, яка може бути трохи нудним або повторюватися. Міні-ігри часто використовуються як нагорода за багато ігор, і відкриваються після того, як гравець робить щось хороше.

У дні 2D ігор було дуже важко зробити міні-ігри, оскільки зазвичай це вимагало нового двигуна, але в ці дні з 3D це досить легко зробити. Небезпека з виконанням міні-ігор полягає в тому, що є можливість витратити час на роботу над функціями та здібностями, які використовуються лише у грі протягом дуже короткого часу, і вимагають багато роботи - не отримуючи хорошого вибуху для долара. Для міні-ігор варто, ми повинні думати про те, як ми будемо використовувати і розвивати їх.

2.3. Метод розвязку задачі

Для розвязання задачі для створення ігри, потрібно аби не було ніяких помилок під час тестування, щоб і після також не було помилок і гравці могли спокійно грати. Основною задачею у грі має бути здатність пройти гру, тобто, щоб її могли пройти багато різних гравців, незважаючи на вік та здібності.
Основною метою завдання було створити гру платформер із елементами квесту.

Також згідно із пунктом 2.2 потреба у графіці є важливою, оскільки більшість платформерів розробленні з мультяшною, кольоровою графікою; музика має бути підібрання згідно із видом рівня. Платформи та шматки землі мають бути доступними для пригання, (мати колайдери та бути на відстані такій, щоб до них було можливо дістатися та зарпригнути).

Рис 2.2. Приклад платформеру із кольоровою графікою. Гра “Спайро”.

Квести також повинні бути доволі простими, мають не глючити, оскільки вони є одним із головних елементів продукту, та без них неможливо буде пройти гру.

2.4. Стадії та етапи розробки

Для кожного проекту має бути план по розробці, та етапи створення, редагування та додавання елементів до продукту. Тому для створення цього конкретного проекту були такі етапи:

1. Обрання движка, на якому буде створюватися гра;
2. Обрання версії движка, оскільки є функції та спеціальні пакети, які можуть бути лише на конкретній версії;
3. Створення сюжету;
4. Побудова землі та платформ на яких пригатиме герой;
5. Створити циліндр як пробного героя, додати донього колайдер, щоб він не провалювався кріз землю, написати код для руху та дій персрнажа і протестувати приклад.
6. Скачати із магазина Unity безкоштовні пакети на рівні та матеріали;
7. Імпортувати їх у Unity;
8. Видалити пробний приклад;
9. Імпортувати перший рівень через стартовий набір;

10. Пройти тренування, як обновити рівень та навчитися працювати з моделями, ефектами та блоками, які додають ефекти та дії.
11. Оновити перший рівень із новими предметами, надати квусти, можливість вільно подорожувати світом гри, додати якомога більше платформ на які пригатиме герой;
12. Створити героя, або вибрати його із вже готових моделей, які є у наборах;
13. Надати герою скриптів на рух, та прижки;
14. Створити анімацію для героя, або предмети, щоб надати їм життя;
15. Надати музику, використовуючи аудіо із наборів, або викачати самоси із інтернету;
16. Протестувати перший рівень, на можливість перемогти, та на присутнісь багів і чи усі моделі та предмети мають правильну анімацію, звуки;
17. Згідно із гайдом здублювати перший рівень і назвати його рівнем 2;
18. Переробити рівень 2 змінивши текстури, кольора, предмети, щоб рівень був іншим;
19. Зробити нове завдання;
20. Додати тематику та створити квест на удачу;
21. Змінити умови перемоги та переходу на наступний рівень;
22. Поміняти блоки перемоги на блоки поразки;
23. Створити поле для рівня 3;
24. Скачати набор полоси перешкод та імпортувати його;
25. Створити перешкоди та платформи по яким буде пригатиме герой;
26. Додати додаткових ворогів та персонажів гравців;
27. Додати предмети які потрібно збирати;
28. Протестувати полосу перешкод на можливість проходження та рухи перешкод та платформ;
29. Зробити трофей, який буде кінцевим предметом для перемоги;
30. Створити меню перемоги, поразки та початкове(для початку проходження гри);
31. Додати їх у настройки рівнів, щоб були переходи між рівнями, та через крнкретні дії, через які відкриватимуться вікна меню.
32. Зявилася нова ідея: створення бонусного рівня;
33. Скачати набор Хеллоуін для бонусного рівня та імпортувати його;
34. Модернезувати рівень, створивши додаткові моделі, та додавши блок перемоги;
35. Протестувати бонусний рівень;
36. Протестувати усю гру повністью та визначити на наявність помилок.

РОЗДІЛ 3
ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1. Опис архітектури

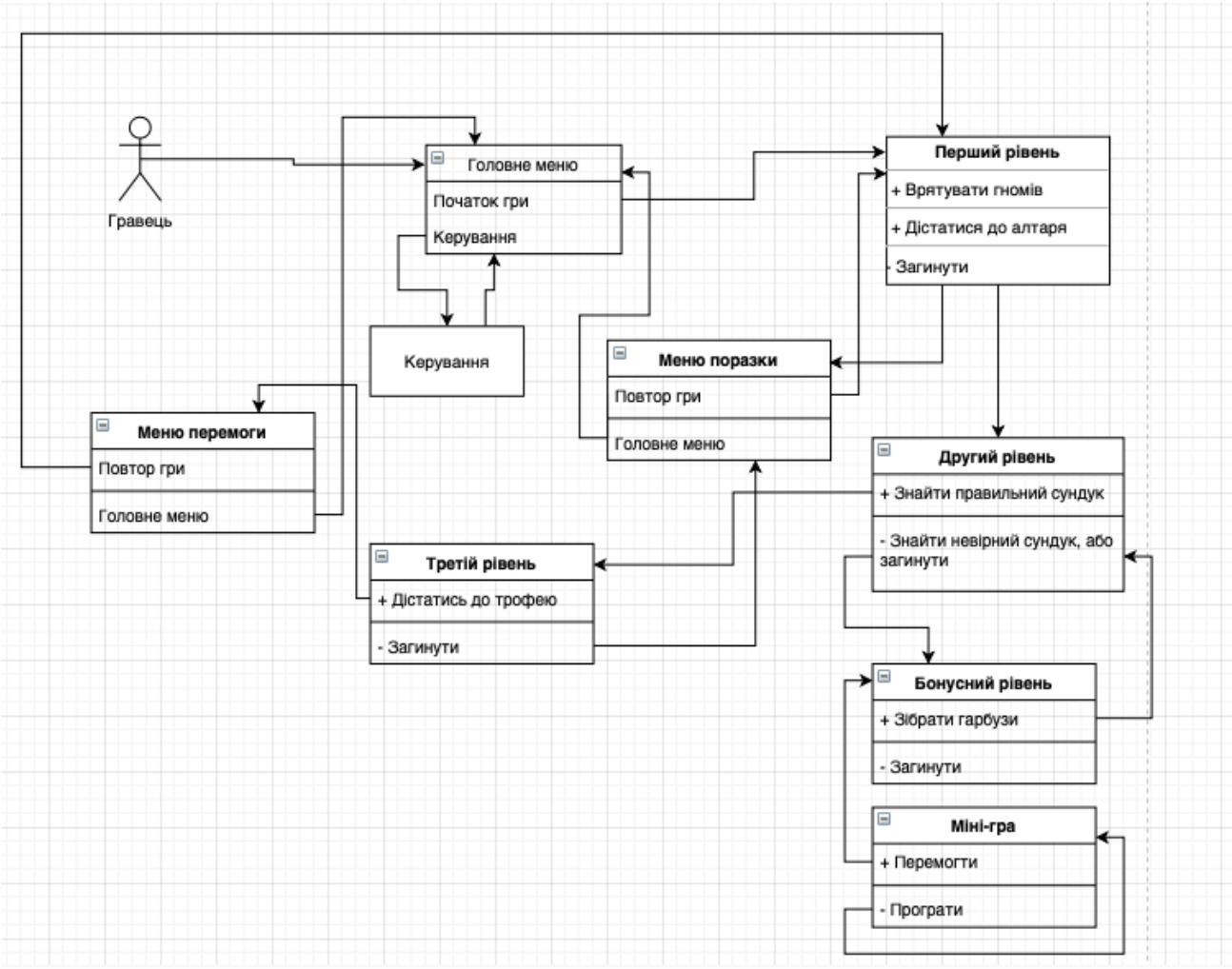


Рис 3.1. Схема взаємодії гравця із грою, та як відбувається перехід між рівнями.

Головною частиною графіки проекту є те, що більшість предметів, персонажів та матнріальм створенні за допомогою лего шматочків. Такі предмети, як гори, шматочки землі, які висять у повітрі буди створенні за допомогою звичайних 3D об’єктів: кубів, циліндрів. Додати іх можно швидко використовуючи комбінаціями: GameObject->3D Object, або в Hierarchy натиснути на плюс та обрати 3D Object.

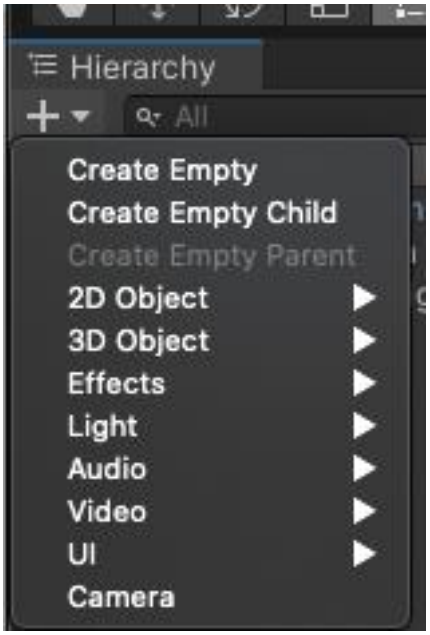


Рис 3.2. Друга комбінація створення 3D об’єкту.

Однак у стартовому наборі вже є достатня кількість гір, платформ та землі, тому створювати нові не потрібно. Платформи вже є в пакеті набору, тому її можно додавати куди завгодно а також ло них додати особистий лего блок дії.

Меню

Всього було створенно три види меню: перемоги, поразки та головне(початкове). Кожне з цих меню відповідає на відповідні функції, але нрафіка однакова. Кожне з них має дві кнопки та місце для тексту. До кнопок були додані картинки, щоб зробити графіку більш дитячою. Замінити зображення можно за допомогою пакета в наборі під нащвою “Stickers” (Наклейки). Їх можно знайти у нижній частині меню “Project”. Основні є у початковому наборі, додаткові можливо отримати придбавши пакет MgLEGO Stickers. У правому меню “Inspector” обравши SourceImage потрібно обрати стікер, який треба.

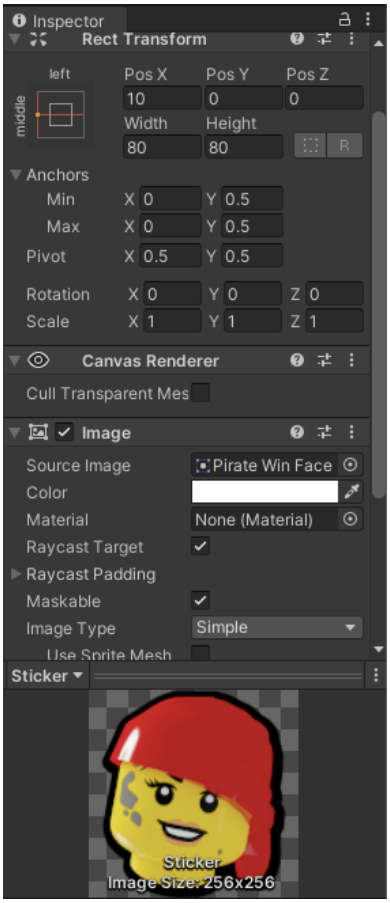


Рис 3.3. Меню Інспектор.

У головному меню гри, окрім кнопки починання гри є також кнопка “Controls”, де можно побачити як керувати персонажем. Згідно з умовами платформегу геймплей є досить простим, яу і керування:

- Кнопки руху: вперед, назад, вліво, вправо;
- Кнопка прижка, двічі клацнувши на неї двійний прижок;
- Мишка, шоб керувати камерою, та роздивлятися;
- Кнопка паузи;

Рис 3.4. Головне меню проекту “LEGO: Welcome to the wacky world”.

Рис 3.5. Інструкція керування.

Герой

Незважаючи на те, що були вже готові моделі персонажів, було створено модель нового героя який був названний як “Шукач пригод” (Adventurer). Згідно з умовами платформеру, герой є милим, хорошим хлопцем, який намагається врятувати світ. Однак по сюжету гри, герой не врятовує світ, а як вже зрозуміло з його імені, він шукає пригоди і яке місце може бути краще ніж божевільний світ

у якому дагато усього різноманітного. Там є і звірі, і пірати, і чарівники з гномами, і монстри. На кожному шагу багато пригод, де можливо знайти піратські скарби, врятувати містичних створінь та людей, а також прийняти участь у змаганні на проходження полоси перешкод і отримати величезний золотий трофей.

Рис 3.6. Модель головного героя.

До героя також була додана аніміяція на рух, стрибки та перемогу. Згідно із сбжетом герой потрапив до божевільного світу, про який він чув, заради неймовірних та цікавих пригод.

Моделі

Моделями можуть бути як і гравці так і союзники і вороги. Модель можно отрвмати або із наборів, або створити її самому використовуючи пакет блоків, які потім можно збирати разм, аби створити нову модель.

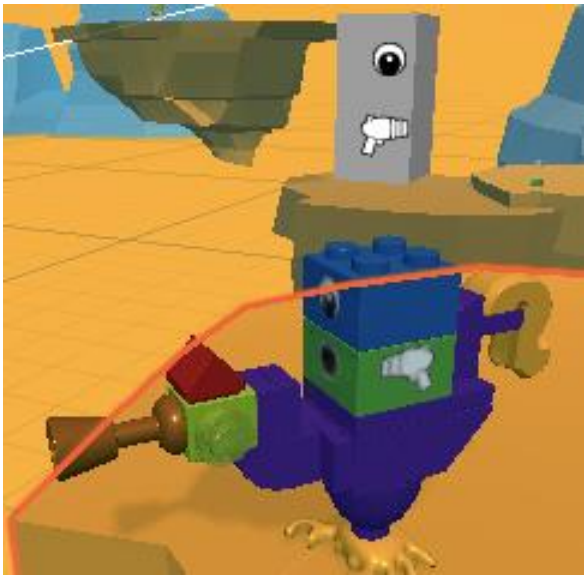


Рис 3.7. Модель додо, який стріляє і стежить за гравцем.

Для того щоб створювати нові моделі за допомогою блоків або навіть розбирати вже готову модель на блоки, потрібно на екрані обрати функція пересування не всієї моделі а одного блоку, який було обрано.

Музика

Щоб додати мкзику було створенно модель магнітофону, у якому є один із блоків дії – аудіоблок.

Аудіоблок є одним із двадцяти п’яти дійових блоків, які можно використовувати для надання моделям певних дій. Цей блок служить для звукових ефектів. Його можно додати до будь якої моделі та обрати ефект аудіо: музика, звуки перемоги, поразки, тварин. У блоці вже створенний скрипт дій:

Код 1 Скрипт на аудіо.

```
public class AudioAction : Action
   {
      [SerializeField, Tooltip("Play the audio in 3D. The player will only hear it when nearby.")]
      bool m_Spatial = false;
      [SerializeField, Tooltip("Play the audio continuously.")]
      bool m_Loop = true;
      float m_Time;
      bool m_IsPlaying;
protected void Update()
      {
         if (m_Active)
         {
```

```
            if (!m_IsPlaying)
            {
                PlayAudio(m_Loop, m_Spatial);
                m_IsPlaying = true;
            }
            if (!m_Loop)
            {
                m_Time += Time.deltaTime;
                if (m_Time >= m_Audio.length)
                {
                    m_IsPlaying = false;
                    m_Time = 0.0f;
                    m_Active = false;
                }
            }
        }
    }
```

Пікапс

На кожному рівні є свої конкретні пікапси, які потрібно зібрати. На деяких рівнях їх збирати не є важливим завданням, а на інших потрібно. Кожен із пікапів має скрипт на них, щоб вони летіли у повітрі і щоб герой міг їх зібрати і не з'являлися знову після зібрання.

Рис 3.8. Приклади пікапів.

Код 2 Частина коду на пікапи.

```
public class PickupAction : Action
   {
      public static Action<PickupAction> OnAdded;
      public static Action<PickupAction> OnCollected;
      protected override void Reset()
      {
         base.Reset();
         m_IconPath = "Assets/LEGO/Gizmos/LEGO Behaviour Icons/Pickup
Action.png";
      }
```

```
protected override void Start()
{
    base.Start();
    if (IsPlacedOnBrick())
    {
        // Add particle system.
        // Add SensoryCollider to all brick colliders.
        foreach (var brick in m_ScopedBricks)
        {
            foreach (var part in brick.parts)
            {
                foreach (var collider in part.colliders)
                {
                    var sensoryCollider = LEGOBehaviourCollider.Add<SensoryCollider>(collider, m_ScopedBricks, 0.64f);
                    SetupSensoryCollider(sensoryCollider);

                    // Make the original collider a trigger.
                    collider.isTrigger = true;
                }
            }
        }
```

public static Action<PickupAction> OnAdded; - дія на додання.

public static Action<PickupAction> OnCollected; - дія на здирання пікапів.

collider.isTrigger = true; - колайде стає трігером, тобто при доторканні, відбувається дія.

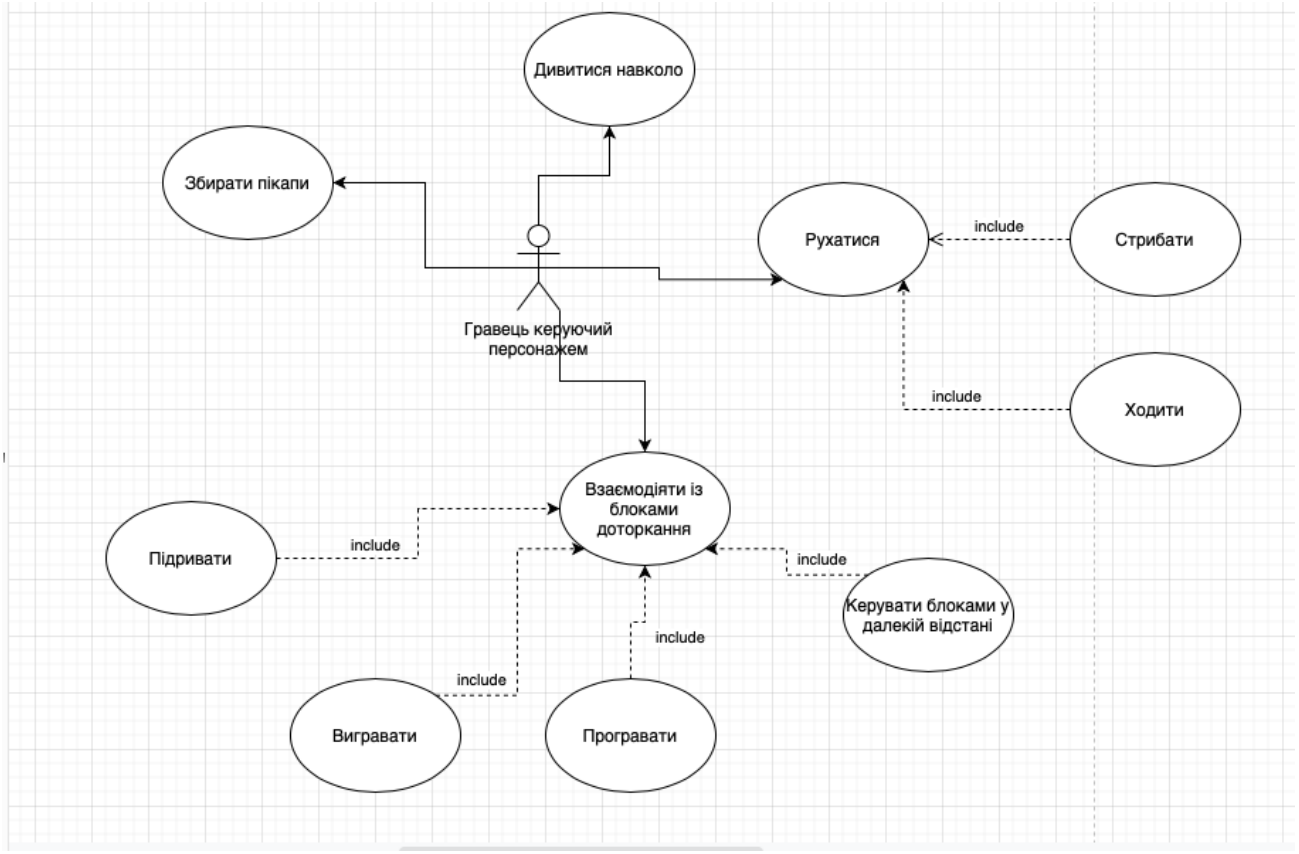


Рис 3.9. Схема дій персонажа, які доступні у грі.

Блоки дій

Усього існує двадцять п’ять таких блоків. Коржен із них відповідає за певну функцію, такі як:

- Перемога;
- Поразка;
- Підібрати (Пікап);
- Підібрати (Персонаж);
- Торкнутися;
- Живий (Друг);
- Аудіо;
- Контроль (для транспортів);
- Ліфт;
- Взрив;
- Смерть (Ворог);
- Висіти у повітрі;
- Натиснути на кнопку;
- Стежити;
- Йти;

- Крутитися;
- Стріляти;
- Говорити;
- Платформено рухатися;
- Зявлятися;
- Таймер;
- Наугад;
- Поблизу тригер;
- Лічильник;
- Лічильник трігер.

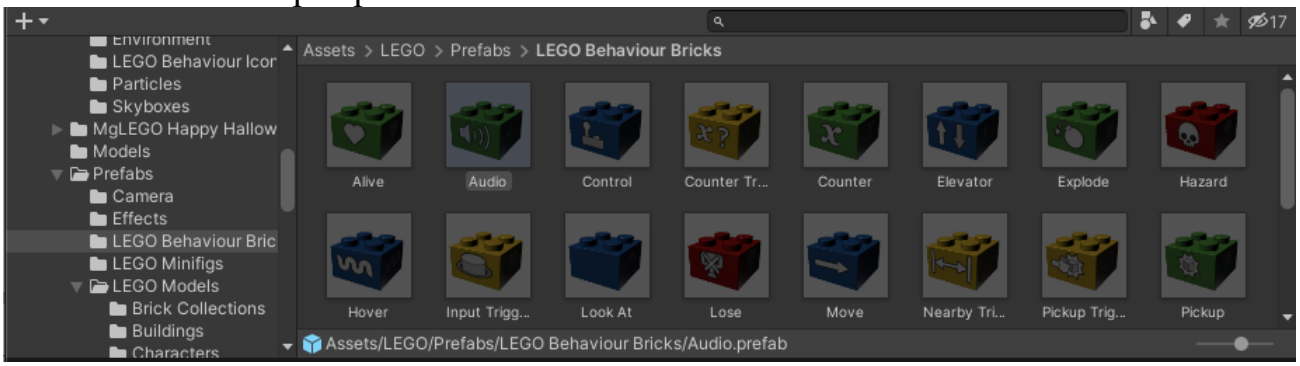


Рис 3.10. Блоки дій.

Використовуючи кожен із цих блоків надає дію моделі, або предмету. Додатково є свмі скрипти, які можно додавати до блоків, аби виповнялися декілька дій разом. При цьому є одна проблема – не можливо змінити послідовність дій, якщо це не зробити із самого початку.

Код 3: Скрипт смерті впавши.

```
using Unity.LEGO.Game;
using UnityEngine;
namespace Unity.LEGO.Gameplay
{
   public class GroundHazard : MonoBehaviour
   {
      void OnTriggerEnter(Collider other)
      {
         if(other.gameObject.CompareTag("Player"))
         {
            GameOverEvent evt = Events.GameOverEvent;
            evt.Win = false;
            EventManager.Broadcast(evt);
         }
      }
```

```
    }
}
```

if(other.gameObject.CompareTag("Player")) – шукає об’єкт Гравця, коли персонаж зіткнеться із колайдером.
void OnTriggerEnter(Collider other) – увімкнути колайдер.
GameOverEvent evt = Events.GameOverEvent; - гра закінчена, поразка.
EventManager.Broadcast(evt); - відображати, до певного моменту падіння.

Міні-ігри
Додаткову роля для платформера також грають міні-ігри, тому аби задовольнити цю потребу було створенно три міні-ігри:

1. Зібрати фрукти. Ціль гри: у героя стріляють, потрібно ховатися за кактусами, дати ворогу їх знищити, щоб потім забрати фрукти, які виваляться з них;

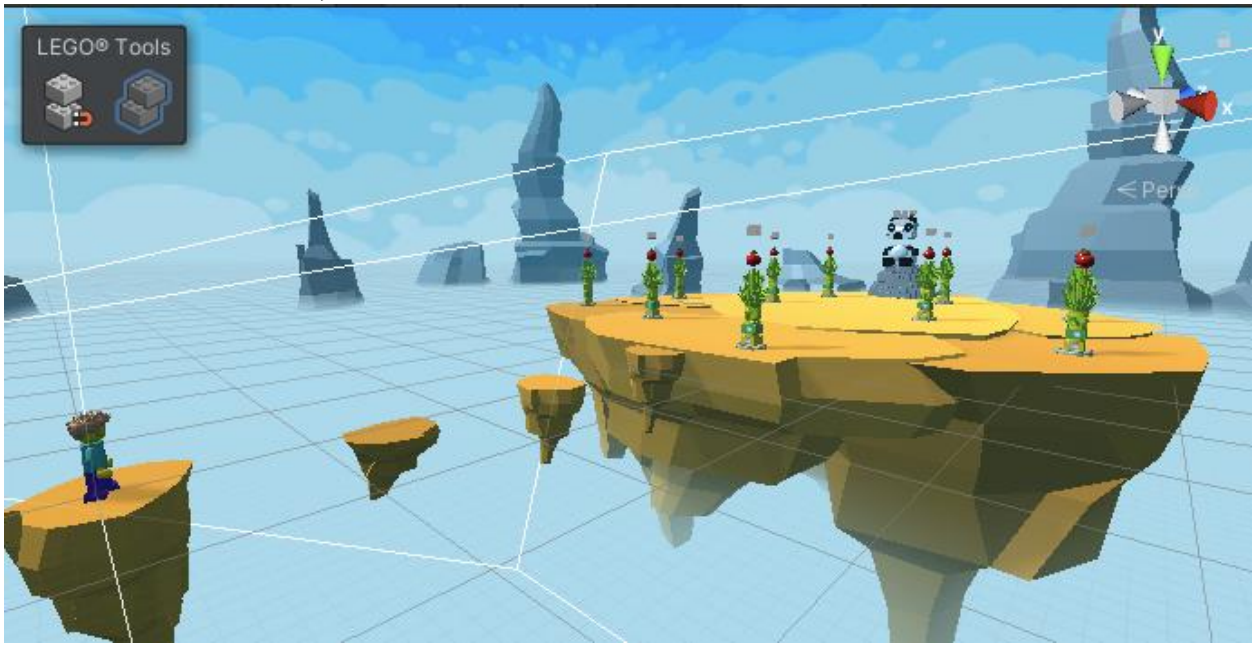


Рис 3.11. Міні-ігра 1.

2. Залишитися живим. Гармата стріляє у платформи по яких стрибає герой. Гра закінчиться, якщо герой звалиться, або залишиться стояти на одній залишеній платформі.

Код 4. Скрипт на стрільбу:

```
namespace Unity.LEGO.Behaviours.Actions
{
    public class ShootAction : RepeatableAction
    {
        [SerializeField, Tooltip("The projectile to launch.")]
```

```
        GameObject m_Projectile = null;
        [SerializeField, Range(1, 100), Tooltip("The velocity of the projectiles.")]
        float m_Velocity = 25f;
        [SerializeField, Range(0, 100), Tooltip("The accuracy in percent.")]
        int m_Accuracy = 90;
protected override void Reset()
        {
            base.Reset();
            m_Scope = Scope.Brick;
            m_IconPath = "Assets/LEGO/Gizmos/LEGO Behaviour Icons/Shoot Action.png";
        }
void Fire()
        {
            if (m_Projectile)
            {
                var go = Instantiate(m_Projectile);
                go.transform.position = transform.TransformPoint(m_ScopedPivotOffset);
                var accuracyToDegrees = 90.0f - 90.0f * m_Accuracy / 100.0f;
                var randomSpread = Random.insideUnitCircle * Mathf.Tan(accuracyToDegrees * Mathf.Deg2Rad * 0.5f);
                go.transform.rotation = transform.rotation * Quaternion.LookRotation(Vector3.forward + Vector3.right * randomSpread.x + Vector3.up * randomSpread.y);
                var projectile = go.GetComponent<Projectile>();
                if (projectile)
                {
                    projectile.Init(m_ConnectedBricks, m_Velocity, m_UseGravity, m_Lifetime);
                }
                PlayAudio();
            }
        }
```

[SerializeField, Tooltip("The projectile to launch.")] – додати у вікно предмет, яким стрілятимуть.

[SerializeField, Range(1, 100), Tooltip("The velocity of the projectiles.")] – швидкість стрільби(чим більше, тим швидше вилітають заряди).

var projectile = go.GetComponent<Projectile>(); - обрати заряд, які знаходяться у компоненті Projectile.

if (projectile) {projectile.Init(m_ConnectedBricks, m_Velocity, m_UseGravity, m_Lifetime);} PlayAudio(); - коли заряд вилітає, іде аудіо.

3. Зібрати кристали, аби знищити машину смерті. Порібно зібрати пікап кристал, а потім дістатися до блока перемоги.

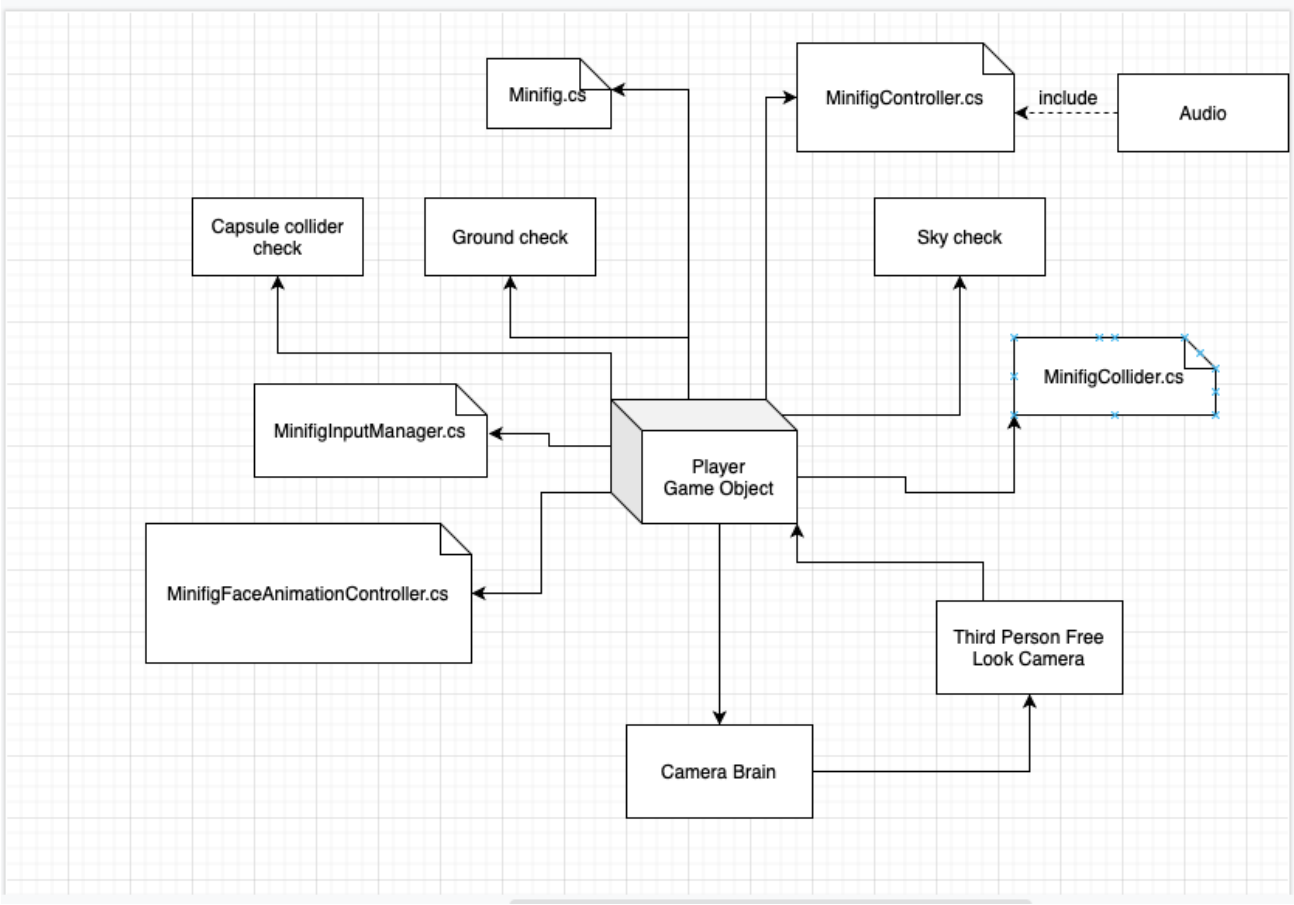


Рис 3.12. Схема взаємодії об’єкту персонажу, з камерою та скриптами.

3.2. Тестування програмного продукту

Під час проведення тестування було знайденно декілька помилок, через які відбувалися проблеми у грі.

Перший рівень. Був спроектований без помилок, але були дві проблеми:

- Аби дістатися до панелі яка відкриває клітку у якій сидять гноми та чарівники, платформа із блокол ліфта, мала дуже коротку довжину піднімання;
- Також блок перемоги у клітці не працював правильно, але це було пофіксено..

А шлях до алтаря залишився без змін, тому проблем ніяких.

Другий рівень. Скриньки із скарбами працювали, як потрібно, одна із блоком перемоги направляє потім до рівня три.

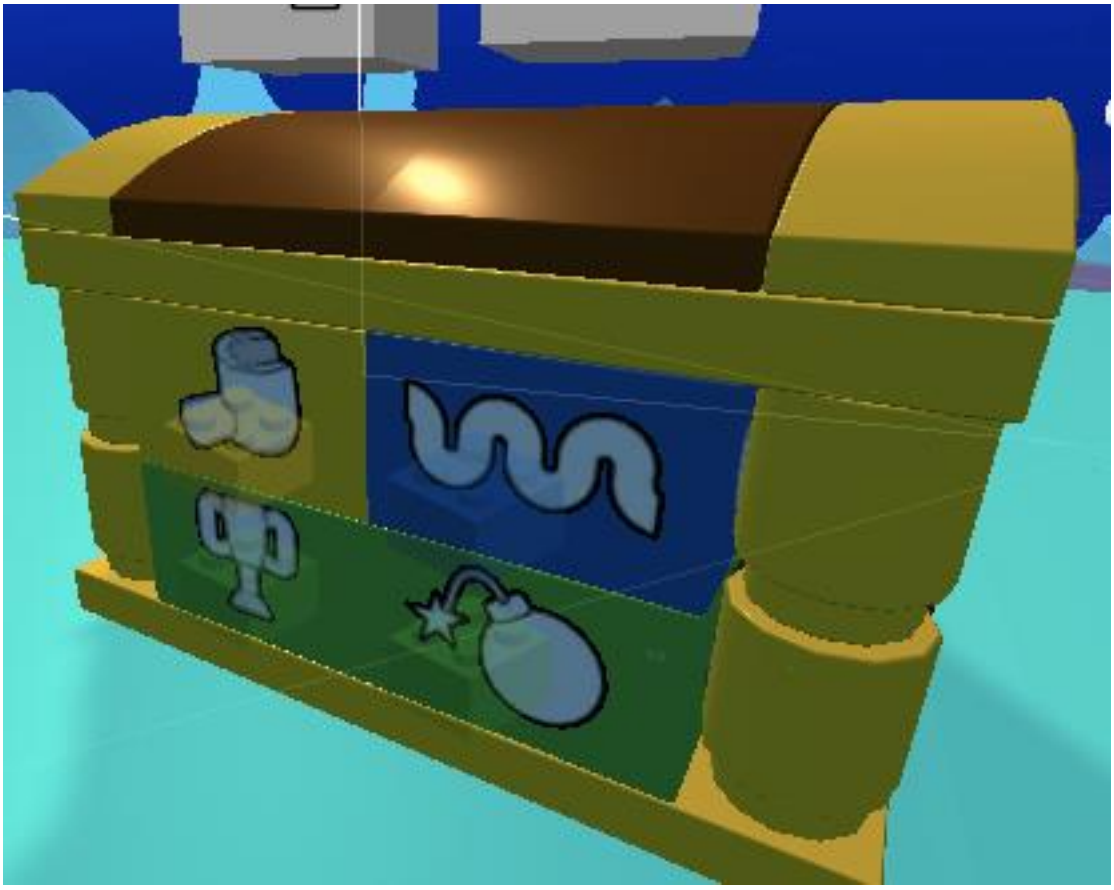


Рис 3.13. Вірний сундук.

А три інших із блоками поразки направляють на бонусний рівень.

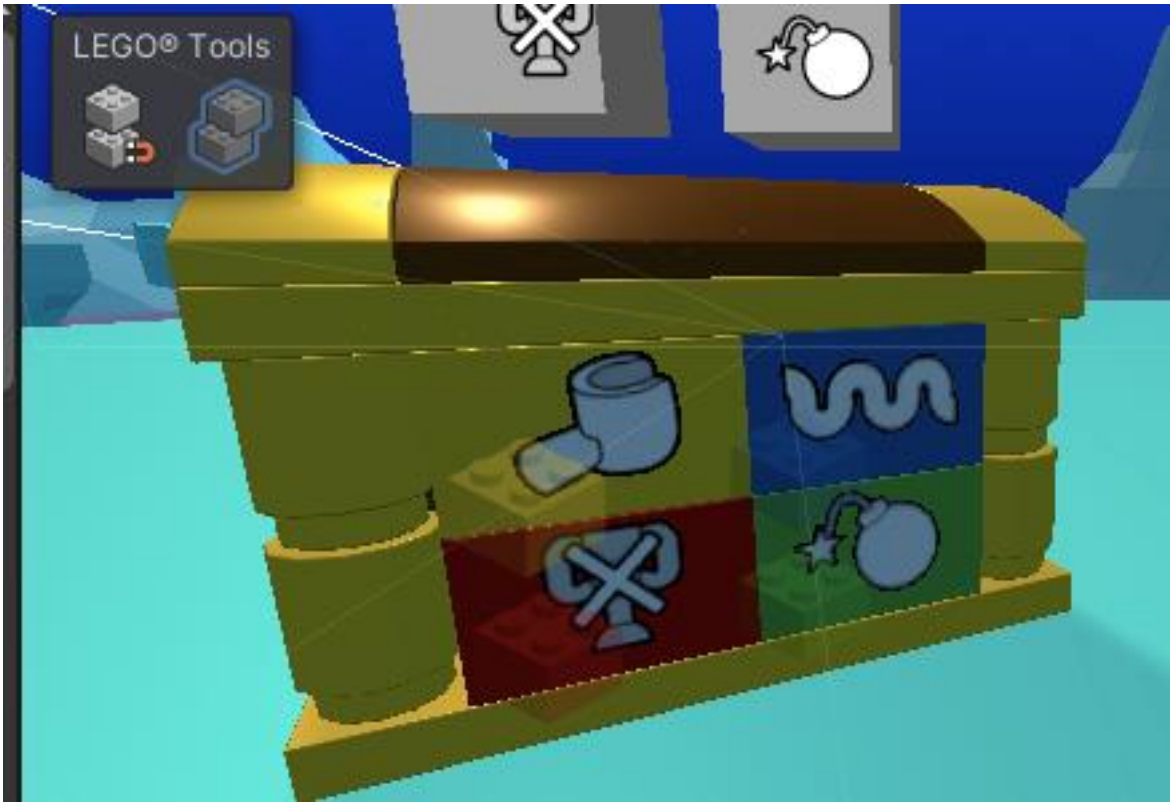


Рис 3.14. Не вірний сундук.

Бонусний рівень. Не було ніяких проблем, рівень можливо пройти, можливо зібрати усі пікапи, щоб перемогти.

Третій рівень. Під час тестування, були проблеми із платформами, оскільки було складно запригнути на них, а також мости, які крутяться застрягали біля платформ, але це усе було пофіксано, тепер можно дістатися до трофею.

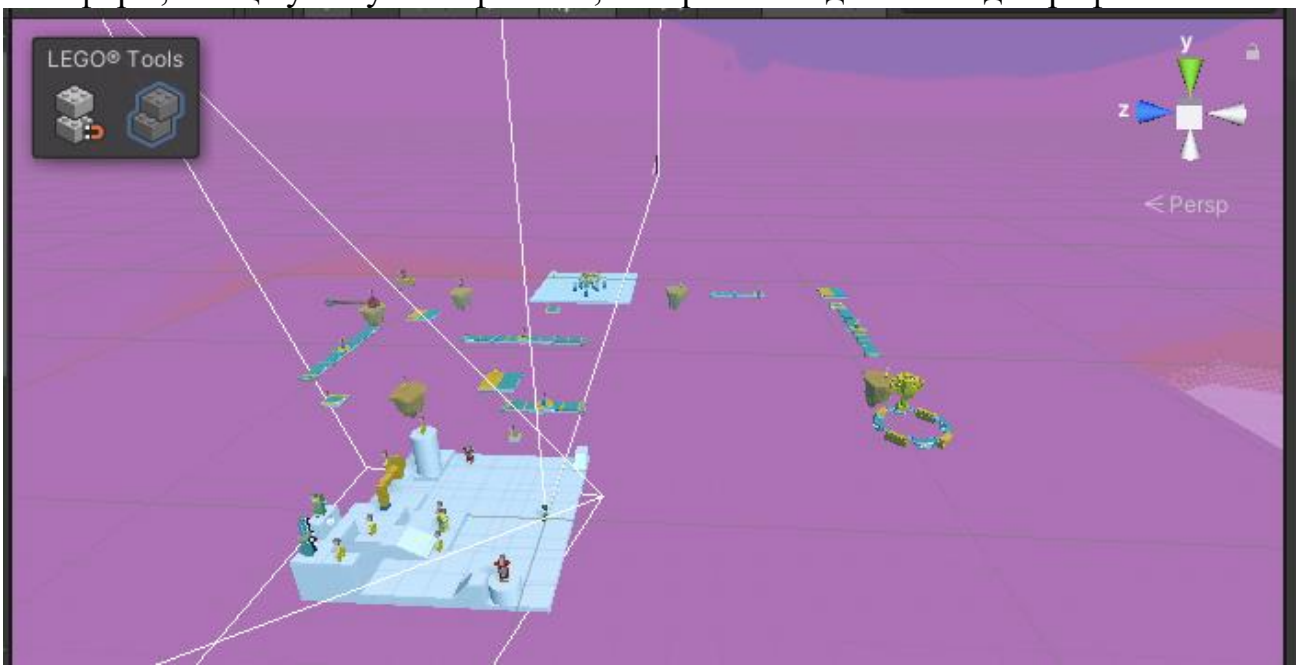


Рис 3.15. Показ третього рівня.

3.3. Результати програмного продукту

За результатами програмний продукт відповідає поставленним умовам, які були наданні у темі, а також він підходить умовам жанру гри та графіки. Після проведення тестування, гру було пройденно і кожен рівень дав свій результат:

- Перший рівень: можливість пройти його дуже легко, повсюди є платформи на які можно легко запригнути, без ніяких складнощів, ворогів у цьому рівні замало, тому немає складних перешкод, був доданий ще додатковий квест врятувати чарівників, яких захопили пірати, згідно із сюжетом, усього їх три на карті і щоб пройти рівень потрібно їх усіх врятувати, а потім дістатися алтаря із блоком “Перемога” і герой переходить до рівня два;
- Другий рівень: воргів досить багато та перешкод, тому рівень є складнішим, усе працює ідально, час на стрільбу гармат було зменшено, щоб було більше шансів пройти рівень. Підказка для гравця працює. Сундуки із скарбами працюють згідно блокам; правильний сундук відправляє на рівень три, невірний на бонусний рівень “Підземний світ”;
- Бонусний рівень: працює без помилок. Після знайдення усіх гарбузів, герой дістався до вампіра та перемагає, повертаючись назад до рівня два. Поразка, попадає на рівень міні-ігри. Після проходження міні-ігри, повернення назад до другого рівня. Поразка – залишається на тому ж міні рівні, оскільки усі міні-ігри є доволі простими, тому давати перехід на іншу не має сенсу;
- Рівні міні-ігор: усі міні-ішри працюють без проблем. Кожну із них можливо легко пройти, без ніяких складнощів. Блоки перемоги працюють та пересилають на мню перемоги;
- Третій рівень: полоса перешкод працює чудово. Жодна з перешкод не зупиняється. Платформи рухаються, не стикаючись одна з іншою, перешкоди також. Можливість зібрати предмети легка. Виповнення квесту можливе для виповнення. Дістатися до фінального трофею можливо, аби закінчити ігру.

Отже за усіма результатами ігра працює без помилок, та її можливо пройти та насолоджуватися не тільки геймплеєм, а й кольоровою графікою, яка працює без ніяких багів, а також насолоджуватися музикою, яка грає на кожному рівні. Музика була підібрання для теми кожного рівня, аби додати особисту тематику.

Рис 3.16. Відображення повної перемоги та проходження гри.

ВИСНОВКИ

Під час виконання даної випускної кваліфікованної бакалаврської роботи було розробленно, створенно та безкоштовно опубліковано гру жанру платформера з елементами квесту, використавши для її створення кросалатформенний движок Unity. Ця гра є завершальним продуктом, але її можна покращити, для створення повноцінного унікального продукту, який потім може стати популярним.

У даній роботі були розглянуті та проаналізовані вже існуючі продукти, пояснивши їх унікальність, проблеми які мали гравці під час грання, які були баги і було створено продукт, який не матиме багів і буде легкий для гравців будь якого віку. Під час розробки движок Unity показав себе як зручний, простий в освоєнні і професійно виконаний інструмент для створення ігор. Unity – ігровий рушій, зібравший в собі фізичний, графічний рушій, великий набір інструментів та широкий функціонал. Гарний баланс міл кількістю написаного коду та використання вбудованого інструментарію для налаштування елементів гри. Зручна візуалізація продукту та чудові можливості відладки як коду так і поведінки гри в цілому. Великі можливості для повторного використання коду та вбудована оптимізація графіки, фізики. Unity виявився найпотужнішим засобом для розробки гри серед розглянутих, використаний для аналізу прототип виглядав надто легким для використання представленного функціоналу. Найкраще Unity покаже себе в розробці великих та середніх проектів командою чи поодиноким розробником.

Було створенно коди на скрипти для дійових блоків, аби при з’єднанні їх у грі скрипти взаємодіяли між собою, що покращило та полегшало розробці продукту, а також аби виповняти багато дій з однією моделю, при цьому не створюючи бази даних, як у багатьох інших продуктах. Гра була повністю створенна таким чином, що скрипти можливо об’єднати до однієї моделі, яка зможе витримати кілька скриптів в один і теж самий час, при цьому дає можливість обрати який із скриптів буде виконуватися першим і так у послідовності.

Було створено гру з архітектурою виду LEGO, що є доволі дитячою, оскільки LEGO ігри створенні більше для споживачів молодого та юного віку, але за умовами створення платформерів, такий вид архітектури підійде, оскільки більшість платформерів розробленні із кольоровою архітектурою та графікою. Також використовуючи такий вид архітектури, розробник починає розвивати креативність та уяву для створення нових моделей. За допомогою функції розкладання та збирання блоків LEGO, можливо створювати багато різних предметів, моделей створінь і також створити мініфігуру персонажа, так само як і у реальному житті із конструкторами LEGO іграшок.

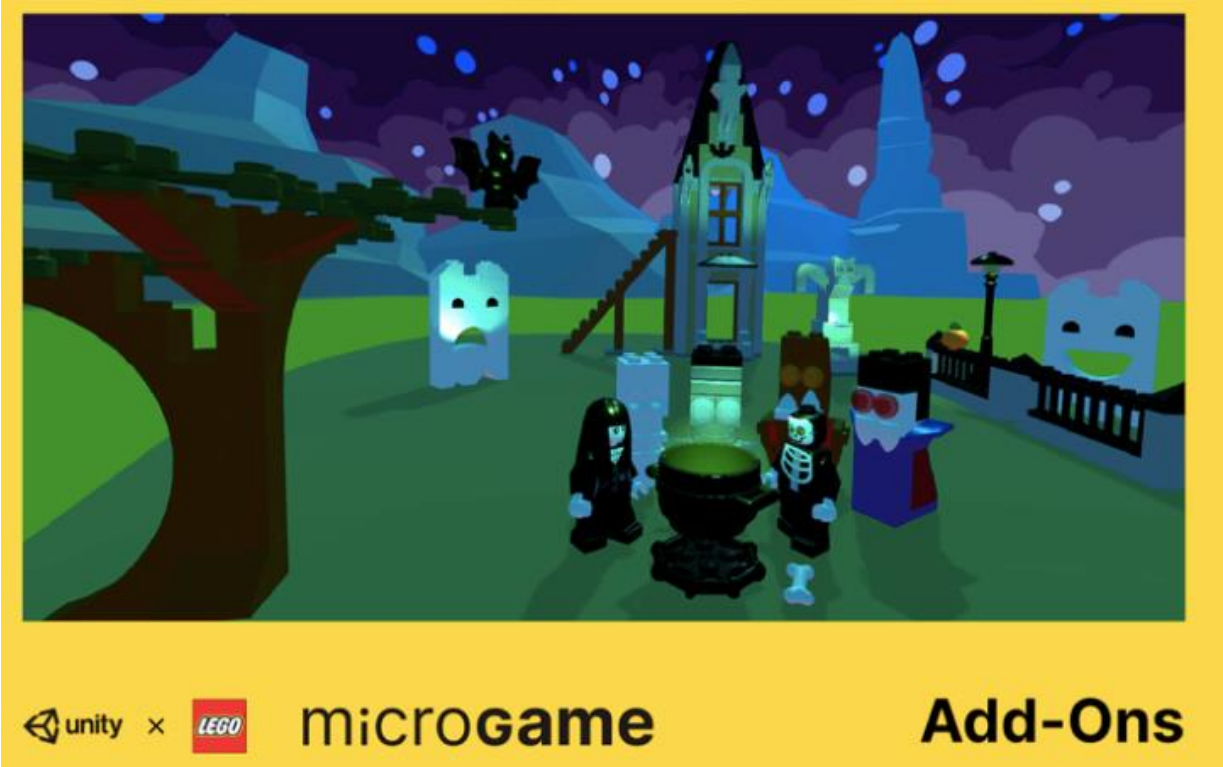
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Базова ігрова архітектура: https://www.studytonight.com/3d-game-engineering-with-unity/game-development-architecture.
2. Відео тренування, як створити приклад лего гри: https://www.youtube.com/watch?v=LwOU2JtBQD4&t=2165s.
3. Гайд для вибору движка: https://gamedevacademy.org/unity-vs-unreal/.
4. Дизайн та основа платформерів: https://www.linkedin.com/pulse/platformer-character-games-design-troy-dunniway.
5. Детальна інформація про Unity, плюси та мінуси: https://novicedock.com/learn/computer-science/unity-game-engine/pros-and-cons.
6. Інструкції в Unity, як спворити ЛЕГО ігру: https://learn.unity.com/tutorial/lego-mod-upgrade-your-levels.
7. Інформація про 10 найкращих движків, які використовують розробники ігор у своїй карьєрі: https://www.gamedesigning.org/career/video-game-engines/.
8. Обранний сайт для створення онлайн блок-схем: https://app.diagrams.net.
9. Посібник для роботи з Unity: https://docs.unity3d.com/Manual/AssetWorkflow.html.
10. Сайт де можна знайти додатки для ігри на Unity: https://assetstore.unity.com.
11. Список 15ти кращих сайтів для створення онлайн блок-схем: https://kj.media/kissel/15-onlajn-servisov-dlya-sozdaniya-blok-shem/.
12. Список ігрових движків, які існують: https://en.wikipedia.org/wiki/List_of_game_engines.
13. Список ігр створених на Unit, аби розглянути різні архітектури: https://itch.io/games/tag-unity.
14. Стадії розробки ігор: https://www.g2.com/articles/stages-of-game-development.
15. Unity проти Unreal Engine: https://hackr.io/blog/unity-vs-unreal-engine.
16. Youtube канал, де можно навчитися базово програмувати скрипти на ігри для Unity: https://www.youtube.com/channel/UCvuY904el7JvBlPbdqbfguw.

ДОДАТКИ

Додаток А

Набор в Unitystore на створення рівня у виді Хеллоуін



Додаток Б

Скрипт коду на поразку

```
namespace Unity.LEGO.Behaviours.Actions
{
   public class LoseAction : ObjectiveAction
   {
      public override ObjectiveConfiguration
GetDefaultObjectiveConfiguration(Trigger trigger)
      {
         ObjectiveConfiguration result = new ObjectiveConfiguration();
         if (trigger)
         {
            var triggerType = trigger.GetType();
            if (triggerType == typeof(PickupTrigger))
            {
               result.Title = "Don't Pickup the Pickups!";
               result.Description = "Don't do it!";
               result.ProgressType = ObjectiveProgressType.Amount;
            }
            else if (triggerType == typeof(TouchTrigger))
            {
               result.Title = "Don't Touch the Object";
               result.Description = "You can't touch this!";
            }
            else if (triggerType == typeof(NearbyTrigger))
            {
               result.Title = "Avoid the Object";
               result.Description = "Avoid it!";
            }
            else if (triggerType == typeof(TimerTrigger))
            {
               result.Title = "Finish Before the Time is Up";
               result.Description = "Hurry up!";
               result.ProgressType = ObjectiveProgressType.Time;
            }
            else if (triggerType == typeof(RandomTrigger))
            {
               result.Title = "Finish Before a Random Time is Up";
               result.Description = "Hurry up!";
            }
```

```
            else if (triggerType == typeof(InputTrigger))
            {
                result.Title = "Don't Press the Button";
                result.Description = "Don't push it";
            }
            else if (triggerType == typeof(CounterTrigger))
            {
                result.Title = "Don't Complete the Objective";
                result.Description = "Don't do it!";
            }
            else
            {
                result.Title = "Don't Complete the Objective";
                result.Description = "Don't do it!";
            }
        }
        else
        {
            result.Title = "Didn't Stand a Chance!";
            result.Description = "Connect a Trigger Brick to the Lose Brick to make your game easier.";
        }
        result.Lose = true;
        return result;
    }
    protected override void Reset()
    {
        base.Reset();
        m_FlashColour = MouldingColour.GetColour(MouldingColour.Id.BrightRed) * 2.0f;
        m_IconPath = "Assets/LEGO/Gizmos/LEGO Behaviour Icons/Lose Action.png";
    }
  }
}
```

Додаток В

Скрипт коду на колайдер комбінації блоків, для кострукції нових моделей

```
namespace Unity.LEGO.Behaviours
{
   public class BrickColliderCombiner : MonoBehaviour
   {
      static Dictionary<Part, List<Collider>> s_OriginalColliders = new Dictionary<Part,
List<Collider>>();
      void Awake()
      {
         var bricks = FindObjectsOfType<Brick>();
         var behaviours = FindObjectsOfType<LEGOBehaviour>();
         var behaviourScopes = new Dictionary<LEGOBehaviour, HashSet<Brick>>();
         foreach (var brick in bricks)
         {
            var combinableBehaviourOnBrick = false;
            foreach (var behaviour in behaviours)
            {
               var behaviourType = behaviour.GetType();
               if (behaviourType == typeof(HazardAction) || behaviourType == typeof(PickupAction)
|| behaviourType == typeof(TouchTrigger) || behaviour is MovementAction)
               {
                  if (!behaviourScopes.ContainsKey(behaviour))
                  {
                     behaviourScopes.Add(behaviour, behaviour.GetScopedBricks());
                  }
                  if (behaviourScopes[behaviour].Contains(brick))
                  {
                     combinableBehaviourOnBrick = true;
                     break;
                  }
               }
            }
            if (combinableBehaviourOnBrick)
            {
               // Combine all colliders in part into one box collider.
               foreach (var part in brick.parts)
               {
                  if (part.colliders.colliders.Count > 0)
                  {
                     var collidersParent = part.transform.Find("Colliders");
                     var min = new Vector3(Mathf.Infinity, Mathf.Infinity, Mathf.Infinity);
                     var max = new Vector3(Mathf.NegativeInfinity, Mathf.NegativeInfinity,
Mathf.NegativeInfinity);
```

```
                    foreach (var collider in part.colliders)
                    {
                        var colliderType = collider.GetType();
                        if (colliderType == typeof(BoxCollider))
                        {
                            var boxCollider = (BoxCollider)collider;
                            var c0 =
part.transform.InverseTransformPoint(boxCollider.transform.TransformPoint(boxCollider.center +
Vector3.Scale(new Vector3(-0.5f, -0.5f, -0.5f), boxCollider.size)));
                            var c1 =
part.transform.InverseTransformPoint(boxCollider.transform.TransformPoint(boxCollider.center +
Vector3.Scale(new Vector3(-0.5f, -0.5f, 0.5f), boxCollider.size)));
                            var c2 =
part.transform.InverseTransformPoint(boxCollider.transform.TransformPoint(boxCollider.center +
Vector3.Scale(new Vector3(-0.5f, 0.5f, -0.5f), boxCollider.size)));
                            var c3 =
part.transform.InverseTransformPoint(boxCollider.transform.TransformPoint(boxCollider.center +
Vector3.Scale(new Vector3(-0.5f, 0.5f, 0.5f), boxCollider.size)));
                            var c4 =
part.transform.InverseTransformPoint(boxCollider.transform.TransformPoint(boxCollider.center +
Vector3.Scale(new Vector3(0.5f, -0.5f, -0.5f), boxCollider.size)));
                            var c5 =
part.transform.InverseTransformPoint(boxCollider.transform.TransformPoint(boxCollider.center +
Vector3.Scale(new Vector3(0.5f, -0.5f, 0.5f), boxCollider.size)));
                            var c6 =
part.transform.InverseTransformPoint(boxCollider.transform.TransformPoint(boxCollider.center +
Vector3.Scale(new Vector3(0.5f, 0.5f, -0.5f), boxCollider.size)));
                            var c7 =
part.transform.InverseTransformPoint(boxCollider.transform.TransformPoint(boxCollider.center +
Vector3.Scale(new Vector3(0.5f, 0.5f, 0.5f), boxCollider.size)));
                            min = Vector3.Min(min, c0);
                            min = Vector3.Min(min, c1);
                            min = Vector3.Min(min, c2);
                            min = Vector3.Min(min, c3);
                            min = Vector3.Min(min, c4);
                            min = Vector3.Min(min, c5);
                            min = Vector3.Min(min, c6);
                            min = Vector3.Min(min, c7);
                            max = Vector3.Max(max, c0);
                            max = Vector3.Max(max, c1);
                            max = Vector3.Max(max, c2);
                            max = Vector3.Max(max, c3);
                            max = Vector3.Max(max, c4);
                            max = Vector3.Max(max, c5);
                            max = Vector3.Max(max, c6);
                            max = Vector3.Max(max, c7);
                        }
                        else if (colliderType == typeof(SphereCollider))
                        {
```

```
                        var sphereCollider = (SphereCollider)collider;
                        var c =
part.transform.InverseTransformPoint(sphereCollider.transform.TransformPoint(sphereCollider.cent
er));
                        min = Vector3.Min(min, c - Vector3.one * sphereCollider.radius);
                        max = Vector3.Min(max, c + Vector3.one * sphereCollider.radius);
                    }
                    collider.gameObject.SetActive(false);
                }

                var combinedCollisionBox = new GameObject("Collision Box");
                combinedCollisionBox.transform.parent = collidersParent;
                combinedCollisionBox.transform.localPosition = Vector3.zero;
                combinedCollisionBox.transform.localRotation = Quaternion.identity;
                var combinedCollider =
combinedCollisionBox.gameObject.AddComponent<BoxCollider>();
                combinedCollider.center = (min + max) * 0.5f;
                combinedCollider.size = max - min;
                s_OriginalColliders[part] = part.colliders.colliders;
                part.colliders.colliders = new List<Collider>();
                part.colliders.colliders.Add(combinedCollider);
            }
        }
    }
    public static void RestoreOriginalColliders(Part part)
    {
        if (s_OriginalColliders.ContainsKey(part))
        {
            foreach (var collider in part.colliders)
            {
                Destroy(collider.gameObject);
            }

            part.colliders.colliders = s_OriginalColliders[part];

            foreach (var collider in part.colliders)
            {
                collider.gameObject.SetActive(true);
            }
        }
    }
}
```

Додаток Г

Скрипт коду на знищення мініфігури (смерті), коли герой гине

```
namespace Unity.LEGO.Minifig
{

   public static class MinifigExploder
   {
      private class EnableCollider : MonoBehaviour
      {
         private Collider theCollider;
         private int fixedUpdateDelay = 20;

         void Start()
         {
            theCollider = GetComponent<Collider>();
         }

         void FixedUpdate()
         {
            if (fixedUpdateDelay <= 0)
            {
               theCollider.isTrigger = false;
               Destroy(this);
            }

            fixedUpdateDelay--;
         }
      }

      private class BlinkAndDestroy : MonoBehaviour
      {
         public float timeLeft = 4.0f;

         private float blinkPeriod = 0.8f;
         private float blinkFrequency = 0.1f;

         private Renderer theRenderer;
         private float timeBlink;

         void Start()
         {
            theRenderer = GetComponent<Renderer>();
            timeLeft += Random.Range(-0.3f, 0.3f);
         }

         void Update()
```

```
        {
            timeLeft -= Time.deltaTime;

            if (timeLeft <= blinkPeriod)
            {
                if (timeBlink <= 0.0f)
                {
                    timeBlink += blinkFrequency;
                    theRenderer.enabled = !theRenderer.enabled;
                }

                timeBlink -= Time.deltaTime;
            }

            if (timeLeft <= 0.0f)
            {
                Destroy(gameObject);
            }
        }
    }

    public static void Explode(Minifig minifig, Transform leftArmTip, Transform rightArmTip,
Transform leftLegTip, Transform rightLegTip, Transform head, Vector3 speed, float angularSpeed)
    {
        const float spreadForce = 6.0f;
        const float removeDelay = 3.0f;

        // FIXME Add speed, angularSpeed and spreadForce to rigid body.

        // Use wrist, arm and armUp transforms.
        ExplodeLimb(minifig.transform, "Left arm", minifig.GetLeftArm(),
leftArmTip.parent.parent.parent, leftArmTip.parent.parent, leftArmTip, 0.3f, speed, angularSpeed,
spreadForce, removeDelay);
        ExplodeLimb(minifig.transform, "Right arm", minifig.GetRightArm(),
rightArmTip.parent.parent.parent, rightArmTip.parent.parent, rightArmTip, 0.3f, speed,
angularSpeed, spreadForce, removeDelay);

        var leftHandTransforms = new List<Transform> { leftArmTip, leftArmTip.parent };
        var rightHandTransforms = new List<Transform> { rightArmTip, rightArmTip.parent };
        var leftHandColliderCenters = new List<Vector3> { Vector3.forward * 0.2f, Vector3.zero };
        var rightHandColliderCenters = new List<Vector3> { Vector3.back * 0.2f, Vector3.zero };
        var handColliderSizes = new List<Vector3> { new Vector3(0.4f, 0.4f, 0.4f), new
Vector3(0.2f, 0.4f, 0.2f) };
        var handColliderInitialTriggers = new List<bool> { false, true };
        ExplodeWithTransforms(minifig.transform, "Left hand", new List<Transform>
{ minifig.GetLeftHand() }, leftHandTransforms, leftHandColliderCenters, handColliderSizes,
handColliderInitialTriggers, speed, angularSpeed, spreadForce, removeDelay);
```

```
        ExplodeWithTransforms(minifig.transform, "Right hand", new List<Transform>
{ minifig.GetRightHand() }, rightHandTransforms, rightHandColliderCenters, handColliderSizes,
handColliderInitialTriggers, speed, angularSpeed, spreadForce, removeDelay);

        // Use foot, leg and legUp transforms.
        ExplodeLimb(minifig.transform, "Left leg", minifig.GetLeftLeg(), leftLegTip.parent.parent,
leftLegTip.parent, leftLegTip, 0.5f, speed, angularSpeed, spreadForce, removeDelay);
        ExplodeLimb(minifig.transform, "Right leg", minifig.GetRightLeg(),
rightLegTip.parent.parent, rightLegTip.parent, rightLegTip, 0.5f, speed, angularSpeed, spreadForce,
removeDelay);

        // Use hip transform.
        var hipTransforms = new List<Transform> { leftLegTip.parent.parent.parent,
leftLegTip.parent.parent.parent, leftLegTip.parent.parent.parent, leftLegTip.parent.parent.parent };
        var hipColliderCenters = new List<Vector3> { Vector3.up * 0.4f, Vector3.zero, new
Vector3(0.0f, 0.7f, 0.4f), new Vector3(0.0f, 0.7f, -0.4f) };
        var hipColliderSizes = new List<Vector3> { new Vector3(0.8f, 0.2f, 1.6f), new
Vector3(0.7f, 0.7f, 0.2f), new Vector3(0.4f, 0.4f, 0.4f), new Vector3(0.4f, 0.4f, 0.4f) };
        var hipColliderInitialTriggers = new List<bool> { false, false, true, true };
        ExplodeWithTransforms(minifig.transform, "Hip", minifig.GetHip(), hipTransforms,
hipColliderCenters, hipColliderSizes, hipColliderInitialTriggers, speed, angularSpeed, spreadForce,
removeDelay);

        // Use spine05 and spine01 transforms.
        var torsoTransforms = new List<Transform> { head.parent.parent.parent.parent,
head.parent.parent.parent.parent.parent.parent.parent.parent };
        var torsoColliderCenters = new List<Vector3> { Vector3.up * 0.1f, Vector3.up * 0.3f };
        var torsoColliderSizes = new List<Vector3> { new Vector3(0.8f, 0.5f, 1.2f), new
Vector3(0.8f, 0.5f, 1.4f) };
        var torsoColliderInitialTriggers = new List<bool> { false, false };
        ExplodeWithTransforms(minifig.transform, "Torso", minifig.GetTorso(), torsoTransforms,
torsoColliderCenters, torsoColliderSizes, torsoColliderInitialTriggers, speed, angularSpeed,
spreadForce, removeDelay);

        var headTransforms = new List<Transform> { head };
        var headColliderCenters = new List<Vector3> { Vector3.up * 0.48f };
        var headColliderSizes = new List<Vector3> { new Vector3(0.8f, 0.96f, 0.8f) };
        var headColliderInitialTriggers = new List<bool> { false };
        ExplodeWithTransforms(minifig.transform, "Head", minifig.GetHead(), headTransforms,
headColliderCenters, headColliderSizes, headColliderInitialTriggers, speed, angularSpeed,
spreadForce, removeDelay);

        var headAccessory = minifig.GetHeadAccessory();
        if (headAccessory)
        {
            var headAccessoryTransforms = new List<Transform> { head.GetChild(0) };
            // FIXME Get correct colliders for accessories.
            var headAccessoryColliderCenters = new List<Vector3> { Vector3.zero };
            var headAccessoryColliderSizes = new List<Vector3> { new Vector3(0.8f, 0.96f, 0.8f) };
```

```
            var headAccessoryColliderInitialTriggers = new List<bool> { true };
            ExplodeWithTransforms(minifig.transform, "Head accessory", new List<Transform>
{ headAccessory }, headAccessoryTransforms, headAccessoryColliderCenters,
headAccessoryColliderSizes, headAccessoryColliderInitialTriggers, speed, angularSpeed,
spreadForce + 1.5f, removeDelay, false);
        }

        // Hack for Santa's beard.
        var beardParent = head.GetChild(1);
        if (beardParent.childCount > 0)
        {
            var beardAccessory = beardParent.GetChild(0);
            if (beardAccessory)
            {
                var beardAccessoryTransforms = new List<Transform> { beardAccessory };
                var beardAccessoryColliderCenters = new List<Vector3> { Vector3.forward * 0.2f };
                var beardAccessoryColliderSizes = new List<Vector3> { new Vector3(0.8f, 0.96f,
0.8f) };
                var beardAccessoryColliderInitialTriggers = new List<bool> { true };
                ExplodeWithTransforms(minifig.transform, "Beard accessory", new List<Transform>
{ beardAccessory }, beardAccessoryTransforms, beardAccessoryColliderCenters,
beardAccessoryColliderSizes, beardAccessoryColliderInitialTriggers, speed, angularSpeed,
spreadForce + 1.5f, removeDelay, false);
            }
        }
    }

    private static void ExplodeLimb(Transform parentTransform, string name, List<Transform>
meshes, Transform root, Transform mid, Transform tip, float colliderWidth, Vector3 speed, float
angularSpeed, float spreadForce, float removeDelay)
    {
        var resultGO = CreateMeshGO(name, meshes, removeDelay, true);

        AddOrientedColliderFromTwoPositions(resultGO, mid.position, root.position,
colliderWidth);
        AddOrientedColliderFromTwoPositions(resultGO, tip.position, mid.position,
colliderWidth);

        AddAndMoveRigidBody(resultGO, parentTransform, speed, angularSpeed, spreadForce);
    }

    private static void ExplodeWithTransforms(Transform parentTransform, string name,
List<Transform> meshes, List<Transform> transforms, List<Vector3> colliderCenters,
List<Vector3> colliderSizes, List<bool> colliderInitialTriggers, Vector3 speed, float angularSpeed,
float spreadForce, float removeDelay, bool bakeMesh = true)
    {
        var resultGO = CreateMeshGO(name, meshes, removeDelay, bakeMesh);

        for (var i = 0; i < transforms.Count; ++i)
```

```
            {
                AddOrientedColliderFromTransform(resultGO, transforms[i], colliderCenters[i],
colliderSizes[i], colliderInitialTriggers[i]);
            }

            AddAndMoveRigidBody(resultGO, parentTransform, speed, angularSpeed, spreadForce);
        }

        private static GameObject CreateMeshGO(string name, List<Transform> meshes, float
removeDelay, bool bakeMesh)
        {
            var resultGO = new GameObject(name);
            resultGO.transform.localPosition = Vector3.zero;
            resultGO.transform.localRotation = Quaternion.identity;

            var resultMesh = new Mesh();

            var resultFilter = resultGO.AddComponent<MeshFilter>();
            resultFilter.sharedMesh = resultMesh;

            var resultRenderer = resultGO.AddComponent<MeshRenderer>();
            var materials = new List<Material>();

            var combineInstances = new List<CombineInstance>();
            for (var i = 0; i < meshes.Count; ++i)
            {
                var mesh = meshes[i];

                // Just parent the result with the parent of the first mesh.
                if (i == 0)
                {
                    resultGO.transform.SetParent(mesh.parent, false);
                }

                if (bakeMesh)
                {
                    var renderer = mesh.GetComponent<SkinnedMeshRenderer>();
                    materials.Add(renderer.sharedMaterial);

                    var bakedMesh = new Mesh();
                    renderer.BakeMesh(bakedMesh);

                    combineInstances.Add(new CombineInstance { mesh = bakedMesh });
                }
                else
                {
                    var renderers = mesh.GetComponentsInChildren<MeshRenderer>();
                    foreach (var renderer in renderers)
                    {
```

```
                materials.Add(renderer.sharedMaterial);
            }

            var filters = mesh.GetComponentsInChildren<MeshFilter>();
            foreach (var filter in filters)
            {
                combineInstances.Add(new CombineInstance { mesh = filter.sharedMesh });
            }
        }

        mesh.gameObject.SetActive(false);
    }

    resultRenderer.sharedMaterials = materials.ToArray();

    resultMesh.CombineMeshes(combineInstances.ToArray(), false, false);

    if (removeDelay > 0.0f)
    {
        var blinkAndDestroy = resultGO.AddComponent<BlinkAndDestroy>();
        blinkAndDestroy.timeLeft = removeDelay;
    }

    return resultGO;
}

private static void AddOrientedColliderFromTwoPositions(GameObject parent, Vector3
positionA, Vector3 positionB, float colliderWidth)
{
    var position = (positionA + positionB) * 0.5f;
    var direction = positionA - positionB;

    var colliderGO = new GameObject("Collider");
    colliderGO.transform.parent = parent.transform;

    colliderGO.transform.position = position;
    colliderGO.transform.rotation = Quaternion.LookRotation(direction);

    var collider = colliderGO.AddComponent<BoxCollider>();
    collider.size = new Vector3(colliderWidth, colliderWidth, direction.magnitude);
}

private static void AddOrientedColliderFromTransform(GameObject parent, Transform
transform, Vector3 colliderCenter, Vector3 colliderSize, bool colliderInitialTrigger)
{
    var colliderGO = new GameObject("Collider");
    colliderGO.transform.parent = parent.transform;

    colliderGO.transform.position = transform.position;
```

```
            colliderGO.transform.rotation = transform.rotation;

            var collider = colliderGO.AddComponent<BoxCollider>();
            collider.center = colliderCenter;
            collider.size = colliderSize;
            collider.isTrigger = colliderInitialTrigger;

            if (colliderInitialTrigger)
            {
                colliderGO.AddComponent<EnableCollider>();
            }
        }

        private static void AddAndMoveRigidBody(GameObject go, Transform parentTransform,
Vector3 speed, float angularSpeed, float spreadForce)
        {
            var rigidBody = go.AddComponent<Rigidbody>();
            rigidBody.AddForce(speed, ForceMode.VelocityChange);
            rigidBody.AddRelativeTorque(0.0f, angularSpeed * Mathf.Deg2Rad, 0.0f,
ForceMode.VelocityChange);
            rigidBody.AddExplosionForce(spreadForce, parentTransform.position, 0.0f, 0.0f,
ForceMode.VelocityChange);
        }
    }
}
```

Додаток Д

Taras Shevchenko National University of Kyiv

Development of a computer game 3D platformer with elements of the quest

Software Architecture Document (SAD)

Owner: Pavel Yurko

DOCUMENT NUMBER: 1.0

RELEASE: 1.0

RELEASE DATE: 01.06.2021

TABLE OF CONTENTS

1. Introduction

1.1. Purpose

The Software Architecture Document (SAD) provides a comprehensive architectural overview of computer game 3D platformer with elements of the quest. It presents a number of different architectural views to depict different aspects of the system.

1.2. Scope

This document describes the aspects of computer game 3D platformer with elements of the quest design that are considered to be architecturally significant; that is, those elements and behaviors.

1.3. Definitions

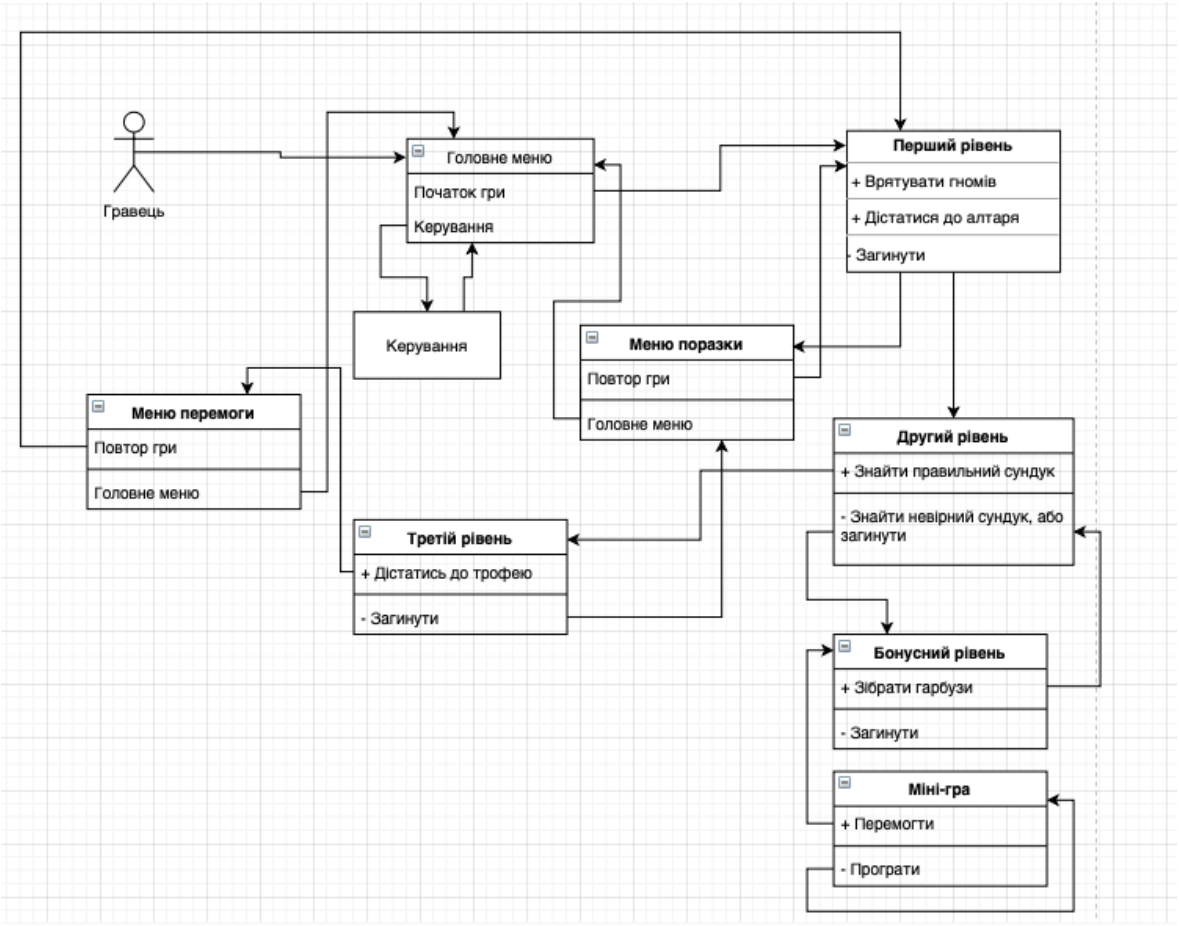
In filler, a platformer is fundamentally an obstacle course, a game where you must navigate an environment to reach a goal. Your main abilities, or “verbs” in game design terminology, are running and jumping, perhaps climbing or gliding.

2. Architectural Goals and Constraints

Develop a game, unique, that is a platformer with quests, which will be important in the game itself. Create sutible graphic for the game, make sure it has the elements, which are required for platformers.

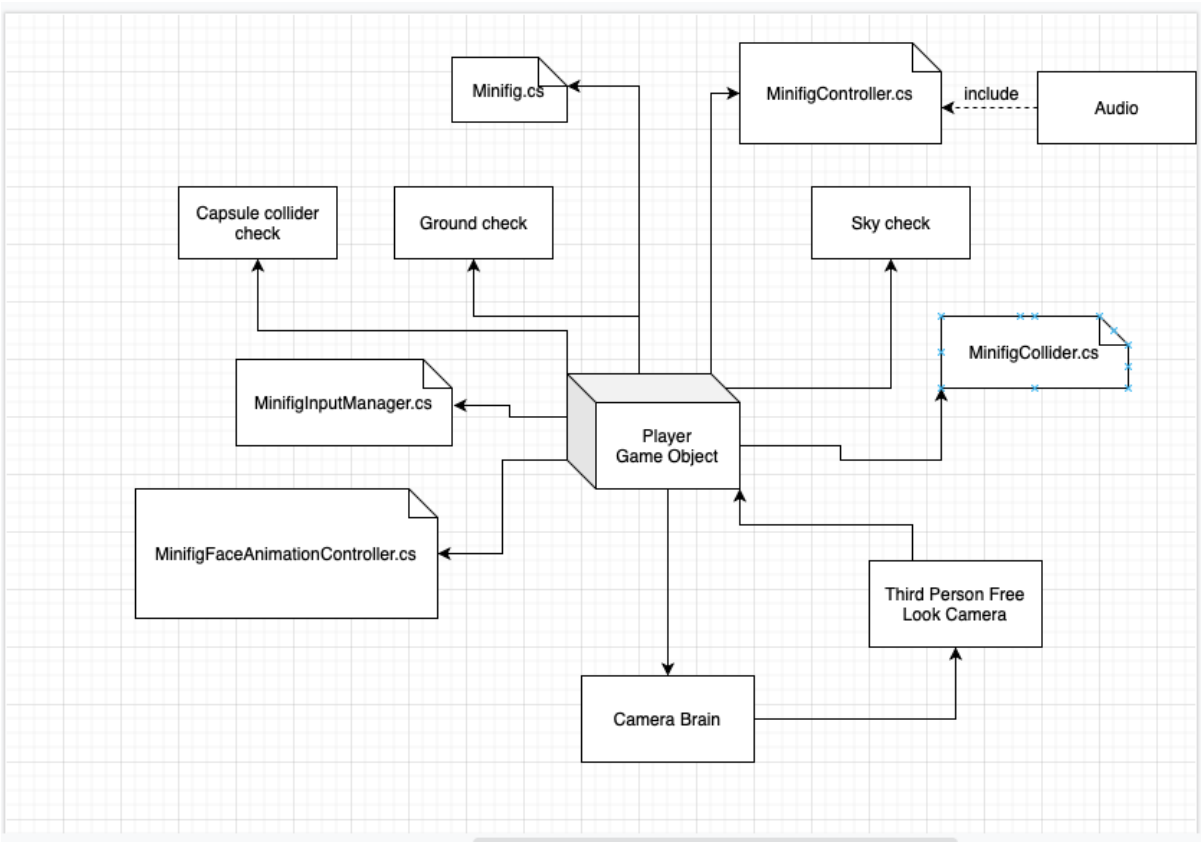
3. Use-Case View

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.



In this game everyrhing is simple, player activates the menu and can start playing the game.

4. Logical View



This shows how the game object interacts with other objects and scripts, that are put in the game.