МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Острозька Академія»

Економічний факультет

Кафедра економіко-математичного моделювання та інформаційних технологій

Кваліфікаційна робота/проєкт

на здобуття освітнього ступеня бакалавра

на тему: **«Розробка ігрового застосунку жанру side-scroller під платформу PC на ігровому рушію Unity в маркетингових цілях»**

Виконав: студент 4 курсу, групи КН-41

спеціальності 122 «Комп'ютерні науки»

освітньо-професійної програми

«Комп'ютерні науки»

Боць Володимир Юрійович

Керівник: Гаврильчик Леонід Сергійович

Рецензент: Місай Володимир Віталійович

"РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ"

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____(проф., д.е.н., Кривицька О. Р.)

Протокол № ____ від « ___ » _____ 2022 р.

Острог, 2022

Міністерство освіти і науки
Національний університет «Острозька академія»

Факультет: економічний
Кафедра: *економіко-математичного моделювання та інформаційних технологій*
Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри економіко-математичного моделювання
та інформаційних технологій

_____ Ольга КРИВИЦЬКА

«___» _____20__ р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ/ПРОЄКТ СТУДЕНТА

_____*Боця Володимира Юрійовича*_____

1. Тема роботи/проєкту: Розробка ігрового застосунку жанру side-scroller під платформу PC на ігровому рушію Unity в маркетингових цілях

керівник роботи/проєкту: Гаврильчик Леонід Сергійович
Затверджено наказом ректора НаУОА від 29 жовтня 2021 року № 110

2. Строк подання студентом роботи: 03 червня 2022 року

3. Вихідні дані до роботи/проєкту: постановка задачі, аналіз аналогів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): опис предметного середовища; огляд наявних аналогів; постановку задачі; опис архітектури рішення, що розроблюється; опис коду та інтерфейсу програми; тестування.

5. Перелік графічного матеріалу: діаграма прецедентів функціональних вимог, діаграма станів ігрового циклу, діаграма станів системи меню, діаграма станів додаткової сцени – пауза.

6. Консультанти розділів роботи/проєкту:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Гаврильчик Л.С.	01.12.2021р.	01.12.2021р.
2	Гаврильчик Л.С.	01.12.2021р.	01.12.2021р.
3	Гаврильчик Л.С.	01.12.2021р.	01.12.2021р.

7. Дата видачі завдання: 01.12.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи/проєкту	Строк виконання етапів	Примітка
1	Затвердження теми роботи/проєкту	до 01.11.2021 р.	
2	Постановка завдання	до 01.12. 2021 р.	
3	Розробка архітектури та загальної структури системи	до 01.02.2022 р.	
4	Розробка структур окремих підсистем	до 01.03. 2022 р.	
5	Програмна реалізація системи	до 01.05.2022 р.	
6	Попередній захист кваліфікаційної роботи/проєкту	до 01.06.2022р.	
7	Здача кваліфікаційної роботи/проєкту на кафедру	03.06.2022 р.	

Студент: _____ Володимир БОЦЬ

Керівник кваліфікаційної роботи/проєкту: _____ Леонід ГАВРИЛЬЧИК

АНОТАЦІЯ
кваліфікаційної роботи/проєкту
на здобуття освітнього ступеня бакалавра

Тема: Розробка ігрового застосунку жанру side-scroller під платформу PC на ігровому рушію Unity в маркетингових цілях
Автор: Боць В.Ю.
Науковий керівник: Гаврильчик Л.С.

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи/проєкту: 51 с., 41 рис., 18 джерел.
Ключові слова: рушій, гра, сценарій, розробка, ігровий додаток.
Короткий зміст праці:

Метою кваліфікаційної роботи/проєкту було створення ігрового додатку на рушію для застосування в маркетингових цілях. Для розробки було використано ігровий рушій Unity, мову програмування C#, а також графічний редактор Adobe Photoshop. Даний проект реалізує собою гру завдяки якій власники онлайн чи фізичних магазинів зможуть заохочувати своїх клієнтів до покупки товару, а саме клієнт зігравши в гру, отримує знижку відповідно до результату проходження яку пізніше й використовує при покупці. Користувачами додатку є власники тих чи інших магазинів та їхні клієнти.

The aim of the thesis was to create a game application for the Unity engine for use in marketing purposes. Unity game engine, C # programming language and Adobe Photoshop graphics editor were used for the development.

This project implements a game through which the owners of online or physical stores will be able to encourage their customers to buy goods, namely the customer playing the game, gets a discount according to the result of the passage which is later used when buying. Users of the application are the owners of certain stores and their customers.

ЗМІСТ

ВСТУП

Ви можете не любити відеоігри і не розуміти їхньої популярності, але неможливо заперечити, що це одна з найуспішніших галузей в еволюції технологій. Ігри стали невід'ємною частиною життя багатьох користувачів персональних комп'ютерів і мобільних пристроїв. Завдяки сильному розвитку цієї галузі створення відеоігор все частіше розглядається авторами та користувачами як окремий жанр мистецтва.

Як і будь-яка інша культурна індустрія, відеоігри мають величезний ринок, наповнений видавцями та розробниками, які створюють високобюджетні продукти, які продаються мільйонами щороку. Лише у 2020 році інтерактивні розваги, що охоплюють більшість медіа про ігри та їх оточення, принесли понад 100 мільярдів доларів доходу.

З року в рік це призводить до випуску значної кількості схожих один на одного продуктів. Зростання цифрового розповсюдження відеоігор дозволило більш дрібним і незалежним розробникам вести розробку без ризику, пов'язаного з витратами на створення фізичних копій гри. Це призвело до створення більш нішевих проектів.

Після міжнародного успіху таких ігор, як Braid, Castle Crashers та World of Goo, «інді-ігри» стали новою тенденцією в індустрії. Невеликі команди та відсутність творчих обмежень роблять їх відомими своїми інноваційними, творчими та експериментальними . Розробники мають обмежені можливості створювати дорогу і складну графіку, і вони покладаються на інноваційний характер своїх ігор.

Незалежні ігри або інді-ігри (англ. indie games від independent — «незалежний») — відеоігри, створені незалежно від фінансової підтримки великих видавців. Зачасту ці ігри дешеві або безкоштовні, багато з них мають невеликий розмір і тому поширюються через інтернет за допомогою цифрової дистрибуції або як freeware. Більшість спочатку вільних ігор також належить до цієї категорії.

Звичайно, інді-розробка відеоігор не є чимось новим у галузі, але за останнє десятиліття вона принесла величезний приріст. Ці проекти та їх розробники з кожним роком стають популярнішими, тому рух став ключовим для індустрії відеоігор.

Тільки в 2020 році було випущено близько 7700 відеоігор (близько 21 новий проект на день), що становить 39% від усіх випущених проектів на платформі Steam. Переважна більшість цих проектів від незалежних розробників. І ці цифри продовжують зростати все більше і більше. Але є і негативна сторона. Продажі в Steam в основному припадають лише на 100 популярних ігор (0,5% усіх ігор), що становить 50% загального доходу. Це пов'язано з тим, що більшість ігор у Steam – це проекти інді-розробників, дуже низької якості, оскільки останніми роками їх розробка стала простішою та дешевшою. Таким чином, незважаючи на те, що Steam був найуспішнішим минулого року, кількість нових користувачів, які прийшли на платформу і купили товари, не встигала за кількістю нових продуктів.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ КЛАСИФІКАЦІЙ ТА АНАЛІЗ ІСНУЮЧИХ РОЗРОБОК КОМП'ЮТЕРНИХ ІГОР

1.1 Аналіз та загальна характеристика комп'ютерних ігор

Комп'ютерна гра – комп'ютерна програма, що служить для організації ігрового процесу зв'язку з партнерами по грі, або сама виступає в якості партнера. До комп'ютерних ігор також відносять відеоігри й мобільні ігри. Під час комп'ютерної гри за допомогою спеціальних програм створюється імітація прямої взаємодії у віртуальному просторі між ігровим персонажем та користувачем (або групою користувачів) за певним алгоритмом. Спостерігаючи ігрову ситуацію на екрані монітора, гравець впливає на неї за допомогою клавіатури, «миші», контролера, тощо.

Комп'ютерні ігри часто розробляють на основі сторонніх джерел, таких як книги та фільми, але останнім часом існують приклади і зворотного напрямку, коли на основі відомої гри чи ігрової серії починають з'являтися додаткові матеріли, що розширюють всесвіт гри.

Більш того, спеціально розроблені ігри можуть виступати в якості навчального матеріалу або дозволяють використовувати гравців в науково-дослідних цілях. За деякими популярними іграми такими як CS GO,DOTA 2 та іншими, проводяться змагання різних видів масштабності, від регіональних до світових, які мають окрему назву «кіберспорт».

Комп'ютерні ігри мають настільки істотний вплив на сучасне суспільство, що останнім часом з'являється стійка тенденція до гейміфікації для неігрового прикладного програмного забезпечення.

З усього цього можна зробити висновок, що комп'ютерні ігри сьогодні стали невід'ємною частиною нашого життя. За останні роки сфера їх використання розширилася, ігри використовуються не тільки для розваг, а й для дослідження та навчання.

Оскільки критерії, за якими ігри належать до певного жанру, не визначені чітко, класифікація комп'ютерних ігор недостатньо систематизована, і в різних джерелах

дані про жанр конкретного проекту можуть розрізнятися. Однак розробники ігор сходяться на думці, що приналежність гри до одного з основних жанрів майже завжди можна однозначно визначити.

За основний критерій поділу жанрів беремо дії, які найбільш часто здійснюються в іграх цього жанру. Ігри діляться на три великі групи: ігри контролю, ігри дії та ігри інформації. Всі ігри, що входять до групи дуже схожі між собою, але одночасно з цим мають різні відхилення.

Ігри інформації: Як зрозуміло з назви, головне в іграх цієї групи-отримання інформації у всіх її проявах. В таких іграх іноді присутнє і планування, і динаміка, але інформація в них набагато важливіше. Золотою серединою групи є «RPG (Role Playing Game)» - «рольова гра». Ігри, в яких можна жити, вживатися в роль героя, в яких в якості головних достоїнств виставляють атмосферу, сюжет, ігровий світ.

Ігри дії: Головне в іграх цієї групи - рух, які необхідно здійснювати керуючи тілом (людським або гуманоїдним) , або технічним засобом. Золотою серединою групи є "Action" (гра-бойовик) – рахуються найбільш динамічними іграми. Саме ці ігри прийнято хвалити за те, що вони розвивають швидкість реакції.

Ігри контролю: Група «гри контролю» складається з тих ігор, головна суть яких - планування подій і управління для досягнення переваги в подальшому. Сюди потрапляють всі види стратегій, різні економічні ігри, варгейми, тактики. Золотою серединою групи є «Strategy» (звичайна локальна стратегія).

Усі ігри поділяються на одиночні та мультиплеєрні. Також мультиплеєрні ігри поділяються між собою на:

- Мультиплеєр на одному комп’ютері;
- Мультиплеєрні оффлайн-ігри;
- Масові онлайнові.

Саме завдяки цій класифікації встановлюються режими, що будуть присутні у грі. Одиночна гра – вид гри, у яких приймає участь одна людина. Зазвичай гравцю протистоїть штучний інтелект, а його метою є рух до кінця гри (проходження), накопичення ресурсів або прокачування навичок. Часто ці цілі комбінуються.

Іншим видом ігр є мультиплеєр. Цей режим гри пристосований до гри більше ніж однієї людини одночасно. В багатьох іграх одиночна гра там мултиплеєр комбінуються.

За візуальною складовою комп'ютерні ігри поділяються на:

- Текстові – гра з мінімальною кількістю графічних елементів, а спілкування з гравцем відбувається за допомогою тексту;
- 2D – усі елементи гри розроблені за допомогою двовимірної графіки;
- 3D – усі елементи гри розроблені за допомогою тривимірної графіки.

Після тривалого аналізу класифікацій комп'ютерних ігр, прийнято рішення розробити двовимірну гру у жанрі Платформер. Ігри цього жанру
Мають перевагу у зрозумілій та простій механіці, що є великим плюсом для початківців. Крім того, на відміну від ігор із 3D-графікою, для розробки 2D-ігор не потрібно багато робочої сили та ресурсів.

1.2 Аналіз існуючих розробок

Платфо́рмер (англ. platformer), також відомий як платфо́рмна гра (англ. platform game) — жанр відеоігор, ігровий процес в якому складається зі стрибків персонажа по різноманітних платформах (звідси і назва) та через перешкоди, збирання предметів, звичайно необхідних для завершення рівня чи отримання певної нагороди.

В платформерах гравець керує персонажем, який рухається платформами різних розмірів, висоти та масштабів в залежності від типу гри та її складності. Традиційно у перших двовимірних платформерах персонаж рухався лише зліва направо, але вже в сучасних іграх головний герой може рухатися у будь-якому напрямку.

Деякі предмети, зібрані гравцем під час гри, дають додаткову нагороду керованому гравцем персонажу, яка може бути як вичерпною (на певному рівні або протягом короткого періоду часу), так і до завершення гри. Предмети, зброя та різні ігрові нагороди зазвичай збираються за допомогою звичайних дотиків персонажа і не

вимагають від гравця особливих дій. Предмети збираються в «інвентарі» персонажа і використовуються за допомогою спеціальних команд. Вороги в іграх-платформерах представлені у вигляді так званих численних і різноманітних «монстрів» з примітивним штучним інтелектом, які намагаються якомога ближче підійти до гравця і завдати йому шкоди, або взагалі не мають штучного інтелекту, продовжуючи певні траєкторії або повторюючи дії. У багатьох випадках зіткнення з ворогом призведе до втрати здоров'я або смерті персонажа. Іноді вороги задані тікати, видавати звуки або змінювати свій зовнішній вигляд при наближенні до гравця. У перших платформерах вороги нейтралізувались звичайним стрибком на них, але на сьогодні майже в усіх платформерах головний герой володіє якоюсь зброєю для боротьби з монстрами. Переможені монстрами зазвичай зникають або провалюються за ігрові елементи.

Рівні часто мають приховані проходи в стінах та заховані у важкодоступних місцях предмети, які можуть істотно полегшити гравцю проходження гри.

Майже усі ігри цього жанру характеризуються фентезійними елементами та мальованою графікою. Головними героями таких ігор можуть бути як і звичайні люди у незвичайних обставинах, так і міфічні істоти (наприклад гноми, дракони) чи антропоморфні тварини.

Першу активність жанр почав проявляти на початку 1980-х років, з виходом на той час дуже популярних платформерів Pitfall та Super Mario Bros, поява яких сприяла розвитку усього ігрового жанру. Поява Pitfall принесла до індустрії зміну в системі проходження рівнів з вертикальної до горизонтальної, що залишилось і до нашого часу. Натомість Super Mario Bros є прикладом для творців ігор, завдяки розмірам та складності ігрових рівнів.

Платформер став для багатьох з нас жанром, з якого розпочалося знайомство зі світом відеоігор: для когось першою грою стала Super Mario Bros, хтось захоплювався швидкістю Соніка на Sega Mega Drive, а хтось долав перешкоди у Crash Bandicoot на PS One. Жанр досі залишається на піку популярності, тому вибір платформерів на ПК та консолях тішить різноманітністю.

Оскільки технології з кінця 1980-х отримали стрімкий розвиток, аналізувати будемо саме сучасні приклади платформерів, такі як:

- Sonic Mania;
- Cuphead;
- Limbo.

Sonic Mania

Комп’ютерна відеогра серії Sonic the Hedgehog у жанрі платформер, розроблена Крістіаном Уайтхедом та студіями Headcannon та PagodaWest Games та видана компанією Sega для платформ PlayStation 4, Xbox One, Switch та персональних комп'ютерів під керуванням Windows влітку 2017 року. (Рис 1.1.)

Ігровий процес і візуальний стиль Sonic Mania схожі на перші частини серії для приставки Mega Drive/Genesis. За сюжетом їжак Сонік, лис Тейлз та єхидна Наклз виявили на острові Ангелів потужне джерело енергії — Рубін Ілюзій, який дозволяє контролювати час і простір, проте доктор Еггман разом зі своїми роботами Hard Boiled Heavies дісталися до нього першими, щоб використати камінь з метою захоплення світу . Герої збираються запобігти підступним планам лиходіїв. Гравцеві доведеться проходити різні рівні, серед яких є як старі зони з перших частин серії, так і нові. Серед ігрових персонажів доступні Сонік, Тейлз і Наклз, кожен з яких має свої унікальні здібності та особливості проходження зон. Гравці вибирають одного з 3 ігрових персонажів, кожен зі своїми унікальними здібностями: Сонік може виконувати «випадаючий ривок», який змушує його скочуватися після стрибка, Тейлз може літати і плавати, Наклз може ковзати і лазити. Як і у Sonic 2 (1992), гравці можуть грати за Соніка і Тейлза одночасно, або другий гравець може керувати Тейлзом незалежно.

Платформер був присвячений 25-річному ювілею серії Sonic the Hedgehog. Розробники поставили собі за мету відтворити оригінальний геймплей класичних ігор серії, при цьому додавши нові можливості та покращення.

Рис 1.1 Знімок екрана з гри Sonic Mania.

Cuphead

Відеогра жанрів «біжи і стріляй» і платформера, розроблена і видана канадською командою StudioMDHR Entertainment. Видана для Windows і Xbox One 29 вересня 2017 року, для macOS 19 жовтня 2019, та для Nintendo Switch 17 квітня 2019 року. Гра виконана в комічному стилі анімації Disney 1930-х років та оповідає про пригоди двох хлопців — Капхеда і Маґмена, що повинні вчасно перемогти низку ворогів аби врятувати свої душі від Диявола.

Головний герой, Капхеда – це хлопчик з чашкою замість голови, та здійснює постріли по ворогах із пальця. Програвши суперечку з дияволом, Капхед повинен пройти через велику кількість босів, щоб погасити борг перед дияволом та повернути свою душу. Доступ до рівнів у грі відбувається за допомогою мапи зовнішнього світу, що виконана у стилі ігор жанру Action-RPG, сама мапа є розгалуженою послідовністю рівнів, а також має свої таємні проходи та магазин, у якому персонаж може придбати додаткові життя та різні допоміжні навички для боротьби з ворогами.

Крім звичайних пострілів з пальца, у головного героя є можливість поєднувати різні типи атак і об’єктів долонею, а при вдалому поєднання атак назбирується шкала кредитів , яка дозволяє використовувати інші спеціальні навички.

У цілому бойова система в грі складається у багаточисельних битв с босами у стилі ігор жанра shoot’em up («біжи та стріляй») , де кожен бос має свої унікальні навички, а після перемоги на екран виводиться статистика з успіхами гравця, що дозволяє при бажані перемогти боса ще раз для підвищення своїх результатів. Гра Cuphead також має кооперативний режим гри, в якому приймає участь ще один

гравець управляючи персонажем Mugman, та допомагаючи головному герою у битвах з ворогами. (Рис 1.2)

Рис 1.2 Знімок екрану з гри Cuphead.

Limbo

Limbo — це 2D платформер, побудований на основі фізики рушія Box2D, що відповідає за керування об'єктами навколишнього середовища та ігровим персонажем. Гравець керує безіменним головним героєм. Він має спрямовувати його крізь небезпечне середовище та пастки в пошуках сестри. Дозволяється рух вправо і вліво, стрибки та взаємодія з об'єктами. В разі загибелі персонаж опиняється на початку глави, яких всього є 39. Розробники побудували такий дизайн головоломок, за яким головний герой неодмінно буде помирати перед тим, як знайде розгадку. Playdead назвали такий стиль гри стилем «спроб та смертей» та використовують образи жахливої смерті хлопчика у разі невдачі, аби підштовхнути гравця до правильного шляху. Пройдені раніше глави можна пройти заново.

Головоломки передбачають переміщення ящиків, блоків, натискання кнопок, що запускають механізми й керують силою тяжіння. Головного героя чекатимуть пастки до яких можна віднести: прірви, водойми, капкани, циркулярні пили та шипи, а також хижаки та інші діти, які намагаються вбити хлопчика особисто чи заманити в пастку.

Рис 1.3 Знімок екрана з гри Limbo

1.3 Постановка задачі

Ціль та задача проекту

В данній роботі було реалізовано ігровий застосунок в жанрі «Side-Scroller» під платформу PC на рушію Unity у сетингу – «2D Platrformer».

Для досягнення поставленої цілі необхідно було вирішити такі моменти:

- Проаналізувати аналоги;
- провести аналіз та обрати рушій та мову програмування для реалізації гри;
- описати концепцію гри;
- створити графічне зображення персонажа;
- спроектувати програмну систему;
- реалізувати гру;

Знайомство з світом

Так як гра була створена для магазину фірмового одягу, відповідно і навколишнє середовище мало бути з цим пов’язано. Для цього було обрано задній фон на якому зображені магазини, та сам герой одягнений в фірмовий одяг, який

ідеально підходить під концепцію гри. Також тематику гри особливо підкреслюють монети в вигляді кросівок.

Рис 1.4 Фон гри

Рис. 1.5 Головний герой

Рис 1.6 Монети в вигляді кросівок

Висновок до розділу 1

Після аналізу існуючих розробок, можливо зробити висновок, що ігор, які включають себе лише один жанр платформера дуже невелика кількість, через те що інші ігрові жанри також розвивались і платформери перестали відповідати вимогам гравців. Однак не можна говорити, що ігри цього жанру втратили свою популярність. На сьогоднішній день виходить маса різних платформерів, які можна зарахувати і до інших жанрів. Це може говорити про те, що «чисті платформери» стали продуктом направленим на вузький круг споживачів, тому для отримання великих прибутків розробники почали додавати до цього жанру елементи сторонніх ігрових жанрів. Тому можливо зробити заключний висновок, що для успіху гри її потрібно урізноманітнити елементами різних жанрових категорій.

РОЗДІЛ 2
РОЗРОБКА ПРОЕКТУ

2.1 Мета проекту. Визначення жанру та стилю гри.

Мета проекту – розробити 2D гру на рушії Unity в жанрі платформер основна задача якої використання в маркетингових цілях.

Проаналізувавши рушії було вибрано Unity та наступним кроком став вибір жанру гри та стилю. Вибір впав на жанр 2D платформер, так як він найбільше підходить під поставлену задачу, має буди продукт який просто сприймається, та не навантажений різними додатковими персонажами та надзвичайною графікою.

Стиль було вибрано саме 2D по тій причині, що гра має бути з мінімальними вимогами, для безперешкодного відтворення як і на низькопродуктивних пристроях так і на високопродуктивних. Також подібні ігри не мають бути переповнені різного роду графічними елементами. Продукт має бути інтуїтивно зрозумілий, простий та оптимізований.

2.2 Аналіз середовищ розробки та обґрунтування вибору технології розробки проекту

Ігровий рушій – програмне забезпечення, призначене для розробки комп'ютерних ігор та інших інтерактивних програм, що обробляють графіку у режимі реального часу. Для аналізу було взято на розгляд декілька ігрових рушіїв – Unreal Engine 4, Godot, Unity.

Unreal Engine

Ігровий рушій, що розробляється та підтримується компанію Epic Games. Цікавим представником цього рушія являється популярна гра Fortnite яка набрала великої популярності в 2018 і до цього часу являється одною з найпопулярніших ігор свого жанру. Спочатку Unreal Engine був призначений для розробки шутерів від першої особи, але його наступні версії успішно застосовувалися в іграх самих різних жанрів, в тому числі стелс-іграх, файтингах та і масових багатокористувацьких рольових онлайн-іграх.

Рис 2.1 Знімок екрана з гри Fortnite.

Unreal Engine підтримує більшість існуючих платформ та операційних систем – Windows, Linux, MacOS, iOS, Android, Xbox One, Xbox 360, PlayStation 2, PlayStation 3, PlayStation 4, GameCube, Wii, Wii U, Nintendo Switch, PSP, Dreamcast.

Також для комфортної роботи з редактором Unreal Engine персональний комп'ютер має задовольняти мінімальні системні вимоги:

- Версії операційних систем повинні бути не нижче ніж Windows 7/8/10 64x, Linux Ubuntu 15.04 та MacOS 10.13;
- Процесор Intel Quad-core чи AMD (з частотою 2.5Hz та вище);
- Оперативну пам'ять 8 GB (для операційних систем Linux 16 GB);
- Відеокарта AMD Radeon 6870 HD або NVIDIA GeForce 470 GTX та вище.

Godot

Відкритий багатоплатформовий 2D та 3D гральний рушій під ліцензією MIT, що розробляється співавторством Godot Engine Community. До публічного релізу у вигляді відкритого ПЗ рушій використовувався всередині деяких компаній Латинської Америки. Оточення розробника працює на Windows, Linux, OS

X, BSD і Haiku та може експортувати ігрові проекти на ПК, консолі, мобільні та веб платформи. Яскравим представником цього рушія є гра Anthill.

Рис 2.2 Знімок екрану з гри Anthill.

Задача Godot — бути максимально інтегрованим та самодостатнім середовищем для розробки ігор. Середовище дозволяє розробникам створювати ігри з нуля, не користуючись більше ніякими інструментами окрім тих, що потрібні для створення ігрового контенту (елементи графіки, музичні треки тощо). Процес програмування також не потребує зовнішніх інструментів (хоча при необхідності використовувати зовнішній редактор, це зробити досить легко).

Хоча Godot є хорошим та простим для розуміння продуктом, він ще досі перебуває на стадії розробки та не підтримує певний ряд функцій які є в Unity та Unreal Engine.

Для роботи з ігровим рушієм потрібні відносно невисокі характеристики комп’ютера:

- Персональний комп’ютер з операційною системою Windows 7 / MacOS 10.12;
- Оперативна пам’ять 4 GB;
- Відеокарта з підтримкою OpenGL 2.1.

Unity

Кросплатформерний рушій для розробки ігор, розроблений компанією Unity Technologies. Unity підтримує майже усі платформи – Windows, Linux, MacOS, Android, iOS, FireOS, PlayStation Vita, PlayStation 4, Xbox One, Nintendo Switch, Nintendo 3DS, Oculus Rift, Steam VR, Gear VR, PlayStation VR, Android TV, Smart TV, TvOS. Для роботи з Unity потрібен комп'ютер з мінімальними вимогами:

- Персональний комп'ютер з операційною системою Windows 7/8/10 x64 або MacOS X 10.9+;
- Процесор з підтримкою SSE2;
- Відеокарта з підтримкою DirectX 10

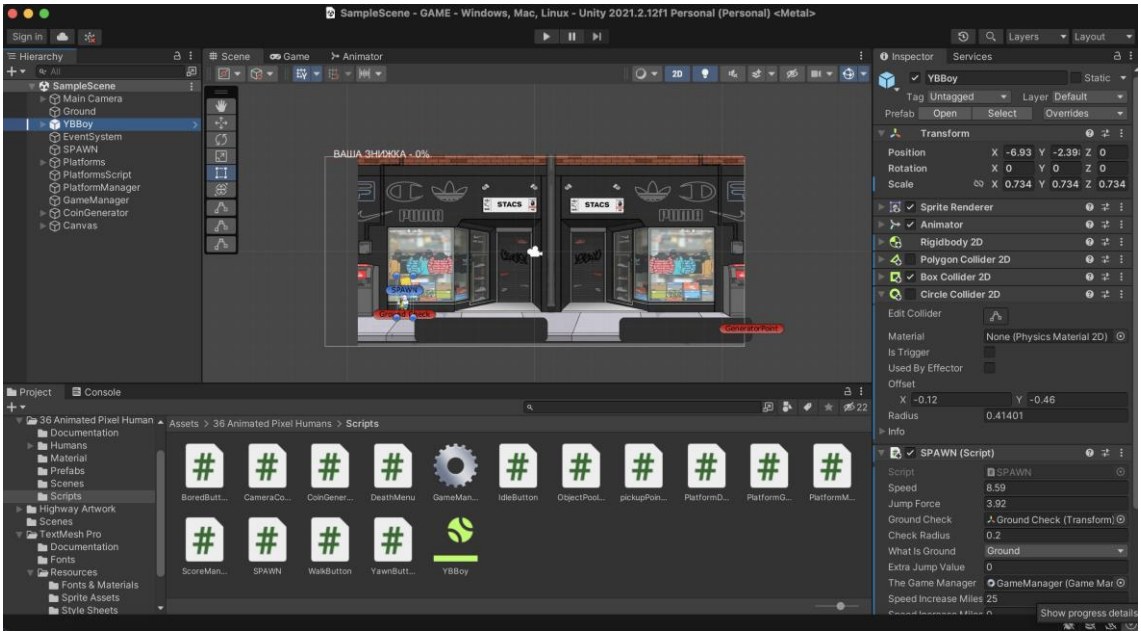


Рис 2.3 Інтерфейс Unity.

Unity відзначається чудовою оптимізацією, що відображається на якості та швидкості роботи проекту. Підтримує мови програмування C# та UnityScript.

C# — об'єктно-орієнтована мова програмування, розроблена у 1998−2001 роках групою інженерів під керівництвом Андерса Хейлсберга в компанії Microsoft як мова розробки застосунків для платформи MicrosoftNETFramework Вона відноситься до мов з C-подібним синтаксисом, і 18 найбільш всього схожа на C++ та

Java. За допомогою C# розробляється різноманітне програмне забезпечення: веб-сайти, веб-додатки, офісні додатки, десктопні та мобільні додатки, ігри та ін.
Основні переваги C#:

- створена паралельно з каркасом Framework .Net, тому ця мова повністю враховує всі його можливості — як FCL (Framework Class Library — стандартная библиотека класів платформи «.NET Framework»), так і CLR (Common Language Runtime — «загальномовне виконавче середовище» — віртуальна машина);
- повністю об'єктно-орієнтована мова, де навіть типи, вбудовані у мову, представлено класами, з можливостями спадкування й універсалізації;
- успадкувала кращі риси мов C/C++;
- завдяки каркасу Framework .Net, що стали надбудовою над операційною системою, програмісти C# одержують ті самі переваги роботи з віртуальною машиною, що й програмісти Java;
- виконавче середовище CLR є компілятором проміжної мови, у той час як віртуальна Java-машина є інтерпретатором;
- потужна бібліотека каркасів підтримує зручність побудови різних типів застосунків мовою C#, дозволяючи легко будувати Web-служби, інші види компонентів, досить просто зберігати й одержувати інформацію з бази даних й інших сховищ даних.

Unity має свій відомий магазин Store Unity Asset Store - це бібліотека ассетів, яка збільшується кожен день. Unity Technologies та члени спільноти створюють асети та публікують в цьому магазині. Типи ассетів варіюються від текстур, анімацій та моделей до прикладів закінчених проектів, навчальних матеріалів та розширень редактора. Багато ассетів надаються безкоштовно, тоді як інші доступні за помірними цінами; всі ці ассети ви можете завантажувати у свій проект Unity.

Unity пропонує 3 версії придбання: Personal (безкоштовна), Professional (45$/місяць), Enterprise (250$/місяць). Версії відрізняються лише пропонованим функціоналом та максимально допустимим річним прибутком (100 тисяч доларів, 200

тисяч доларів та без обмежень відповідно). Рушій є дуже різностороннім та унікальним, що дозволяє використовувати його для розробки різноманітних продуктів.

Так як головними критеріями для реалізації проекту були перш за все підтримка створення 2D проектів, безкоштовна версія, зрозумілий інтерфейс та не значні характеристики ПК, був вибраний рушій Unity який відмінно підходить під даний проект.

Написання коду у Unity буде відбуваєтись за допомогою Microsoft Visual Studio (Рис 2.4.)

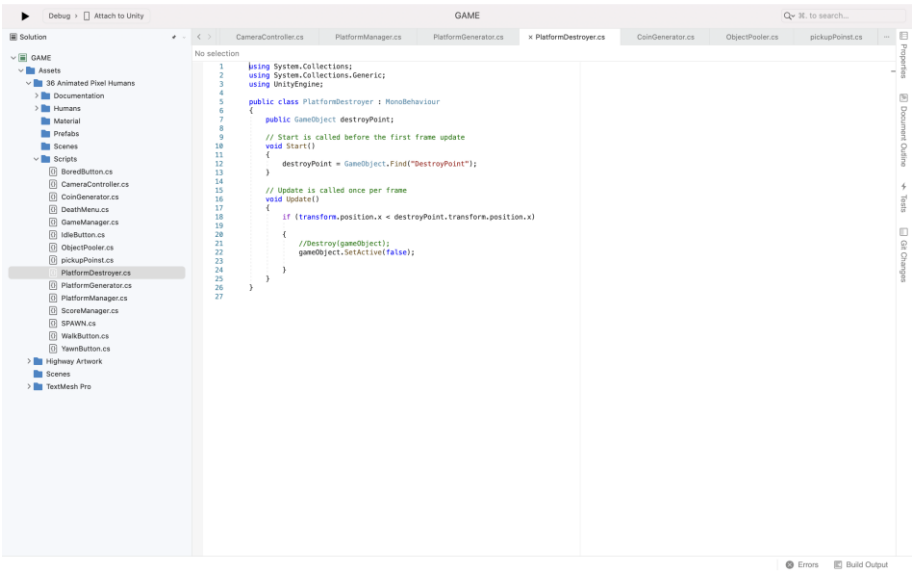


Рис. 2.4 Інтерфейс Visual Studio.

Visual Studio Community 2022 – інтегроване середовище розробки програмного забезпечення від фірми Microsoft. Дане середовище дозволяє створювати різноманітні програмні продукти: консольні програми, програми з графічним інтерфейсом, наприклад віконні додатки Windows Forms, а також Web-додатки тощо. Середовище Visual Studio дозволяє розробляти додатки, використовуючи різні мови програмування: Visual C#, Visual Basic, Visual F#, Visual C++, Python і т.д. Також існує можливість розробляти додатки не тільки під Windows, а і під інші популярні платформи: Android, iOS.

2.3 Створення анімації спрайтів

Анімація – процес оживлення зображення, методом створення низки зображень з відмінностями, які пізніше створюють кадри і відповідно покадрову анімацію.

Кадр – одне з багатьох зображень у анімації.

Покадрова анімація – анімація в якій використовується декілька кадрів відмінних один від одного.

Комп'ютерна анімація — вид анімації в якому об'єкти створюються з допомогою комп'ютерних засобів.

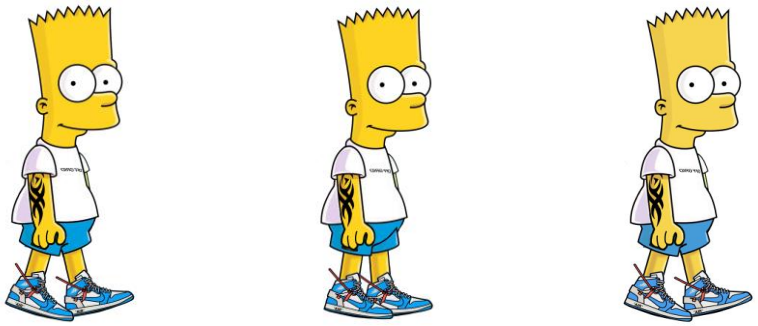


Рис 2.5 Приклад покадрової анімації

Графічний редактор Adobe Photoshop

Для створення анімації спрайтів було використано графічний редактор Adobe Photoshop. Використання цього редактора обумовленне певним досвідом його використання.

Adobe Photoshop (Едоубі Фотошоп) — графічний редактор, розроблений і поширюваний фірмою Adobe Systems. Цей продукт є лідером ринку в галузі комерційних засобів редагування растрових зображень і найвідомішим продуктом фірми Adobe. Часто цю програму називають просто Photoshop (Фотошоп). У наш час Photoshop доступний на платформах Mac OS X/Mac OS і Microsoft Windows.

Використання цього редактора обумовленне певним досвідом його використання.

Створення кадрів

Створення кадрів відбувалось наступним чином, ми маємо звичайне статичне фото, за допомогою інструменту Прямолінійне Лассо вирізаємо частину яку ми хочему анімувати, припустимо ногу, копіюємо на новий шар і міняємо її положення. Звичайно це не єдиний метод анімації, але він мені найбільше підходив так як він досить швидко робиться.

Рис 2.6 Використання Лассо

Після зміни положення в даному випадку, частини ноги, зберігаємо кадр для подальшого виставлення на полотно, яке створюється в форматі PNG та на нього виставляються кадри анімації.

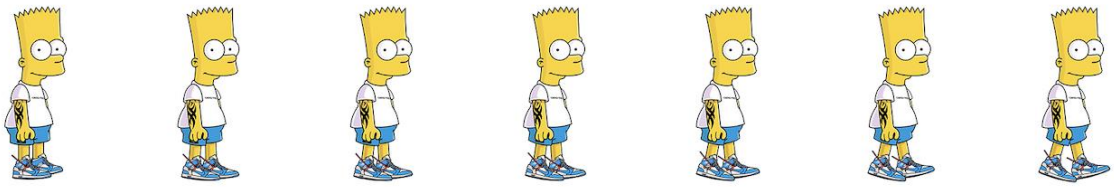


Рис 2.7 Полотно з кадрами

Створення анімації героя в Unity

Перейшовши в Unity завантажуємо PNG файл до файлів гри та додаємо компонент, який буде репрезентувати персонажа – **спрайт рендерер** (анг. *sprite renderer*). Він дозволяє показувати зображення на сцені у вигляді спрайтів.

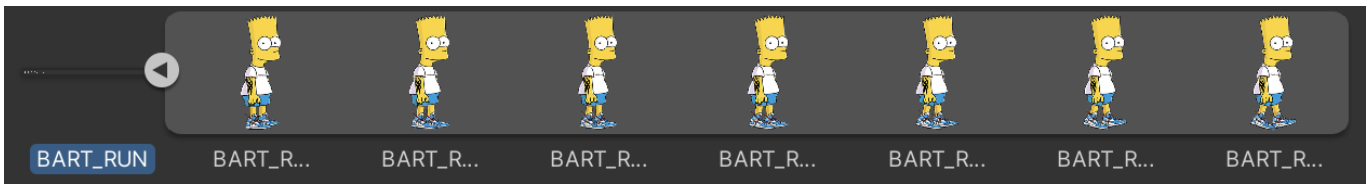


Рис 2.8 Спрайти персонажа

Після чого переходимо в Animator – інструмент для анімації в Unity, та виставляємо зображення в певному порядку на , для створення покадрової анімації.

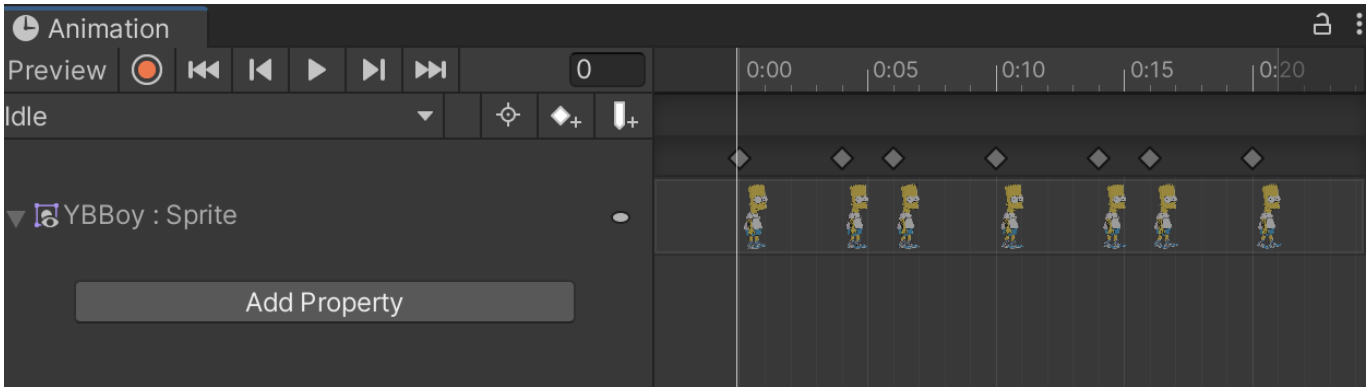


Рис 2.9 Кадри виставлені в редакторі анімацій

2.4 Створення базової сцени в Unity та застосування ігрових ассетів.

Ігровий Ассет (англ. Game Asset) – цифровий об’єкт або ж набір об’єктів, переважно складається з однотипних даних, неподільна сутність, яка представляє частину ігрового контенту в має деякі властивості.

До **ігрових ассетів** (ресурсів) відносяться всі дані, що використовуються та оброблюються комп'ютерною грою: геометричні моделі, текстури, звукові доріжки, тексти діалогів, анімаційні дані, моделі поведінки ігрового та ін. Сукупність всіх ассетів представляє контент гри, тоді як ігровий рушій є сукупністю всіх механізмів, що обробляють контент.

Основною складовою частиною будь-якої комп’ютерної гри є її візуальна складова. Перед початком опису частини графічного оформлення, слід розібрати основні поняття, що використовуються у геймдизайні, такі як спрайт та тайл.

Спрайт (з англ. sprite) – у двовимірних іграх є графічним зображенням якогось об’єкту, та може містити у собі декілька зображень, з яких можна зробити анімацію до об’єкту.

Спрайти ігрового героя

У якості персонажа у нашій гри буде використано спрайти одного із головних героїв мультиплікаційного серіалу Сімпсони Барт’а. (Рис 2.1.)

Рис 2.10 Спрайт головного героя

Спрайти інтерфейсу

Після ігрового персонажа наступним кроком слід підібрати спрайти для фонового зображення, а також елементи візуальної складової гри. У якості фонового зображення було підібрано фон з магазинами одягу. (Рис 2.2.)

Рис 2.11 Спрайт фону

Оскільки мета створення саме платформера, то для реалізації ігрового процесу потрібно також підібрати спрайти для саме платформ, по яких буде стрибати та бігати ігровий персонаж.

Рис 2.12 Спрайт платформ

Спрайт монеток

Останнім елементом візуальної складової гри є спрайти монеток, які будуть розташовані на платформах. Було обрано спрайт кросівка. (Рис 2.4.)

Рис 2.13 Спрайт монеток

Для створення базової сцени потрібно помістити всі ассети на ігрову сцену, і в результаті отримуємо початкову сцену, яка на даний момент є статичним набором ігрових об’єктів. (Рис 2.5.)

Рис 2.14 Вигляд ігрової сцени

Додавання та налаштування компонентів для героя та платформ.

На початку потрібно зробити так щоб ігрові об’єкти, змогли взаємодіяти між собою, для цього додаємо до них компонент **Rigidbody 2D**.

Rigidbody2D – компонент, який надає фізичні властивості для об’єкта. Якщо цей компонент доданий для нашого герою, він може взаємодіяти, з іншими фізичними об’єктами на сцені.

Будь-який ігровий об’єкт повинен містити компонент який згадувався вище, щоб на нього могла діяти гравітація, діяти відповідно до зазначених в сценарії інструкцій чи взаємодіяти з іншими обєктами через фізичний рушій.

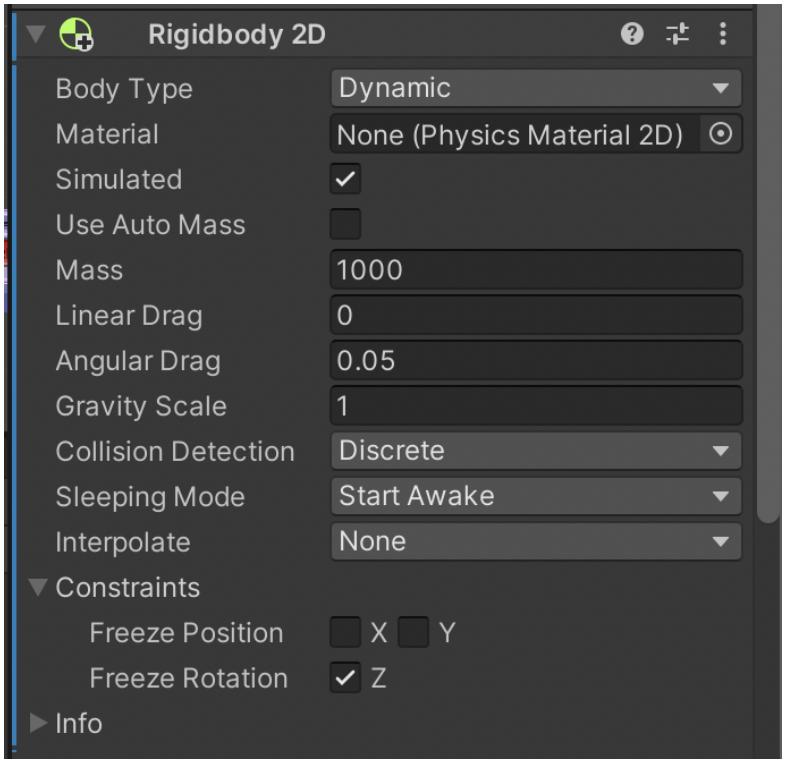


Рис 2.15 Налаштування компонента Rigidbody 2D

Далі застосовуємо компонент який буде визначати межі об’єктів. Для цього використовуємо – **Collider 2D** компонент, який визначає форму об’єкта. Цей компонент застосовується до платформ та до героя.

2.5 Написання сценаріїв для героя, головної камери, генерації платформ та монет, їхнього створення та безпечного видалення.

З візуальною складовою було вирішено, тому можна перейти до поетапного проектування елементів гри.

У грі ми повинні керувати персонажем за допомогою клавіатури, тому у подальшій роботі ми будемо застосовувати - Сценарії (анг. Scripts).

Сценарії (анг. scripts) – це послідовність дій, які можна застосувати до об'єктів в Unity, які описані за допомогою скриптової мови програмування, які виконуються під час гри.

Також в коді будуть застосовані такі функції як **Start(), Update(), FixedUpdate()**.

Start() - викликається перед першим кадром.

Update() - викликається раз в кадр і є головною функцією для оновлення кадрів.

FixedUpdate() – може викликатись декілька раз за один кадр.

Переходимо до написання сценаріїв. Так як гра жанру "Runner" у головного героя є дві основні функції, такі як: ходьба (Move), стрибок(Jump).

Рух героя

Першим буде сценарій ходьби який виглядає наступним чином :

```
public class SPAWN1 : MonoBehaviour
{
    Rigidbody2D rb;
    public float speed;
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();

    }

    public void Update()
    {
        rb.velocity = new Vector2(speed, rb.velocity.y);
    }
}
```

Рис.2.16 Сценарій руху

В сценарії героя, реалізовуємо компонент **Rigidbody 2D** та вводимо змінну speed яка відповідає за швидкість героя. В функції void Start() робимо взаємодію з компонентом і в функції void Update() створюємо вектор на який буде діяти швидкість яка задана в параметрах по осі Y.

Також пишемо скрипт для виконання стрибку, так як сама гра полягає в тому щоб здолати певні перешкоди та опинитись на іншій платформі. Сценарій повинен бути написаний таким чином щоб можна було уникнути стрибку в повітрі а також подвійного стрибку, тому отримуємо наступний скрипт :

```
public class SPAWN1 : MonoBehaviour
{
    Rigidbody2D rb;
    public float speed;
    private int extraJumps;
    public float jumpForce;
    private bool isGrounded;
    public LayerMask whatIsGround;
    public Transform groundCheck;
    public float checkRadius;
    private float moveInput;
    public int extraJumpValue;

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

    }

    void FixedUpdate()
    {
        isGrounded = Physics2D.OverlapCircle(groundCheck.position, checkRadius, whatIsGround);
        moveInput = Input.GetAxis("Horizontal");


    }

    public void Update()
    {

        rb.velocity = new Vector2(speed, rb.velocity.y); // на РБ йде тиск і вказуємо зо діємо тільки по горизонталі
        if (isGrounded == true)
        {
            extraJumps = extraJumpValue;
        }


        if (Input.GetKeyDown(KeyCode.Space) && extraJumps > 0)
        {
            rb.velocity = Vector2.up * jumpForce;
            extraJumps--;
        }

        else if (Input.GetKeyDown(KeyCode.Space) && extraJumps == 0 && isGrounded == true)
        {
            rb.velocity = Vector2.up * jumpForce;
        }


    }
}
```

Рис. 2.17 Сценарій стрибка

Сценарій стрибка складається з нових змінних таких як isGrounded, whatIsGround, groundCheck, jumpForce. isGrounded – містить в собі змінні

whatIsGround, groundCheck, які визначають знаходження персонажа на платформі чи поза нею. whatIsGround – змінна в якій задається шар платформи. groundCheck – це точка яка кріпиться на нижню частину персонажа для перевірки чи він зараз на платформі.

Слідкування камери

Головна камера (анг. Main Camera) - пристрій, який захоплює та відображає сцену гравцю.

У рушію Unity камери є пристроями, які захоплюють та відображають світ гравцю. Можна зробити камеру статичною, яка охоплює всю сцену, або зробити камеру дочірньою по відношенню до гравця та розмістити її на рівні очей персонажа. Але для нашої гри потрібно закріпити камеру за гравцем і змусити її слідувати за ним тому використовується сценарій - **CameraController**.

```
public class CameraController : MonoBehaviour
{
    public SPAWN player;
    Vector3 lastPosition;
    float distanceToMove;


    void Start()
    {
        player = FindObjectOfType<SPAWN>();

        lastPosition = player.transform.position;

    }

    // Update is called once per frame
    void Update()
    {
        distanceToMove = player.transform.position.x - lastPosition.x;
        transform.position = new Vector3(transform.position.x + distanceToMove, transform.position.y,transform.position.z);
        lastPosition = player.transform.position;
    }
}
```

Рис.2.18 Сценарій CameraController

Створено клас **CameraController** в якому задаємо три змінні : public SPAWN player – дозволяє зчитати інформацію з сценарію гравця, Vector3 lastPosition – дозволяє пересуватись по осях, float distanceToMove – зчитує на скільки пересунувся персонаж та на скільки потрібно пересунутись камері. В функції void Start() задаємо пошук об’єкта на якому є сценарій SPAWN та останнє положення персонажа це те де він зараз знаходиться. В функції void Update() відстань на яку потрібно пересунутись,

рух камери тільки по осі X та те що останнє положення дорівнює теперешньому положенню гравця.

Генерація платформ

Наразі постає питання генерації поверхні на якій буде рухатись об’єкт героя. Потрібно написати такий код який буде сам генерувати платформи на різній відстані та різного розміру для ускладнення геймплею.

Для вирішення поточної проблеми існує два варіанти створення таких платформ.Перший це створення безкінечної кількості платформ які генеруються, але тоді виникне проблема, з навантаженням пам'яті і коли всі ресурси будуть вичерпані гра не плановано завершиться.

Другий варіант, це генерація нової платформи, та її видалення відразу після того як вона виходить за межі сцени, тим самим нова платформа буде генеруватись, а стара відразу видалятись. Саме цей метод і буде використаний у роботі.

Процес створення платформ та їх видалення складається з трьох сценаріїв :

PlatformManager, PlatformGenerator, PlatformDestroyer. PlatformManager – зберігає масив платформ, **PlatformGenerator –** дістає платформи з масиву та робить їх активними**, PlatformDestroyer –** видаляє платформи, вимикаючи їх в масиві.

Сценарій – **PlatformManager** :

```
public class PlatformManager : MonoBehaviour
{
    public GameObject highway;
    public int platformAmount;

    List<GameObject> platforms;

    // Start is called before the first frame update
    void Start()
    {
        platforms = new List<GameObject>();
        for (int i = 0; i < platformAmount; i++)
        {
            GameObject obj = (GameObject)Instantiate(highway);
            obj.SetActive(false);
            platforms.Add(obj);
        }
    }

   public GameObject GetHighway()
    {
        for (int i = 0; i < platforms.Count; i++)
        {
            if (!platforms[i].activeInHierarchy)
            {
                return platforms[i];
            }
        }
        GameObject obj = (GameObject)Instantiate(highway);
        obj.SetActive(false);
        platforms.Add(obj);
        return obj;
    }
}
```

Рис.2.19 Сценарій PlatformManager

В цьому сценарії створено змінні GameObject highway – змінна в яку поміщається об’єкт платформи та platformAmount – число платформ які будуть генеруватись. Створено масив List з об’єктами платформ.

Сценарій – **PlatformGenerator :**

```
public class PlatformGenerator : MonoBehaviour
{
    public GameObject highway;
    public Transform generatoinPoint;
    public float distanceBetween;

    float platformWidth;

    public float distanceMin;
    public float distanceMax;

    public PlatformManager platformM;
    private CoinGenerator theCoinGenerator;

    public float randomCoinTreshold;

    // Start is called before the first frame update
    void Start()
    {
        platformWidth = highway.GetComponent<BoxCollider2D>().size.x;
        theCoinGenerator = FindObjectOfType<CoinGenerator>();
    }

    // Update is called once per frame
    void Update()
    {
        if(transform.position.x < generatoinPoint.position.x)
        {
            distanceBetween = Random.Range(distanceMin, distanceMax);

            transform.position = new Vector3(transform.position.x + platformWidth + distanceBetween, transform.position.y, transform.position.z);
            // Instantiate(highway, transform.position, transform.rotation);

            GameObject newPlatform = platformM.GetHighway();

            newPlatform.transform.position = transform.position;
            newPlatform.transform.rotation = transform.rotation;
            newPlatform.SetActive(true);

            if (Random.Range(0f, 100f) < randomCoinTreshold)
            {
                theCoinGenerator.SpawnCoins(new Vector3(transform.position.x, transform.position.y + 1f, transform.position.z));
            }
        }
    }
}
```

Рис.2.20 Сценарій PlatformGenerator

Даний сценарій реалізує генерацію платформ. В ньому добавились такі змінні як : generationPoint – точка від якої рахується дистанція генерації нової платформи, distanceBetween – дистанція між платформами, та distanceMin / distanceMax – в них задається мінімальна та максимальна відстань між платформами.

Сценарій – **PlatformDestroyer :**

```
public class PlatformDestroyer : MonoBehaviour
{
    public GameObject destroyPoint;

    // Start is called before the first frame update
    void Start()
    {
        destroyPoint = GameObject.Find("DestroyPoint");
    }

    // Update is called once per frame
    void Update()
    {
        if (transform.position.x < destroyPoint.transform.position.x)

        {
            //Destroy(gameObject);
            gameObject.SetActive(false);

        }
    }
}
```

Рис.2.21 Сценарій PlatformDestroyer

Цей сценарій своєю чергою видаляє платформи коректно, є змінна destroyPoint – точка при проходженні якої об’єкт платформи вимикається в масиві, методом SetActive(false), після цього платформа стає знову доступна для повторної генерації на ігровій сцені.

Генерація монет

Наступна задача полягає в створенні генератора монет, які будуть випадково появлятись на певних платформах. Для цього використані такі сценарії як – objectPooler, CoinGenerator та CoinDestroyer(PlatformDestroyer).

Сценарій **objectPooler *:***

```
public class ObjectPooler : MonoBehaviour
{
    public GameObject pooledObject;
    public int pooledmAmount;

    List<GameObject> pooledObjects;

    // Start is called before the first frame update
    void Start()
    {
        pooledObjects = new List<GameObject>();
        for (int i = 0; i < pooledmAmount; i++)
        {
            GameObject obj = (GameObject)Instantiate(pooledObject);
            obj.SetActive(false);
            pooledObjects.Add(obj);
        }
    }

    public GameObject GetPooledObject()
    {
        for (int i = 0; i < pooledObjects.Count; i++)
        {
            if (!pooledObjects[i].activeInHierarchy)
            {
                return pooledObjects[i];
            }
        }
        GameObject obj = (GameObject)Instantiate(pooledObject);
        obj.SetActive(false);
        pooledObjects.Add(obj);
        return obj;
    }
}
```

Рис.2.22 Сценарій objectPooler

В цьому сценарії створено змінні GameObject pooledObject – змінна в яку поміщається об’єкт який буде генеруватись, в даному випадку це монети та platformAmount – число об’єктів які будуть генеруватись. Створено масив List.

Сценарій **CoinGenerator :**

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

public class CoinGenerator : MonoBehaviour
{
    public ObjectPooler coinPool;

    public float distanceBetweenCoins;

    public void SpawnCoins(Vector3 startPosition)
    {
        GameObject coin1 = coinPool.GetPooledObject();
        coin1.transform.position = new Vector3(startPosition.x, startPosition.y + 1f, startPosition.z);
        coin1.SetActive(true);
    }
}
```

Рис.2.23 Сценарій CoinGenerator

Цей сценарій реалізує генерацію монет на платформах. Змінна ObjectPooler coinPool – сворює масив об’єктів, змінна distanceBetweenCoins – задає відстань між монетами та SpawnCoins саме генерує монети на платформах, дістаючи їх з масиву та роблячи їх активними до поки вони не пройдуть точку видалення.

Для видалення монет використовується той сценарій що і для платформ, так як немає потреби робити новий сценарій для кожного об’єкта.

Колекціонування монет

Так як монети вже генеруються, потрібно зробити так щоб герой взаємодіяв з ними та колекціонував їх і в результаті їх накопичення гравець отримував відповідну винагороду. Для цього створюємо сценарій pickupPoints та додаємо його на об’єкт монетки.

Сценарій **pickupPoints :**

```
public class pickupPoinst : MonoBehaviour
{

    public float scoreToGive;

    private ScoreManager theScoreManager;

    // Start is called before the first frame update
    void Start()
    {
        theScoreManager = FindObjectOfType<ScoreManager>();

    }

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

    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.gameObject.name == "YBBoy")
        {
            theScoreManager.AddScore(scoreToGive);
            gameObject.SetActive(false);
        }
    }
}
```

Рис 2.24 Сценарій pickupPoints

В даному сценарії вводимо нову змінну scoreToGive яка відповідає за цінність монети, цінність задається в налаштуваннях об’єкта(Рис 2.15), також задаємо змінну для доступу до сценарію ScoreManager в який і буде передаватись інформація про зібрані монети. В Start задаємо що змінна theScoreManager це об’єкт типу ScoreManager, Update залишаємо без змін. Далі добавляємо метод OnTriggerEnter2D який спрацьовує при умові, коли герой торкається монетки передається інформація в ScoreManager про те що потрібно додати зібрану монетку на табло результатів(Рис 2.17) і зробити монетку не активною на ігровій сцені.

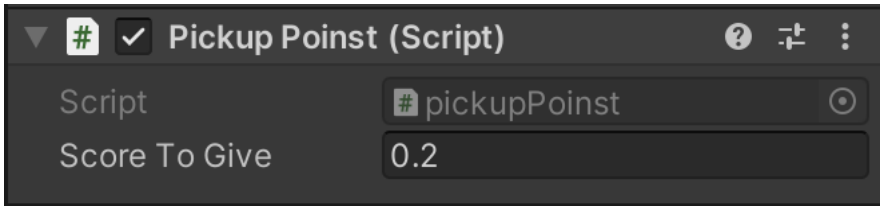


Рис.2.25 Налаштування змінної scoreToGive

Переходимо до сценарію ScoreManager (Рис 2.18) який відповідає за підрахунок монет та вивід їх на табло результату. Для цього була створена змінна Text в яку помістили елемент тексту. В Update прописуємо текст який має виводитись на табло і в цьому тексті вказуємо змінну scoreCount яка приймає значення змінної pointsToAdd, і відповідно кожен раз коли колекціонується монетка ціна якої 0.2, на табло передається ця інформація і змінюється число(Рис 2.17).

Рис 2.26 Табло результатів

```
public class ScoreManager : MonoBehaviour
{
    public Text scoreText;

    public float scoreCount;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        scoreText.text = "ВАША ЗНИЖКА - " + scoreCount.ToString("0.##") + "%";

    }

    public void AddScore(float pointsToAdd)
    {
        scoreCount += pointsToAdd;

    }
```

Рис 2.27 Сценарій ScoreManager

Умова завершення гри

Для того щоб гра мала своє логічне завершення, потрібно створити відповідні умови. Так як гра платформер, і смерть персонажа наступає тоді коли він падає в ущілину, то відповідно і створити такі умови щоб при попаданні в ушілину герой помирав, і гра закінчувалась. Для цього потрібно створити такий об’єкт, при доторканні якого герой помирає, для цього було створено “Killer” – платформа при

попаданні на яку герой помирає, для неї було застосовано BoxColider2D та присвоєно тег – “killbox”.

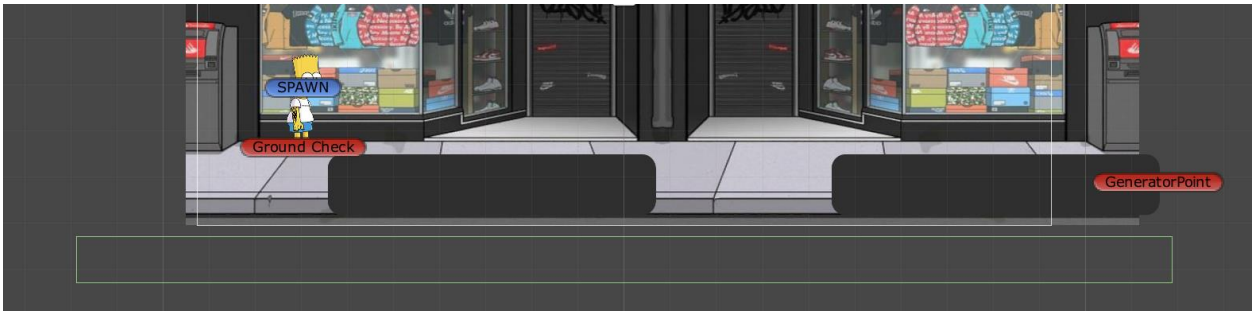


Рис 2.28 Платформа – “Killer”

Рис 2.29 Тег “killbox”

Цей тег застосовується в подальшому сценарії, для спрощення взаємодії з об’єктом.

```
void OnCollisionEnter2D(Collision2D other)
{
    if (other.gameObject.tag == "killbox")
    {
        theGameManager.RestartGame();
        speed = speedStore;
        speedMilestoneCount = speedMilestoneCountStore;
        speedIncreaseMilestone = speedIncreaseMilestoneStore;
    }
}
```

Рис 2.30 Сценарій смерті

В даному сценарії створено умову, що при взаємодії героя з об’єктом тег якого “killbox”, то виклакається функція RestartGame для перезапуску рівня, а також швидкість героя стає початковою.

Створення меню смерті

Для того щоб гра могла завершуватись, потрібно створити так зване меню смерті, табло яке буде активовуватись коли герой падає в щілину і відповідно програє. Для цього створено графічний елемент, звичайний екран чорного кольору з прозорісттю. Для активації цього екрану створюємо сценарій DeathMenu основна функція якого зв'язок з сценарієм GameManager, та поміщаємо його на графічний елемент екрану. Наступний крок, це створення самого перезапуску гри та виводу

екрану смерті на ігрову сцену, так як за замовчуванням він не відображається. Це все створюється в сценарії GameManager(Рис 2.20) в RestartGame прописуються умови перезапуску гри, тобто при смерті гравець набуває значення false, тобто не відображається, а екран смерті набуває значення true, та відображається на сцені до поки ми не нажмемо кнопку Restart(Рис 2.21) в меню та цим самим викликаємо функцію Reset яка перезавантажить гру та поверне все на початкові позиції, а саме вимкне меню смерті, відновить початкові платформи та активує персонажа на його початкових позиціях.

```
public class DeathMenu : MonoBehaviour
{
    public void RestartGame()
    {
        FindObjectOfType<GameManager>().Reset();

    }

}
```

Рис 2.31 Сценарій DeathMenu

```
public void RestartGame ()
{
    theScoreManager.scoreCount = 0;
    theYBBoy.gameObject.SetActive(false);
    //StartCoroutine("RestartGameCo");

    theDeathScreen.gameObject.SetActive(true);
}


public void Reset()
{
    theDeathScreen.gameObject.SetActive(false);
    //theYBBoy.gameObject.SetActive(false);
    platformList = FindObjectsOfType<PlatformDestroyer>();
    for (int i = 0; i < platformList.Length; i++)
    {
        platformList[i].gameObject.SetActive(false);
    }
    theYBBoy.transform.position = playerStartPoint;
    platformGenerator.position = platformStartPoint;
    theYBBoy.gameObject.SetActive(true);

    //theScoreManager.scoreCount = 0;
}
```

Рис 2.32 Сценарій GameManager

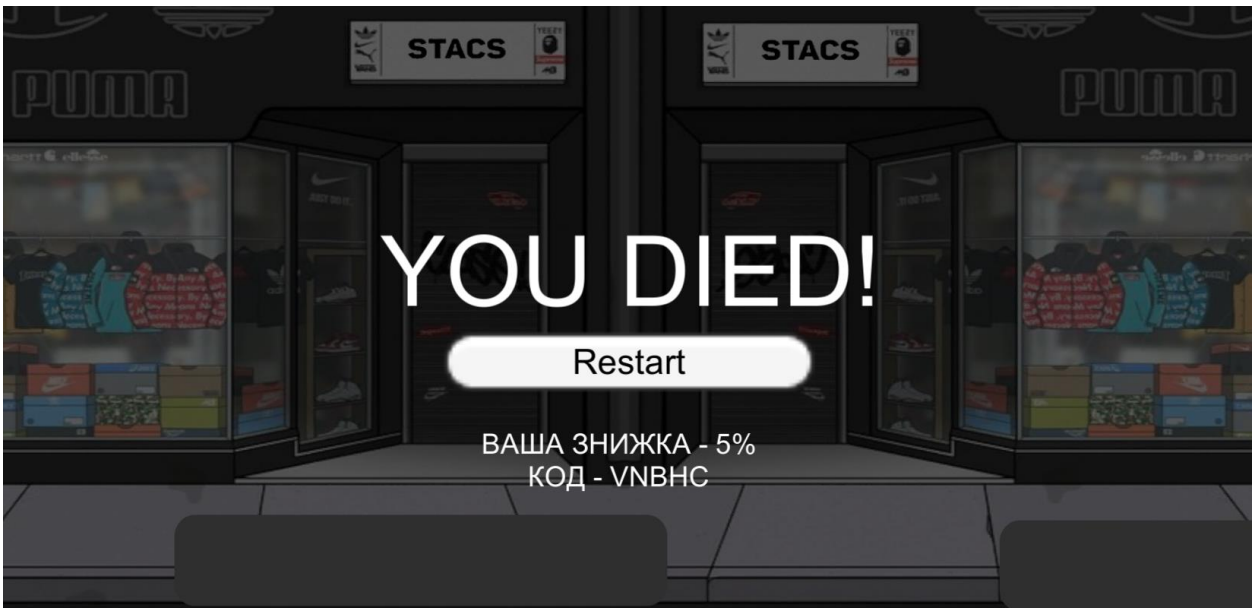


Рис 2.33 Меню смерті

Висновок до розділу 2

Цей розділ присвячено аналізу та вибору середовища розробки та реалізації проекту. Було проаналізовано три середовища та вибрано те яке найбільше підходило, а саме Unity. Реалізація гри почалась з підбору графічних елементів а також поясненню понять які стосувались графічної складової. Були вибрані спрайти які використовувались в розробці проекту.

Розділ містить в собі пояснення етапів створення та сценаріїв. Основну частину розділу відведено на розбір найважливіших сценаріїв таких як рух персонажа, генерації платформ та монет, докладно описано кожен з них.

Було створено графічний інтерфейс який показує кількість зібраних монет та їхню цінність, також створено меню смерті, яке показує результат та має в собі такі елементи як клавіші.

РОЗДІЛ 3
АНАЛІЗ ПРОЕКТУ

3.1 Актуальність проекту

Останнім часом спостерігається тенденція спаду попиту на звичайні рекламні стратегії для просування продуктів як і в мережі інтернет так і в оффлайн варіантах. Суспільство перестало звертати увагу на банальні банери, нудні шаблонні рекламні вивіски і тому подібне. Рекламодавці стараються шукати більш цікаві рішення, наприклад такі як електронні банери на яких демонструють свою рекламу. Тут і відкривається цінність цього проекту, так як люди вже не звертають увагу на банальні речі, висувається на ринок саме такий проект який є цікавим та інтерактивним рішенням для заохочення клієнтів до взаємодії, в виді здавалося б простої гри. Таке рішення заставляє клієнта зацікавитись в отриманні результату, а саме знижки, та в її подальшому використанні, тим самим зробивши покупку, та запам’ятавши даний процес не як звичайну покупку, а покупку з цікавим та особливим елементом гри.

3.2 Застосування проекту

Основна мета проекту – це застосування його в маркетингових цілях для заохочення нових клієнтів до покупок.

Інтеграція проекту буде в онлайн instagram – магазин одягу STACS STORE. Гра буде застосовуватись для надання вітальної знижки для нових клієнтів, гравець отримає три спроби здобути знижку на свою першу покупку. Знижка буде варіюватись від 0 до 5%. Відповідно отримавши знижку, в людини виникне бажання щось купити застосувавши її. На початкових етапах гра буде відображатись як посилання в описі профілю а також в інстаграм-сторіс. При переході за посилання гравець попадає на веб сторінку де й розпочинається ігровий процес.

В подальшому планується інтеграція з більшою кількістю тих чи інших магазинів які будуть зацікавлені в заохочені клієнтів до купівлі продукту саме таким інтерактивним методом.

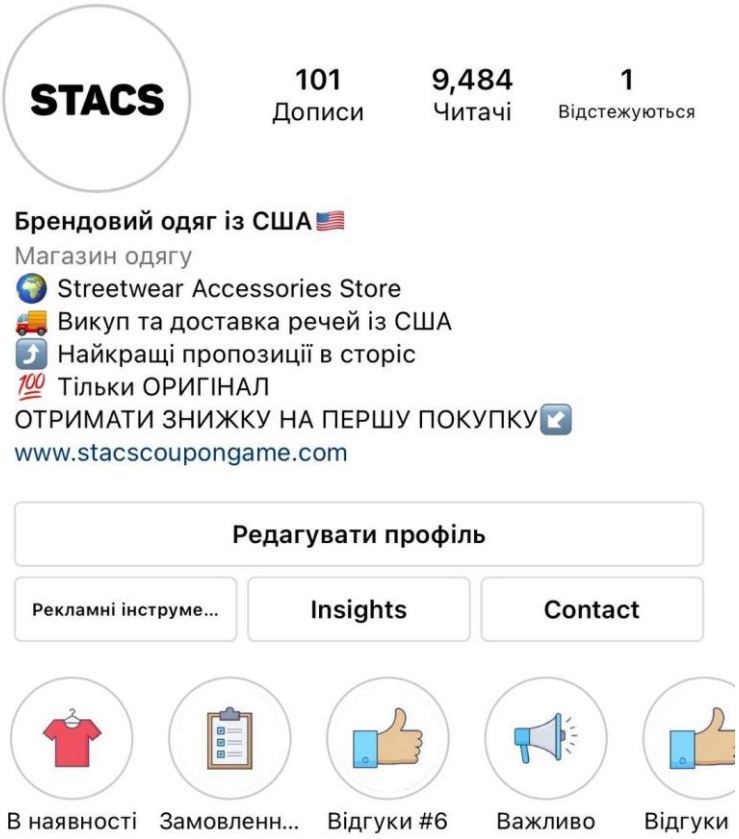
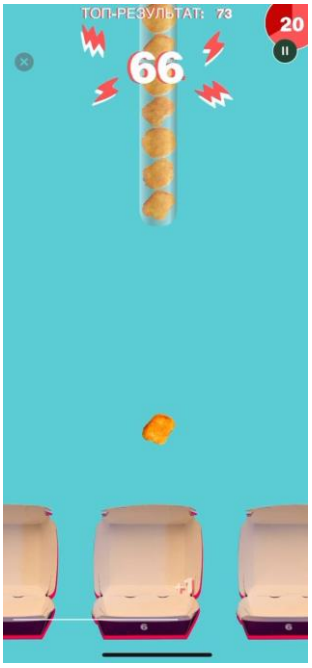


Рис 3.1 Відображення посилання на гру в профілі Instagram

3.3 Аналіз конкурентів

Вивчаючи подібні продукти, я натрапив тільки на один подібний варіант отримання купонів, від ресторану McDonalds.



3.2 Гра від McDonalds

Дана гра заключається в заповненні пакунків харчами, в верхній частині є резервуар з якого випадає продукт при натисканні на екран. Задача полягає у влучанні продуктом в коробку, за це ми отримуємо бали, які в результаті формують купон на знижку.

Конкуренції в даній категорії одиниці, ринок вільний для нових ідей та їхнього застосовування.

Висновок до розділу 3

В даному розділі було проведено аналіз актуальності проекту, в результаті якого проект можна вважається актуальним. Також показано застосування на практиці, а саме в інтернет-магазині. І на завершення було проведено пошук та аналіз конкурентів, в результаті чого приходить розуміння того що дана категорія ігрових продуктів є пустою, подібних продуктів на даний момент є не багато, і цим самим розробники таких продуктів мають можливість уникнути конкуренції.

ВИСНОВОК

На початку розробки був проведений аналіз доцільності використання даного жанру гри в маркетингових цілях. Після чого було прийняте рішення про розробку ігрового додатку через високу актуальність подібних проектів на сьогоднішній день.

Перед початком розробки було проаналізовано кілька ігрових рушіїв та вибрано той який найбільше підходить під дану задачу. Також були описані існуючі подібні проекти.

Було вибрано рушій Unity, що ідеально підходив під всі задачі, та був використаний для подальшої розробки. Це рушій на якому щодня створюють нові проекти, його використовують як і досвідченні розробники так і новачки, через простоту та інтуїтивність його інтерфейсу.

Для написання коду було використано мову C#. Вона чудово взаємодіє з Unity і завдяки їй були реалізовані все сценарії проекту. Спочатку було розроблені: переміщення персонажа та слідкування камери, пізніше генерація рівня та монеток, і звісно ж закінчення рівня.

Для створення 2D графіки використано Adobe Photoshop. За допомогою нього був створений фон, персонаж та монетки, інтерфейс гри.

Результатом проекту є ігровий додаток під назвою “Stacs Coupon” у жанрі Платформер, який має простий та інтуїтивний інтерфейс, зрозуміле управління та результат проходження гри. Даний проект буде використаний в маркетингових цілях як і було зазначено в меті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Комп'ютерна гра [Електронний ресурс] – Режим доступу до ресурсу: https:// uk.wikipedia.org/wiki/PC_game.
2. Жанр Платформер [Електронний ресурс] – Режим доступу до ресурсу: uk.wikipedia.org/wiki/Platform_game.
3. Інді-ігри [Електронний ресурс] – Режим доступу до ресурсу: uk.wikipedia.org/wiki/інді-ігри.
4. Історія розвитку комп’ютерних ігор [Електронний ресурс] – Режим доступу до ресурсу: http://maptimes.inf.ua/PublicUa/A25HistoryOfGames.html.
5. Комп’ютерна гра та авторське право [Електронний ресурс] — https://blog.liga.net/user/amyisenko/article/38093.
6. Які є різновиди ігор [Електронний ресурс] — https://genomukr.com/igri/22910-jaki-e-riznovidi-onlajn-igor.html.
7. Розробка комп’ютерних ігор [Електронний ресурс] — https://ukrbukva.net/89526-Razrabotka-komp-yuternyh-igr.html.
8. Комп’ютерна гра [Електронний ресурс] — https://esu.com.ua/search_articles.php?id=4393.
9. Відеоігри з точки зору науки [Електронний ресурс] — https://geektech.me/відеоігри-з-точки-зору-науки.
10. Fortnite [Електронний ресурс] — https://uk.wikipedia.org/wiki/Fortnite.
11. Cuphead[Електронний ресурс] — https://uk.wikipedia.org/wiki/Cuphead.
12. Anthill[Електронний ресурс] — https://uk.wikipedia.org/wiki/Anthill.
13. Unreal Engine [Електронний ресурс] – Режим доступу до ресурсу: https://www.unrealengine.com/
14. Unity Game Engine [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com/ru.
15. Unity Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/index.html.

16. Godot Game Engine [Електронний ресурс] – Режим доступу до ресурсу: https://godotengine.org/.
17. Unity Endless Runner Tutorial [Електронний ресурс] – Режим доступу до ресурсу:https://www.youtube.com/watch?v=GrQalFLtQT4&list=PLiyfvmtjWC_XmdYfXm2i1AQ3lKrEPgc9-
18. Класифікація ігор та ігрових жанрів [Електронний ресурс] – https://infodef.github.io/html_submenu/game_ua.html