ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Шкорута Владислав Романович

УДК 004.0

Створення відеогри в жанрі Platformer

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації бакалавр

Нормоконтроль	Студент
_____Стисло О.В.	_____Шкорута В.Р.
(підпис, дата, розшифрування підпису)	(підпис, дата, розшифрування підпису)
Допускається до захисту	Керівник роботи
Завідувач кафедри	
_____к.т.н., доц. Ващишак С.П.	_____ к.т.н., доц. Ващишак С.П
(підпис, дата, розшифрування підпису)	(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

«____» _____ 2024 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Шкорута Владислав Романович

(прізвище, ім’я, по батькові)

1. Тема кваліфікаційної роботи:

Створення відеогри в жанрі Platformer

керівник роботи:

Ващишак Сергій Петрович, к.т.н., доцент

затверджена наказом вищого навчального закладу від « 12» березня 2023 року № 19/1

2. Термін подання студентом роботи 05.06.2023

3. Вихідні дані роботи: мова програмування C# та ігровий рушій Unity

4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)

1. Аналіз конкурентів в обраному жанрі та індустрії
2. Дослідження ігрових рушіїв та обґрунтування вибору
3. Обір доповнювальних компонентів гри таких як візуал та аудіо
4. Імплементація попередніх компонентів та створення скриптів для гри

5. Дата видачі завдання 14.03.2024

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз конкурентів в обраному жанрі та індустрії	22.03.2024	Виконано
2.	Дослідження ігрових рушіїв та обґрунтування вибору	29.03.2024	Виконано
3.	Обір доповнювальних компонентів гри таких як візуал та аудіо	12.04.2024	Виконано
4.	Імплементація попередніх компонентів та створення скриптів для гри	29.04.2024	Виконано
5.	Формування висновків	09.05.2024	Виконано
6.	Оформлення графічного матеріалу та підготовка до захисту роботи	11.05.2024	Виконано

Студент _____ Шкорута Р.В..
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ващишак С.П.
(підпис) (прізвище та ініціали)

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Сторінка	Опис графічного матеріалу	Сторінка	Опис графічного матеріалу
12	Статистика користувачів платформи “Steam”	30	Приклад стилістики гри Monument Valley
14	Стилістика та зразок ігрового процесу Hollow Knight	31	Приклад гри Thomas Was Alone та геометричного візуального ігрового стилю
15	Карта світу Hollow Knight	32	Піксельний стиль на прикладі гри Stardew Valley
16	Художній стиль Ori And The Blind Forest	49	Візуальний дисплей очок здоров'я гравця
18	Приклад ігрового процесу стилістики та рівнів Dead Cells	53	Меню кінця гри

АНОТАЦІЯ

Кваліфікаційна робота присвячена вивченню, аналізу та створенню відеоігор, а саме гри 2Д платформера, створеної в розважальних цілях з можливістю подальшої публікації та навіть монетизації.

В першому розділі проаналізовано ігри як хобі та індустрію, в якій людина може поглинути та провести вільний час, або навіть знайти робочі можливості. Також проаналізовано конкурентів в вибрано жанрі, обрано недоліки над якими можна попрацювати та вдосконалити користувацький досвід.

В другому розділі досліджено та обґрунтовано вибір ігрового рушія серед декількох альтернатив. Також переглянуто графічний вибір інших ігор, та описано коректний вибір і як він впливає на загальний час потрібний для завершення гри.

В третьому розділі описано переваги використання Unity Asset Store та чому він був застосований для цієї кваліфікаційної роботи. Описана програмна складова проекту та як вона впливає на кінцевий результат.

КЛЮЧОВІ СЛОВА: UNITY, 2Д ПЛАТФОРМЕР, СКРИПТ, АССЕТ.

SUMMARY

The qualification work is devoted to the study, analysis and creation of video games, as well as a 2D platformer game created for entertainment purposes with the possibility of further publication and even monetization.

The first chapter analyzed games as a hobby and an industry in which a person can absorb and spend free time, or even find work opportunities. Competitors in the selected genre were also analyzed, shortcomings were selected that could be worked on and the user experience improved.

In the second chapter, the choice of a game engine among several alternatives is investigated and justified. The graphical choices of other games are also reviewed, and the correct choices are described and how they affect the overall time required to complete a video game.

The third section describes the benefits of using the Unity Asset Store and why it was used for this qualification. The software component of the project is described and how it affects the final result.

KEY WORDS: UNITY, 2D PLATFORMER, SCRIPT, ASSET.

ЗМІСТ

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

Геймдев (від game dev) – створення гри від планування до випуску, може бути зайнятим як одна людина так і велика команда.

Локація (від location) – певне середовище яким може подорожувати ігровий персонаж.

Roguelike – жанр ігор характерний частою смертю користувача та початку гри з самого початку.

Ассет – скорочення для всього, що входить у відеогру – персонажів, об’єктів, звукових ефектів, карт, оточення тощо.

Ігрова механіка – правила по якими користувач проходить гру.

Випікання – назва процесу щодо збереження інформації, пов’язаної з 3D-сіткою, у файл текстури (бітове зображення).

ВСТУП

Актуальність теми: з кожним роком відеоігри стають популярнішими та популярнішими і б'ють рекорди по кількістю користувачів в медіумі та грошей які в нього поступають. Відповідно до інтересного звіту веб-сайту SteamDB, що збирає дані, понад 14 500 ігор знайшли свій шлях на платформу, що становить у середньому приголомшливі 50 випусків на день. У 2023 році дохід ігрової індустрії прогнозується на рівні 365,6 мільярдів доларів у всьому світі. Багато людей відкривають його для себе як альтернативи книгам з сторони розповіді історії а деякі вибирають ігри які допомагають розслабитись після важкого дня.

Геймдев складна та творча індустрія яка дозволяє людині яка бере в ній участь розкритися з різних сторін. Дизайнери, програмісти, художники і т.п. абсолютно необхідні для створення конкретного кінцевого продукту.

Мета і завдання дослідження: розробити відеогру в жанрі Platformer використовуючи ігровий рушій.

Для досягнення поставленої мети вирішуються наступні завдання:

- провести аналіз аналогів в обраному жанрі та різних методів реалізації завдання;
- розробити дизайн гри;
- реалізувати ідеї в вибраному рушії;
- провести тестування розробленого продукту.

Об’єкт дослідження: процес створення відеогри в жанрі Platformer використовуючи ігровий рушій.

Предмет дослідження: методи та інструменти які можуть бути використані при створенні відеоігор, в даному випадку рушій Unity та мова програмування C#.

Методи дослідження. Для досягнення поставленої мети та вирішення завдань дослідження будуть використані наступні методи:

– метод аналізу – для проведення аналізу сучасних методів та інструментів геймдеву, а також аналізу результатів роботи гри та її відповідності до аналогів;

– метод порівняння – вивчення аналогів конкурентів вибраного предмету та вибір успішних методик з них для імплементування;

– метод експерименту – створення гри на основі аналізу;

– метод тестування – знаходження багів це необхідний крок перед випуском функцій в відеоіграх.

Ці методи дослідження допоможуть ретельно вивчити обраний предмет, розробити інтересну відео гру з використанням рушія Unity та C#, а також провести тестування для його ефективної роботи.

Практичне значення одержаних результатів: результатом даної кваліфікаційної роботи є функціональна гра з мінімальною кількістю багів в жанрі платформер за якою гравець може провести вечір чи два з задоволенням. Розроблений проект не є кінцевим і може бути дороблений і в подальшому випущений на онлайн платформи.

Апробація результатів дослідження. Матеріали кваліфікаційної роботи були представлені на XI Міжнародній науковій конференції «Студентські наукові дискусії поза форматом», яка відбулася 11 квітня 2024 року в Університеті Короля Данила.

Структура: Розділи - 3, загальний обсяг основної частини - 52, кількість використаних джерел - 20.

РОЗДІЛ 1. ОПИС ОБ'ЄКТУ ТА ПОРІВНЯННЯ АНАЛОГІВ

1.1 Опис відеоігор як один з способів провести вільний час

Відеоігри - це явище, яке здобуло широку популярність та стало не просто формою розваги, але й важливим аспектом сучасної культури. Їхнє значення як способу проведення вільного часу важко переоцінити, оскільки вони забезпечують гравцям можливість відчути емоційне занурення, розвивати навички та спілкуватися з іншими.

На першому рівні відеоігри виконують роль розваги. Вони створюють можливість для гравців відпочити, розслабитися та відірватися від повсякденних проблем. Відповідно до власних уподобань, гравець може обрати ігри різних жанрів, від пригодницьких та екшенів до головоломок та симуляторів, що забезпечує різноманіття відчуттів та емоційних вражень.

Однак, важливо враховувати, що відеоігри не лише забавляють, але й сприяють розвитку різних навичок. Наприклад, багато ігор вимагають від гравців стратегічного мислення, швидкісної реакції та координації рухів. Це може сприяти розвитку когнітивних здібностей та покращенню рішення проблем у реальному житті.

Соціальний аспект відеоігор також надзвичайно важливий. Багато ігор пропонують можливість грати з друзями або навіть знайомитися з новими людьми онлайн. Це створює можливість для спілкування та спільної гри, що зміцнює соціальні зв'язки та сприяє формуванню спільнот.

Було доволі не мало дебатів на рахунок позитивних чи негативних наслідків для гравців. У статті 2000 року Американська психологічна асоціація (APA) стверджувала, що їх дослідження показали, що насильницькі відеоігри підвищують агресію серед геймерів. Але CNN повідомила, що останні дослідження спростували ці твердження [1].

Зростаюча популярність відеоігор та ігрових платформ, таких як Steam, дозволила багатьом невеликим компаніям і незалежним творцям зайняти ніші в ігровій спільноті. Їхні ігри зазвичай знаходяться на протилежному кінці спектру від «жорстоких відеоігор», які, на думку деяких експертів і батьків, завдають шкоди молодим гравцям [2].

Тим не менш, ентузіазм щодо ігор має шкалу, яка варіюється від хобі до залежності, здорового до шкідливого. Те, де гравець ставиться на ваги, залежить від рівноваги.

Завелика кількість проведена в віртуальному світі може повпливати на навчання, кар'єру, стосунки і інші інтереси та діяльності. Фізичне та психічне здоров’я гравців може погіршитися, якщо вони пропускають їжу, нехтують фізичними вправами, недосипають та відмовляються від особистої гігієни, тому що хочете продовжувати грати. Багато людей, які страждають від ігрової залежності, також мають слабке психічне здоров’я та відчувають стрес, тривогу та навіть в деяких випадках депресію.

Психологічні розлади пов'язані з інтернет-іграми також мають соціальні наслідки. Ті, хто колись любив грати в ігри з близькими, можуть спробувати приховати свою залежність і відійти від соціальних ситуацій, стати ізольованими та самотніми. Це може призвести до втрати впевненості, низької самооцінки та труднощів у спілкуванні з людьми в реальних ситуаціях. Тому важливо звернути увагу на свою поведінку в іграх і перевірити, чи вона перетворилася з розслаблюючого хобі на всепоглинаючу залежність [3].

З кожним роком все більше і більше людей вибирають відеоігри як спосіб послухати історію або відчути себе як її головний герой. За статистикою найпопулярнішої ігрової платформи під назвою “Steam” від 2015 року до 2023 найбільша кількість користувачів які використовували платформу зросла на 400% з 8.4 мільйонів користувачів до 33 мільйонів користувачів (рис. 1.1). І це найбільша кількість одночасно в мережі, якщо взяти статистику по місячно то вона зросла від 60 мільйонів користувачів на місяць до понад 120 [4].

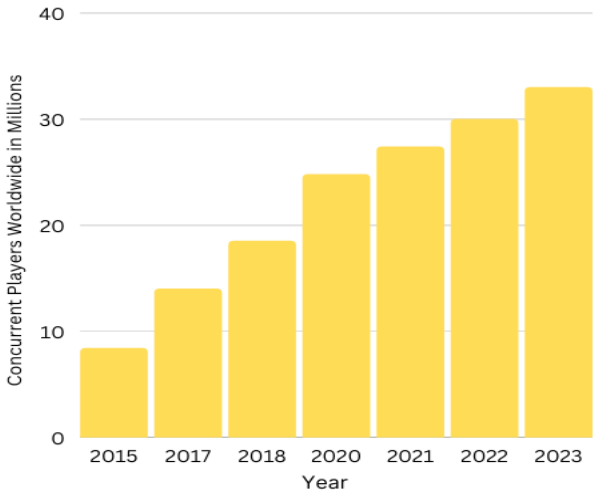


Рисунок 1.1 – Статистика користувачів платформи “Steam”

Загалом, відеоігри представляють собою комплексне явище, яке впливає на різні аспекти життя людини. Вони не просто забавляють, але й надають можливість для розвитку, відпочинку та соціальної взаємодії. Однак, важливо зберігати баланс та не допускати перевантаження відеоіграми, адже, як і з будь-якою формою розваги, зловживання може мати негативні наслідки.

1.2 Аналіз конкурентів в жанрі

Аналіз - ділення повного комплексного явища на менші більш прості частини і виділення їх зв'язків та властивостей. Проте він не є остаточною метою цього дослідження. В цьому розділі я б хотів розбити конкурентів на певні категорії і переглянути як їх поєднання дозволило їм досягти успіху в випуску їх продуктів.

Аналітичний метод – інструмент скрупульозного дослідження особливостей і специфіки внутрішньосистемної взаємодії, і він без сумніву вміщує в собі результати абстрагування, спрощення, формалізації. Просто все це не самоціль, сутнісне завдання аналітичного методу полягає в тому, що він спрямований на виявлення внутрішніх тенденцій та можливостей розвитку певного обраного об’єкта.

Не всі ігри в індустрії є оригінальними творіннями створеними з нуля де всі концепції були придумані власноруч. Можна навіть сказати що більшість ігор не є оригінальними і беруть ідеї в своїх конкурентів. І це не є поганим методом, на мою думку, беручи процеси, які вже працюють в інших іграх, і добавляючи до них свої наробки, багато розробників дійшли до створення проектів які сколихнули світ геймдеву.

Прикладом можливо привести цілий жанр який був створений з допомогою поєднання двох ігор під назвою Метроідванія(Metroidvania**)** [5]. Ігри Metroid та Castlevania були, зачасту, характерними 2д виглядом з пересуванням по платформам, ігровими середовищами по яким гравець подорожує та здобуває нові предмети та здібності, які в свою чергу дозволяють відкрити нові частини середовища для подальшого подорожування.

В свою чергу хорошими прикладами цього жанру є Hollow Knight, Ori And The Blind Forest та Dead Cells**.** Повністю описати ці ігри можливо проте це займе забагато часу і можливо буде не повністю доцільним. Тому в цьому підрозділі буде описано що розробники даних відеоігор зробили та імплементували в своїх проектах і заодно обрано аспект який, на мою думку, був або недороблений або не захоплюючий.

Hollow Knight - пригодницький бойовик платформер розроблений та випущений командою Team Cherry. Розробка була частково профінансована за допомогою краудфандингової кампанії Kickstarter, зібравши понад 57 000 австралійських доларів до кінця 2014 року [6].

Намальовані від руки ескізи Арі Гібсона сканувалися безпосередньо в ігровий двигун, допомагаючи створити яскраве відчуття місця (рис 1.2). «Я їх сфотографував на телефон і почистив на комп’ютері. З цього моменту можна просто додавати фігури у світ, поки він не буде відчуватися повним», — сказав він минулого року Game Informer. «Нашою візуальною метою було «Бути простим». Ця мантра поширилася на решту стилю, ймовірно, була невід’ємною частиною нашої можливості створити такий великий світ за декілька років».

Рисунок 1.2 – Стилістика та зразок ігрового процесу Hollow Knight

Гравець керує жуко-подібним, німим та безіменним персонажем, якого користувачі назвали “Лицар”, і подорожує підземними локаціями. Лицар користується “цвяхом”, своєрідним аналогом меча, для бою або для взаємодії з середовищем (відкриттям таємних проходів, користування перемикачами і т.д.).

В більшості зон гри, гравці зустрінуться ворожих жуків та інших істот. Ближній бій зазвичай зводиться до використання цвяху на короткій дистанції. Гравець може вивчити різні заклинання, які дають лицарю можливість розібратися з ворогами на дальній дистанції. З переможених ворогів випадає ігрова валюта під назвою “Джіо”, яку можна використати в спеціальних магазинах. Лицар на початку гри має 5 Масок, які є ігровою відповідністю очок здоров'я. Уламки масок збираються під час проходження гри для збільшення кількості активних масок. При полученні удару від вороги у гравця знімається одна або, в деяких випадках, дві маски. Для збереження гри використовуються Лавки. Якщо користувач помирає то він переноситься до неї.

Під час гри гравці отримують предмети, які дають нові здібності руху, як-от додатковий стрибок у повітрі, прилипання до стін і стрибки з них, короткий ривок і тому подібні речі. Це дозволяє користувачам подорожувати світом гри в доволі неочікуваними шляхами.

Складність світу базується на Metroid, який дозволяє гравцям заблукати, зосередившись на насолоді від пошуку шляху. По всьому світу (рис 1.3) розміщуються лише основні покажчики, які спрямовують гравців до важливих місць. Найбільшою складністю дизайну гри було створення системи відображення та знаходження балансу між тим, щоб не розголошувати світові таємниці та не бути надто непривітним до гравців.

Рисунок 1.3 – Карта світу Hollow Knight

І тут полягає моя критика - карта занадто складна і непривітна. Team Cherry постарались і забезпечили приємну пригоду для тих хто справді хоче в неї заглибитись. Проте тим самим вони не допомогли новачкам як загалом в іграх так і саме в Hollow Knight.

З самого початку відкриті декілька шляхів для гравців, які не знаючі користувачі сприймуть за коректний шлях для продовження історії але цей шлях буде заблокований ворогом або на відстані стрибку. І цього ворога не можливо перемогти без певного заклинання, або цей стрибок на відстані, більшій за довжину стандартного стрибку і потребує якоїсь модифікації або можливо правильний вихід на іншій частині локації.

Для того же самого типу гравця, якого дуже легко дизорієнтувати, проблема продовжується і погіршується. Карта відкривається більше і більше, способи пересування також і локації не зменшуються. Комбінація цих факторів разом може перевантажити досвід користувача.

Ori And The Blind Forest - пригодницький платформер метроідванія 2015 року створена командою Moon Studios та випущено Microsoft Studios. Гравці керують Ori, маленький білий дух, та Сайном, “очима та світлом” Деревом Духів Лісу [7]. Гравцям доручено переміщатися між платформами та розгадувати головоломки для проходження на подальші локації.

Багатий художній стиль (рис 1.4) змовляється проти гравця: небезпеки не перебільшені та очевидні, а змішуються, як природні особливості, і ви, швидше за все, наткнетеся на скупчення смертоносних колючих наростів, коли наткнетеся на кропиву в саду свого дитинства додому.

Рисунок 1.4 – Художній стиль Ori And The Blind Forest

З самого початку від гравця потребується точність і увага до деталей. Ледачий стрибок не досягне мети автоматично; вам потрібно вдосконалити час і вивчити дугу руху. Крім того, ваші обмеження не відразу очевидні – ви можете

не відразу усвідомити, що деякі області мають бути недоступними, доки Орі не навчиться новому навику.

Як Орі, гравці повинні стрибати, лазити та використовувати різноманітні інші здібності для навігації у світі гри. В свою чергу Сейн може стріляти Духовим Вогнем, щоб боротися з ворогами або ламати перешкоди. І тут і криється проблема цієї гри.

До аналізу різних статей одна проблема гри була зрозуміла, але після цього вона б'ється в очі ще ріще. При описі гри люди хвалять прекрасну стилістику, ідеальну імплементацію жанру метроідванія, приємна музика, проте про битви з ворогами речення два і не більше. Причина цьому проста - бої є доволі неприємними в цій грі.

Більшість зустрічей з ворогами закінчуються після нажаття однієї і тої самої кнопки певну кількість разів. Чим сильніше ворог тим більше разів треба нажати кнопку. Результатом цього є неприємна бойова система, яку хочеться уникати по любій можливості.

Dead Cells - roguelike метроідванія створена командами Motion Twin та Evil Empire, і випущена Motion Twin. Вони черпали натхнення з класу Інженер з іншої гри “Team Fortress 2” і переробили Dead Cells на екшн-платформер, де гравець має використовувати різноманітні комбінації зброї та навичок.

Гравець керує В'язнем, аморфною істотою, яка мандрує островом, повним мутованих монстрів. Коли гравець помирає, він втрачає всю зброю та покращення, отримані під час проходження, за винятком кількох постійних предметів. Зброя насамперед включає мечі, луки, щити та пастки, які можна розставити, які шкодять ворогам, які наближаються до них [8].

Під час бою В'язнень може ухилятися по землі, щоб уникнути атак ворогів, або перестрибувати через атаки. Ухиляючись у простір ворога, В'язнень получає змогу проходити крізь них і атакувати ззаду. Вивчення поведінок ворога є ключовим способом просунутись на настіпні рівні.

Досліджуючи серію рівнів і борючись із істотами всередині, гравець може збирати ігрову валюту під назвою «Клітини» від переможених ворогів. Клітини

можна використовувати для придбання постійних покращень, таких як зілля, що відновлюють очки життя, або додаткову зброю, яку можна одержати випадковим чином під час проходження. Ці клітини можна витратити лише в кінці частини підземелля; якщо гравець помре раніше, він втрачає всі зібрані клітини. Нові варіанти покращення можна знайти, знаходячи креслення в підземеллях, які потрібно винести з рівня, щоб зібрати [9].

Рівні (рис. 1.5) генеруються процедурно шляхом злиття попередньо розроблених секцій у випадковій конфігурації, створюючи підземелля з багатьма різними розташуваннями ворогів і предметів.

Рисунок 1.5 – Приклад ігрового процесу стилістики та рівнів Dead Cells

Це є вся основна інформація про гру без входу в повні деталі. І все це призводить до комплексного, багатогранного досвіду для гравця, якщо він подолає самий перший бар'єр - інформаційний.

Все з чим користувач може взаємодіяти на початку гри йому показано і пояснено. Раз за разом. І результатом цього є 10-20 різних вікон з інформацією яка безпосередньо впливає на досвід який гравець получить протягом партії.

На початку цього підрозділу було описано аналіз та як його можна використати для вивчення об'єктів дослідження. Синтез, метод який буде

використаний в наступних розділах, полягає в тому аби з'єднати складні елементи, які були до того проаналізовані та оброблені.

1.3 Постановка задачі

Основною задачею даної кваліфікаційної роботи є створення відеогри в жанрі платформер з допомогою рушія Unity та мовою програмування C#.

Специфічні задачі, які потрібно вирішити в процесі виконання цієї роботи, включають:

- аналіз вимог до функціональності гри, включаючи огляд існуючих аналогів і визначення їхніх сильних і слабких сторін;
- аналіз рушіїв та їх відповідних мов програмування;
- проектування гри, створення середовищ для гравців, моделі гравців і також музика;
- розробка скриптів для коректного працювання гри;
- тестування та налагодження гри для знаходження багів;
- виставлення гри на онлайн платформі.

Висновки до розділу 1

У даному розділі було досліджено предметну область проекту - світ відеоігор та їх найбільш промінентні учасники. Було проведено детальний огляд та аналіз трьох цих учасників і до того ж аналогами кваліфікаційного проекту, переглянуті їх сильні та слабкі сторони. В ході цього аналізу стали зрозумілі наступні речі:

1. Найкращий ігровий процес створений при комбінації уже існуючих продуманих ідей і власних.
2. Не всі ігри ідеальні і підходять усім. При створенні відеоігор потрібно цілитись на специфічну аудиторію.

3. Створення плавного введення користувача в ігровий процес є самим першим пріоритетом.

На основі проведеного аналізу було сформульовано основну задачу кваліфікаційної роботи - створення відеогри в жанрі платформер з допомогою рушія Unity та мови програмування C#. Специфічні підзадачі, з якими можливо стикнутися в цьому процесі, були також описані.

Цей розділ лягає в основу подальшого дослідження та розробки, надаючи чітке розуміння потреб користувачів та вимог до функціональності відеогри.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ФУНКЦІОНАЛУ ТА ПЛАН РОЗРОБКИ ГРИ

2.1 Ігрові двигуни та мови програмування

При виборі ігрового двигуна є багато різних елементів на які варто звернути увагу. Вибір ігрового двигуна є одним із найважливіших рішень, які ви повинні прийняти як розробник ігор. Ігровий двигун — це програмна основа, яка надає основні функції, інструменти та ресурси для створення, тестування та випуску ігор. Різні ігрові двигуни мають різні сильні та слабкі сторони, функції та вартість, тому необхідно знайти той, який найкраще відповідає заданим потребам, цілям і бюджету [10]. У цьому підрозділі буде надано основні критерії при виборі ігрового двигуна:

1. Жанр та платформа випуску.

Перше, про що потрібно б подумати, це якого жанру гру треба створити та для якої платформи. Різні жанри ігор мають різні вимоги та очікування, як-от графіка, фізика, звук, геймплей та інтерфейс користувача.

Наприклад, якщо для створення 3D-пригодницької гри, може знадобитися ігровий двигун, який підтримує високоякісну візуалізацію, реалістичну фізику, динамічне освітлення та складну анімацію. З іншого боку, якщо натомість є потреба створити 2D-гру-головоломку, в цьому випадку знадобитися ігровий механізм, який підтримує простий дизайн рівнів, логічні сценарії та сумісність між платформами.

В тому ж числі необхідно розглянути, на яку платформу націлена гра, наприклад ПК, консоль, мобільний пристрій або Інтернет. Різні платформи мають різні специфікації, обмеження та методи розповсюдження, тому потрібно вибрати ігровий двигун, який підтримує потрібну платформу, або може легко експортувати вашу гру на кілька платформ.

2. Рівень навичок та розмір команди.

Наступне, про вам потрібно подумати, це рівень навичок та розмір команди при створенні даного проекту. Різні ігрові двигуни мають різні криві навчання, документацію та підтримку спільноти. Деякі ігрові двигуни більш зручні для початківців та інтуїтивно зрозумілі, тоді як інші є більш просунутими та складнішими. Деякі ігрові двигуни мають більш вичерпні та доступні підручники, посібники та форуми, тоді як інші мають більш рідкісні та незрозумілі ресурси.

Розробникам потрібно вибрати ігровий двигун, який відповідає їх рівню навичок і стилю навчання, а також рівню навичок і ролям членів їх команди, якщо вона існує. Наприклад, якщо ви соло розробник із невеликим досвідом програмування або без нього, вам може знадобитися ігровий двигун із системою візуальних скриптів, інтерфейсом з перетягуванням елементів та численними готовими ресурсами та шаблонами. З іншого боку, якщо ви є частиною великої та досвідченої команди з кількома програмістами, художниками, дизайнерами та тестувальниками, вам може знадобитися ігровий двигун із потужною мовою сценаріїв, гнучким редактором і надійним конвеєром ресурсів.

3. Бюджет і часові рамки.

Останнє, про що потрібно думати, це ваш бюджет і часові рамки. Різні ігрові двигуни мають різні цінові моделі, умови ліцензування та роялті. Деякі ігрові двигуни безкоштовні або з відкритим вихідним кодом, а інші платні або засновані на підписці. Деякі ігрові двигуни не мають або мають низькі роялті, а інші мають високі або змінні роялті. При їх виборі потрібно вибрати той ігровий двигун, який відповідає вашому бюджету та очікуваним прибуткам, а також юридичним і етичним наслідкам використання певного ігрового двигуна. Також потрібно враховувати, скільки часу є для розробки та випуску гри. Різні ігрові двигуни мають різну швидкість розробки продукту, оптимізацію продуктивності та засоби налагодження. Деякі ігрові двигуни швидші та простіші у використанні, тоді як інші повільніші та складніші. Вам потрібно

вибрати ігровий двигун, який дозволить вам досягти бажаної якості та масштабу за доступний час.

Вибір ігрового двигуна не є простим чи однозначним завданням. Це вимагає багато досліджень, порівнянь та експериментів. Однак дотримуючись процесу, описаного в цій статті, ви зможете звузити вибір і знайти ігровий двигун, який найкраще підходить для вас і вашої гри.

Тепер взявши всі ці аргументи в думки необхідно подумати яке програмне забезпечення найбільш підходить для цього завдання. Жанром є 2D платформер тому не є необхідним найновіша програма яка підтримує найліпшу високоякісну візуалізацію, реалістичну фізику, динамічне освітлення та складну анімацію. Гра також буде націлена на аудиторію з персональними комп'ютерами, ця функція присутня в більшості двигунів.

Рівень навичок - мінімальний, розмір команди - 0. В такому випадку потрібно вибрати програму яка має зрозумілу документацію та велику кількість користувачів які можуть допомогти і поділитися досвідом - в основному онлайн туторіали або деколи поради.

Бюджет та часові рамки також не великі. Бюджет очевидно 0 гривень 0 копійок. Часові рамки будуть приблизно 4-5 місяців. Це ще більше підтверджує вибір простої програми з невеликим бар'єром входу.

Враховуючи цю інформацію розберемо 4 популярних ігрових двигуна [11] і їх характеристики і чому вони підходять або не підходять:

1. Unreal Engine.
2. Godot.
3. GameMaker: Studio.
4. Unity.

Unreal Engine базований на C++, ця програма від Epic Games є одним із перших варіантів, які розробники рекомендують розглянути під час планування процесу розробки гри. Його розширені функції розробки роблять багато процесів розробки більш доступними, а його природа з відкритим кодом відкриває чудову перспективу для розширення команд розробників.

Unreal Engine надає чудові інструменти розробки, такі як візуальні сценарії Blueprints, створення світу в реальному часі, покращені технології світла й тіні, такі як Lumen, Nanite Virtualized Geometry та багато іншого.

Останнє оновлення Unreal Engine 5.2 також додало багато покращень у структуру процедурної генерації вмісту, Substrate – новий спосіб створення матеріалів, набір інструментів для віртуального виробництва та інші функції.

Переваги:

– гнучкість: незалежно від проекту, Unreal Engine справляється з ним добре. Існують відомі ігри буквально кожного жанру, створені з ним. Те саме для всіх платформ;

– роялті, зручні для творців: для публікації ви можете безкоштовно використовувати UE, доки ваш проект не отримає дохід у 1 000 000 доларів США. Відтоді ви платите Epic Games лише 5% роялті. Існує також повністю безкоштовна ліцензія для творців, але ви не можете використовувати її для публікації готових пропозицій;

– чудово підходить для великих ігор: UE чудово підходить для масштабування для великих команд та ігор із великою кількістю вмісту. Ви легко керуєте великими сценами з великою кількістю акторів прямо з коробки.

Недоліки:

– технічний досвід і добре знання C++ є обов’язковими для належної розробки з Unreal Engine;

– працює ліпше з 3D проектами ніж з 2D;

– велика специфіка: Великі мінімальні характеристики пк;

– навантаження: Велика вага кінцевих файлів проектів.

Підсумуємо, Unreal Engine - прекрасний вибір для великих компаній та досвідчених розробників, враховуючи що C++ вважається однією з тяжчих мов програмування, і проектів які розраховані на високоякісну 3D графіку з хорошою системою фізики та орієнтовано на користувачів з преміум якістю персональних комп'ютерів.

Виходячи з цих фактів він не підійде для цього проекту, який буде 2D без ніякої спеціальної фізики та графіки. Також мінімальні знання C++ зупиняють мене від вибору цього рушія.

Godot це кросплатформний безкоштовний ігровий двигун із відкритим вихідним кодом, випущений згідно з дозвільною ліцензією MIT. Середовище розробки працює на багатьох платформах і може експортуватися на пару інших.

Godot дозволяє розробникам відеоігор створювати як 3D, так і 2D ігри, використовуючи кілька мов програмування, таких як C++, C# але основною є GDScript. Він використовує ієрархію вузлів для полегшення процесу розробки. Класи можуть бути похідними від типу вузла для створення більш спеціалізованих типів вузлів, які успадковують поведінку. Вузли організовані всередині «сцен», які є багаторазово використовуваними, інстанційними, успадкованими та вкладеними групами вузлів. Усі ігрові ресурси, включаючи сценарії та графічні ресурси, зберігаються як частина файлової системи комп’ютера (а не в базі даних). Це рішення для зберігання призначене для полегшення співпраці між командами розробників ігор, які використовують системи контролю версій програмного забезпечення.

Переваги:

– універсальність: розробники ігор, які використовують Godot, часто кажуть, що ви, швидше за все, будете обмежені відсутністю ваших навичок, ніж обмеженнями інструментів Godot. Двигун здатний створювати практично будь-яку гру розміром не AAA;

– чудово підходить для 2D-ігор: завдяки окремим двигунам для 2D- і 3D-проектів Godot охоплює різноманітні завдання. Багато інді-розробників вважають його найкращим двигуном відеоігор для 2D. Більшість відомих проектів Godot - це 2D-ігри. Godot 2D має більшу спільноту та пройшов більше шліфування. Він використовує власні вимірювання на основі пікселів, що робить його чудовим для проектів 2D піксельного мистецтва;

– безкоштовно та з відкритим кодом: ви можете використовувати 100% того, що Godot пропонує вам безкоштовно. Просто зробіть пожертву

Patreon, якщо хочете. Уся документація, навчальні посібники та вихідні коди доступні для всіх.

Недоліки:

– не для 3D-проектів: Godot має всі основні інструменти для створення 3D-ігор, але це не те, де він блищить. Інструменти 3D-дизайну та візуалізації не такі потужні, порівняно з конкурентами;

– немає підтримки консолі: структура Godot з відкритим вихідним кодом унеможливлює розробку консольних ігор. Наразі розробники Godot не планують отримувати ліцензії на основні консолі. Це не означає, що неможливо перенести ігри Godot на консолі. Вам просто знадобиться залучити сторонню компанію;

– відсутність функцій: Редактор спрайтів, блокування оклюзії, атрактори частинок і деякі інші функції, стандартні для інших ігрових двигунів, наразі недоступні в Godot.

Результатом є непогане програмне забезпечення яке має великий потенціал, проте на даний момент не дуже підходить для цього проекту. Я можу бачити хороше майбутнє для цього двигуна враховуючи що він набирає популярність, користувачів та все ще получає оновлення.

GameMaker: Studio дає змогу створювати міжплатформні та багатожанрові відеоігри за допомогою спеціальної мови візуального програмування з функцією перетягування та скидання або мови сценаріїв, відомої як Game Maker Language, яку можна використовувати для розробки більш просунутих ігор, які неможливо створити просто за допомогою функцій візуального програмування. GameMaker спочатку був розроблений, щоб дозволити програмістам-початківцям створювати комп’ютерні ігри без особливих знань програмування за допомогою цих дій. Останні версії програмного забезпечення також зосереджені на зверненні до просунутих або досвідчених розробників.

GameMaker в основному призначений для створення ігор із 2D-графікою, дозволяючи готове використання растрової графіки, векторної графіки (через

SWF) і 2D-скелетної анімації (через Esoteric Software's Spine) разом із велика стандартна бібліотека для малювання графіки та 2D примітивів. Хоча програмне забезпечення дозволяє обмежене використання 3D-графіки, це у формі буфера вершин і матричних функцій, і тому не призначене для початківців користувачів.

Переваги**:**

- простий інтерфейс;
- дружній для новачків.

Недоліки:

- погане для досвідчених розробників;
- дорога стартова ціна.

Найкращий вибір для даного проекту серед попередньо перелічених за одним винятком - ціна [12].

Unity - цей рушій на основі C# досі вважається найпоширенішим ігровим движком у світі. На сьогодні Unity зберігає найбільшу кількість ігор у Steam.

Розробників приваблює двигун Unity, оскільки порівняно з іншими прогресивними 3D-движками до нього потрібно менше зусиль, щоб звикнути. Unity має багато функцій, і його універсальність неймовірна.

Насамперед, за допомогою лише цього рушія можна створити майже все: 2D і 3D проекти різного масштабу. Основне обмеження полягає в його графічних можливостях. Розробнику доведеться трохи попітніти, щоб зробити картинку більш-менш реалістичною. Однак існують межі реалістичного рендерингу, які Unity ще не може перевершити. Але, можливо, ця особливість надає двигуну певної індивідуальності. У той час як усі інші роблять усе можливе, щоб отримати максимально реалістичне зображення, Unity робить його барвистим і приємним для перегляду та зосереджується на зручності використання та модульній системі.

Переваги:

- легко почати: Unity — один із найзручніших механізмів розробки ігор за всю історію;

– підходить для портування: Unity надає спеціалізований набір інструментів для перенесення ігор на будь-яку необхідну ігрову платформу;

– великий магазин ассетів: магазин ассетів Unity добре укомплектовано для інших інструментів, має велику та детальну документацію та підтримується великою професійною спільнотою.

– надійна спільнота: тисячі компаній використовують Unity і охоче діляться своїм досвідом, щоб ви могли звернутися до спільноти за допомогою з будь-якої проблеми.

Недоліки:

– рудиментарність: запустити збірку на Unity легко, навіть якщо вона невисокої якості. В Unity доступні деякі погані методи кодування, які в майбутньому викличуть багато роботи з оптимізації.

– не надто гнучкий: деякі функції Unity досить важко масштабувати або змінювати, що робить його не є самим найпростішим варіантом для ігор більшого масштабу.

– складні візуальні ефекти: порівняно з іншими двигунами, можна зазначити, що Unity не надає широкого спектру інструментів для роботи зі світлом, шейдерами, інтерфейсом користувача та деякими функціями, пов’язаними з мистецтвом.

Враховуючи ці фактори було вирішено вибрати саме Unity та мову програмування, яка в основному використовується в ньому, C# для виконання даного проекту.

2.2 Вибір художнього стилю

Не зважаючи на всі старі приказки “Не суди книгу по обкладинці” і тому подібне, це все рівно перше що ми робимо. Це також можна сказати про художні стилі в іграх.

Графіка не існує окремо від ігрових механік. Звичайно, хороша картинка не спасе погану не інтересний ігровий цикл, але хороший ігровий процес зачасту портиться без мінімального візуалу.

Художній стиль гри - гармонійна комбінація всіх візуальних елементів гри, ідеально підібраних для того, щоб передати підібрану розробниками атмосферу для гравців. В ідеалі все має бути саме так. Проте, часто можна побачити розгубленість та невпевненість у вигляді не сходження стилю з сутністю гри [13]. І це не означає, що розробки десь помилились, або щось не доробили. Можливо ця ідея була занадто амбіційною по зрівнянню з ресурсами команди розробників.

Ресурси художників потребують багато часу і знань. Не важливо скільки людей працює над грою, від одного до великої команди, важливо знати який стиль необхідно використати для проекту. Стилі діляться на 2D та 3D.

3D стилі діляться на:

1. Реалізм.
2. Фентезі Реалізм.
3. Мало-полігонний.
4. Намальований.
5. Мультиковий.

Проте не потрібно на них довго зосереджуватися тому що вибраний стиль для даної гри буде в 2D. Основні стилі 2D:

1. Плоский.

Художня форма включає в себе мінімалізм і чіткі лінії, залишаючись простою і привабливою. За допомогою плоского дизайну такі ігри, як Monument Valley (рис 2.1), створюють візуально чудове середовище та привабливих персонажів, які однаково модні та привабливі.

Переваги плоского стилю включають простоту, мінімалізм і чистий зовнішній вигляд. Він забезпечує легке читання та виразну візуальну мову, що робить його ідеальним для вираження певних настроїв або підкреслення компонентів гри. Глибоке розуміння композиції, теорії кольору та принципів

графічного дизайну є необхідним для створення плоского мистецтва, яке є візуально привабливим. Може бути важко підтримувати візуальний інтерес, не виявляючи себе надмірно простим.

Рисунок 2.1 – Приклад стилістики гри Monument Valley

2. Векторний.

Зображення, створені у векторному стилі, мають гострі краї, і їх можна змінювати без втрати якості, оскільки для їх створення використовуються математичні процедури. Це забезпечує універсальність у створенні активів і добре працює для 2D-проектів або мобільних ігор з різною роздільною здатністю моніторів.

3. Геометричний стиль.

Найпростіший з усіх, а також назва сама собою зрозуміла. Використовуючи геометричні фігури, художник створює середовище, де гравець контролює одну з форм і використовує її, щоб ухилятися або стріляти в інші ігрові фігури.

Thomas Was Alone (рис 2.2) — це гра, яка вшановує елегантність і простоту геометрії та забезпечує унікальний і візуально захоплюючий досвід.

Створюючи гру в стилі геометричного мистецтва, необхідно бути точним у геометричних композиціях і вміти знайти ідеальний баланс між простотою цього стилю та його складністю.

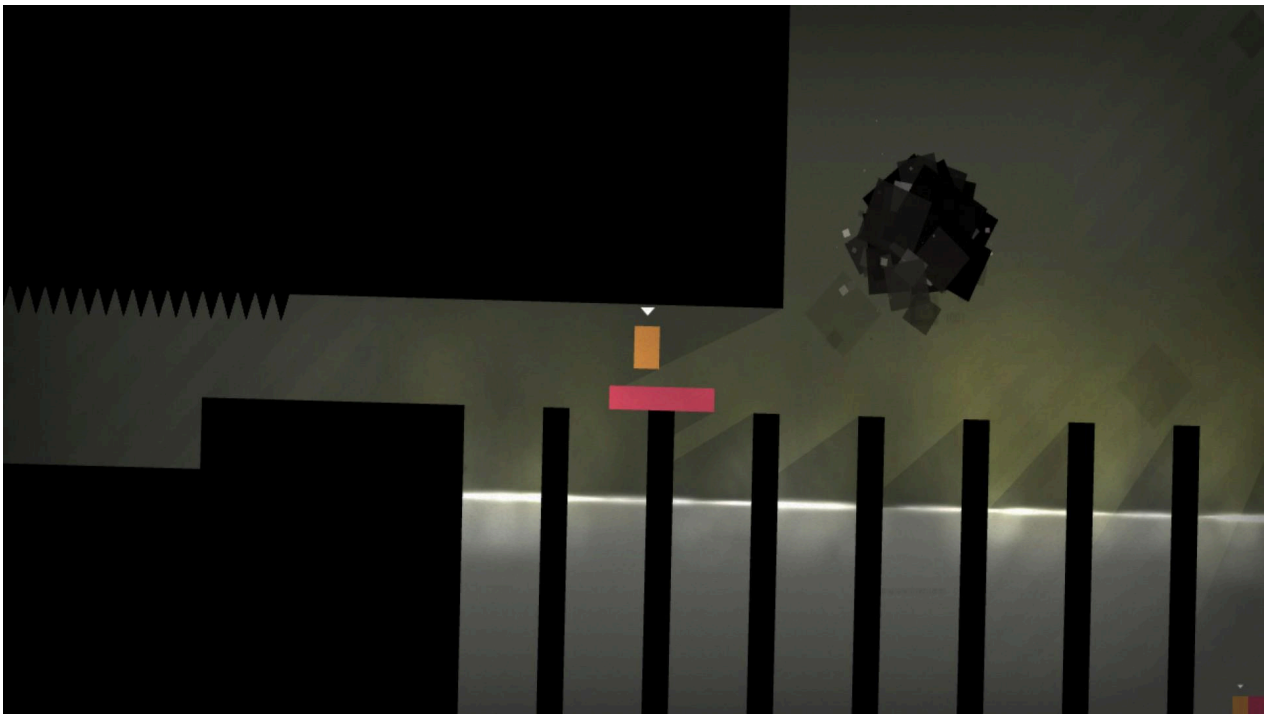


Рисунок 2.2 – Приклад гри Thomas Was Alone та геометричного візуального ігрового стилю

4. Піксель.

Піксельне мистецтво чудово підходить для створення ретро-стилю та відчуття. Завдяки асоціації зі старовинними іграми та ретро-дизайном піксельне мистецтво займає унікальне місце в серцях геймерів. Милі блокові фігури Hotline Miami та Stardew Valley (рис 2.3), а також яскраве, ностальгічне середовище ідеально підходять для любителів ігор 80-х.

5. Монохромний.

У світі мільярдів кольорів і магічного реалізму чорно-білі ігри також мають своїх шанувальників! Цей художній стиль складається з використання

одного кольору (чорного або білого) як базового кольору, а потім використання різних його відтінків для фарбування всього. Хоча я повинен зауважити, що іноді інші кольори використовуються в монохроматичних дизайнах, як-от різні відтінки зеленого, оранжевого тощо [14].

Рисунок 2.3 – Піксельний стиль на прикладі гри Stardew Valley

Limbo та Inside — два найвідоміші приклади, які створили захоплюючі та зловісні враження в захоплюючій та моторошній атмосфері. В підсумку найліпшим вибором буде піксельна або геометрична стилістика, які є найбільш дружні для новачків.

2.3 Аудіо супровід

Для правильної атмосфери, як в кіно так і в відеоіграх, необхідно вибирати правильний візуальний та аудіо супровід. Про візуальний елемент медіуму було сказано достатньо тепер час розібрати музику. Під аудіо супроводом я вважатиму все що гравець чує при проведенні часу в моїй грі під цим я маю на увазі декілька критичних елементів:

1. Музика. Музика дозволяє розробникам задати певну атмосферу яку вони вважають корректную під ту чи іншу локацію. Грайлива музика з швидким темпом підходить ідеально для моментів, де користувачу дана свобода в дослідженні кольорового світу, проте повільне фортепіано може означати серйозніший може навіть сумний момент в історії.

В даному випадку гра планується аркадного типу з простим стилем тому щось по типу 8-бітної музики буде ідеальним вибором.

2. Ігрове середовище по якому просувається гравець. Скрип дверей, шелест вітру, потік води і тому подібні речі необхідно взяти до уваги при створенні необхідного середовища. Маленькі деталі неймовірно важливі тому що їх нестиковки дуже виділяються.

3. Персонажі з якими гравець буде взаємодіяти. Під цим мається на увазі голос, одяг та люба екіпіровка яку вони мають. Звичайно якщо такі персонажі існують і гравець не взаємодіє тільки з ворогами які не є нічим особливими.

4. Сам персонаж гравця. Принцип подібний до інших персонажів гри - голос, одяг, екіпіровка і зброя яку він може використовувати.

5. Тільки цього разу даний персонаж буде на екрані гравців протягом більшості якщо не весь час проведений в грі тому необхідно провести більше часу на цим завданням.

Враховуючи мінімальні знання в створенні аудіо та музики було б доцільно використати онлайн ресурси з безплатними вже створеними ассетами. При використанні безплатних речей потрібно бути впевненим що те що ви використовуєте саме те що не є ліцензованим і те що автор визначив як предмет який можна використовувати в комерційних цілях.

Висновки до розділу 2

Результатом розділу є детальне описання методів які будуть використані при створенні гри. Також описані методи для того аби в подальшому полегшити

задачу. В першому підрозділі розібрали плюси і мінуси різних ігрових двигунів і чому саме вибір Unity підійде для цього проекту. В другому підрозділі розглянули візуальні стилі відеоігор і який можливо використати для цього проекту. В третьому підрозділі описано аудіо контент який використовується в типічних відеоіграх.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ОБ'ЄКТУ

3.1 Unity Asset Store

Як розробнику ігор важливо знайти способи оптимізувати свій робочий процес і заощадити час, де це можливо. Один із способів зробити це — використовувати ресурси з Unity Asset Store [15]. Unity Asset Store — це ринок готових ігрових ресурсів, таких як код, 3D-моделі та аудіо, які ви можете використовувати у своїх проектах. Загалом використання ресурсів із Unity Asset Store і як вони можуть допомогти вам швидше створювати кращі ігри.

Для чого можливо використати Unity Asset Store:

1. Пришвидшення процесу створення відеоігор.

Використання ассетів із Unity Asset Store може заощадити досить багато часу та зусиль, надаючи попередньо написаний код і фонові ассети, які можна включити у свою гру.

Це може допомогти розробникам зосередитися на творчих і унікальних аспектах їх гри, а не витрачати час на більш технічні та трудомісткі завдання.

2. Випробування нових ігрових механік.

Unity Asset Store — чудове місце, де можна спробувати різні ігрові ідеї та механізми. Ви можете експериментувати з різними ресурсами та бачити, як вони працюють у вашій грі, не витрачаючи час на їх створення з нуля.

3. Додатки до вже готових ігор.

Unity Asset Store має широкий спектр ассетів, які можуть допомогти розробникам додати блиск їх грі, як-от ефекти частинок, рішення для запікання та додатки для професійного освітлення. Ці ресурси можуть допомогти зробити вашу гру більш професійною та привабливою для гравців. Так само, використовуючи безліч систем контролю версій, ви можете забезпечити швидший робочий процес і менше помилок.

4. Зворотно проаналізувати як вони працюють.

Використовуючи ресурси з Unity Asset Store, можливо навчитися кодувати та створювати ігрові ассети, досліджуючи код і коментарі, включені до них. Це може бути чудовим способом покращити свої навички та знання як розробника.

5. Використовування ассетів для загальних дій, систем і графіки

У Unity Asset Store також є різноманітні основні ресурси, такі як контролери персонажів, пошук шляху ворога та повторювані ассети, як-от дерева, які ви можете використовувати у своїй грі. Ці ресурси можуть допомогти вам створити надійну основу для вашої гри та заощадити час на створення цих основних елементів з нуля.

6. Розширення охоплення гри.

Використовуючи інструменти локалізації, доступні в магазині Unity Asset, можливо зробити свою гру доступнішою для тих, хто не розмовляє тією ж мовою, що й розробник..

7. Створення ще більше ігор.

Розробники ігор завжди придумують нові ідеї, і чим швидше вони можуть створити ігри, тим швидше зможуть перейти до свого наступного проекту. Використовуючи ресурси з Unity Asset Store, розробники можете заощадити час і зосередитися на створенні нових ігор і втіленні ваших ідей у життя.

Саме тому в цій роботі також буде використано Unity Asset Store для того аби получити персонажа яким буде користуватися гравець та інші предмети які будуть задіяні в грі.

3.2 Скрипти для гри

Перед тим як переходити до розбору коду скриптів, важливо зазначити базові поняття, які будуть використані протягом цього підрозділу для того щоб не пояснювати їх в кожному з скриптів Unity [16].

Visual Studio [17] - програма яка буде використана як редактор коду тому що в ньому є спеціальний Unity пакет.

Метод Awake, викликається під час завантаження скрипту. Awake використовується для ініціалізації будь-яких змінних, або стану гри, перед початком гри. Awake викликається лише один раз протягом “життя” скрипту.

Метод Update відрізняється від методу Awake тим, що він викликається при кожному кадрі гри, дозволяючи приймати та надсилати інформацію для користувача максимально швидко.

SerializeField використовується при створенні змінної для того щоб мати можливість її редагувати в візуальному середовищі Unity.

OnTriggerEnter2D метод який часто використовується якщо потрібно виконати певну дію при стику одного об'єкта з іншим який має тег BoxCollider.

Скрипт камери:

```
    private void Update()
    {
        Vector3 targetPosition = target.position + offset;
transform.position = Vector3.SmoothDamp(transform.position,
targetPosition, ref velocity, smoothTime);
    }
}
```

Цей скрипт прикріплюється до об'єкту камери в Unity буде брати певну ціль target і плавно змінювати позицію камери для того аби вона слідувала за ціллю, в даній ситуації цей скрипт буде використаний для того аби слідувати за персонажем гравця.

Скрипт менеджеру звуку:

```
    public void PlaySound(AudioClip _sound)
    {
        audiosrc.PlayOneShot(_sound);
    }
public void ChangeSoundVolume(float _change)
{
float currentVolume = PlayerPrefs.GetFloat("SoundVolume", 1);
```

PlayerPrefs використовується для того щоб задати параметр користувачем і зберегти його для майбутнього використання, в випадку якщо такий параметр не знайдений натомість взято значення за замовчуванням, в цьому випадку - 1.

```
currentVolume += _change;
if (currentVolume > 1)
      currentVolume = 0;
else if (currentVolume < 0)
      currentVolume = 1;
```

Щоб користувач міг збільшувати та зменшувати звук без слайдерів йому необхідно просто продовжувати нажимати на кнопку.

```
audiosrc.volume = currentVolume;
PlayerPrefs.SetFloat("SoundVolume", currentVolume);
}
```

Саме SetFloat зберігає параметр який йому надається в дужках і переносить його для зберігання.

```
public void ChangeMusicVolume(float _change)
{ float currentVolume = PlayerPrefs.GetFloat("MusicVolume", 1);
currentVolume += _change;
if (currentVolume > 1)
      currentVolume = 0;
else if (currentVolume < 0)
      currentVolume = 1;
musicsrc.volume = currentVolume;
PlayerPrefs.SetFloat("MusicVolume", currentVolume);
}
```

В даному скрипті, який необхідно прикріпити до об'єкту “Audio Manager”, береться референс посилання на компонент AudioSource і використовуючи його створюється метод PlaySound в який буде надаватися аудіофайл який потрібно

буде надати в AudioClip _sound з серіалізованого поля яке буде створюватися після цієї дії.

Скрипт руху гравця PlayerMovement [18]:

В методі Awake створюються референси для компонентів:

```
private void Awake()
    {
        body = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        boxCollider = GetComponent<BoxCollider2D>();
    }
```

І перед тим як переходити до методу Update для того аби всі потрібні функції працювали необхідно створити декілька перевірок.

```
private bool isGrounded()
    {
RaycastHit2D raycastHit = Physics2D.BoxCast(boxCollider.bounds.center, boxCollider.bounds.size,0, Vector2.down,0.1f, groundLayer);
        return raycastHit.collider != null;
    }
```

Метод isGrounded() створює коробку під персонажем та передає інформацію чи вона стикається з предметом зі слоєм groundLayer це буде використано для того аби перевірити чи гравець на землі чи ні.

```
private bool onWall()
    {
RaycastHit2D raycastHit = Physics2D.BoxCast(boxCollider.bounds.center, boxCollider.bounds.size, 0, new Vector2(transform.localScale.x,0), 0.1f, wallLayer);
      return raycastHit.collider != null;
    }
```

Метод onWall() працює рівно в тому ж самому принципі тільки замінює слой groundLayer на wallLayer.

```
private void wallJump()
    {
body.AddForce(new Vector2(-Mathf.Sign(transform.localScale.x) *
WalljumpX, WalljumpY));
    }
```

Напряму після попередньої перевірки створюється wallJump() який буде використовуватися якщо перевірку пройдено і буде відштовхувати гравця від стіни для того щоб стрибати по ній або від неї кудись.

```
  private void Jump()
    {
if (coyoteCounter <= 0 && !onWall() && jumpCounter <= 0) return;
        SoundManager.instance.PlaySound(JumpSFX);
```

Метод Jump() створений для того аби дозволяти персонажу стрибати в грі тобто змінювати його позицію по осі Y. В додаток до простої перевірки додається CoyoteTimer який буде дозволяти гравцю стрибати з певним буфером для приємнішого керування персонажем. Ну і звичайно при кожному стрибку буде відтворюватися звук “JumpSFX”.

```
if (onWall())
        wallJump();
```

Якщо користувач пройшов перевірку onWall() замість простого стрибку буде використано код записаний в методі wallJump().

```
else
    {
        if (isGrounded())
```

```
body.velocity = new Vector2(body.velocity.x, jumpPower);
```

Якщо гравець проходить перевірку isGrounded() виконується код який відповідає за стрибки і піднімає користувача по осі Y на задане в візуальному вікні Unity число jumpPower.

```
else
    {
        if (coyoteCounter > 0)
body.velocity = new Vector2(body.velocity.x, jumpPower);
```

Саме тут виконується перевірка CoyoteTimer і задіюється буфер, якщо таймер coyoteCounter (задана нами в візуальному середовищі Unity) більше нуля гравець все ще зможе виконати стрибок.

```
else
        {
            if(jumpCounter > 0)
            {
body.velocity = new Vector2(body.velocity.x, jumpPower);
                jumpCounter--;
            }
```

І звичайно якщо задано що користувач має більше ніж один стрибок і таймер coyoteCounter вийшов то буде використано саме додаткові стрибки а не основний.

```
coyoteCounter = 0;
```

І звичайно після виконаного стрибку coyoteCounter повинен бути призначений 0 так щоб не виникли помилки і користувач не зміг використовувати головний стрибок більше ніж один раз.

І після цього можна перейти саме в метод Update де це все буде імплементуватися.

```
private void Update()
{
    horizontalInput = Input.GetAxis("Horizontal");
```

Це референс для кожного разу як користувач натискає кнопки A, D, та стрілки вправо та вліво.

```
  if(horizontalInput > 0.01f)
       transform.localScale = Vector3.one;
   else if (horizontalInput < -0.01f)
       transform.localScale = new Vector3(-1, 1, 1);
```

Ця стрічка дозволяє Unity поверати модель користувача вправо та вліво в залежності від клацнутих гравцем клавіш.

```
   anim.SetBool("run", horizontalInput != 0);
    anim.SetBool("grounded", isGrounded());
```

Відправляє інформацію в аніматор на основі заданих параметрів, яка буде використана для того аби відображати задані анімації.

```
  if (Input.GetKeyDown(KeyCode.Space))
       Jump();
    if (Input.GetKeyUp(KeyCode.Space) && body.velocity.y > 0)
body.velocity = new Vector2(body.velocity.x, body.velocity.y / 2);
```

Якщо користувач нажимає кнопку пробіл буде використано попередньо описаний скрипт Jump(), проте якщо він її не зажме і відпустить чуть пізніше то модель гравця не стрибне нижче.

```
        if (onWall())
          {
              body.gravityScale = 0;
              body.velocity = Vector2.zero;
          }
          else
          {
            body.gravityScale = 7;
body.velocity = new Vector2(horizontalInput * speed, body.velocity.y);
```

Цей код дозволить персонажу при перевірці методом onWall() прилипнути до стіни і не рухатися, а якщо він на землі то просто дозволить йому пересуватися використовуючи horizontalInput.

```
            if(isGrounded())
            {
                coyoteCounter = coyoteTime;
                jumpCounter = ExtraJumps;
            }
            else
                coyoteCounter -= Time.deltaTime;
        }
```

Цей код тримає coyoteCounter та jumpCounter таким самим, яким його було задано, якщо користувач все ще на землі, і зменшує coyoteCounter якщо він не на землі.

```
public bool canAttack()
{
return !onWall();
}
```

Цей метод створений для наступного скрипту який буде дозволяти гравцю пускати вогняну кулю, метод створений публічним для його подальшого

використання в іншому скрипті, і він може зробити це в любому стані окрім якщо він прилип до стіни.

Скрипт PlayerAttack:

```
private void Attack()
{
    SoundManager.instance.PlaySound(fireballSound);
    anim.SetTrigger("attack");
    cooldownTimer = 0;
    fireballs[FindFireball()].transform.position = firePoint.position;
fireballs[FindFireball()].GetComponent<Projectile>().SetDirection(Mathf.Sign(transform.localScale.x));
}
```

Метод Attack() буде використаний, коли гравець хоче запустити кулю і грає заданий звук fireballSound і передати аніматору тригер attack, для того щоб відтворити анімацію атаки і ставить її на перезарядку. Створення кулі працює за допомогою її появи на точці firePoint.position, яка задана перед цим, і її налаштування до того в скрипті Projectile. Проте, якщо створено тільки одну кулю, то при спрацюванні методу Attack(), попередня просто зникне і з'явиться нова, саме тому в цьому випадку використано масив в якому є 10 куль, які будуть перераховуватися в масиві fireballs з допомогою методу FindFireball().

```
private int FindFireball()
{
for(int i = 0; i < fireballs.Length; i++)
{
    if (!fireballs[i].activeInHierarchy)
        return i;
}
return 0;
}
```

Цей метод перераховує всі об'єкти в масиві fireballs і вертає кулю, яка на даний момент не є activeInHierarchy, тобто не активна на сцені.

```
private void Update()
{
if (Input.GetMouseButton(0) && cooldownTimer > attackCooldown &&
playerMovement.canAttack())
      Attack();
cooldownTimer += Time.deltaTime;
}
```

Простий метод який дозволяє користувачу атакувати, якщо натиснути ліву кнопку миші, проведену перевірку canAttack та таймер перезарядки більший за заданий параметр attackCooldown.

Скрипт Projectile:

```
lifetime = 0;
direction = _direction;
gameObject.SetActive(true);
hit = false;
boxCollider.enabled = true;
float localScaleX = transform.localScale.x;
if (Mathf.Sign(localScaleX) != _direction)
localScaleX = -localScaleX;
transform.localScale = new Vector3(localScaleX,
transform.localScale.y, transform.localScale.z);
```

Метод SetDirection виглядає так, він використовується для того щоб коректно визначити шлях кулі попереду користувача.

```
private void OnTriggerEnter2D(Collider2D collision)
{
hit = true;
boxCollider.enabled = false;
anim.SetTrigger("explode");
```

```
}
```

Використовується для задання стану кулі hit на true при попаданні в любий об'єкт який має компонент boxCollider, після чого відтворюється попередньо задана анімація.

```
private void Update()
{
if (hit) return;
float movementSpeed = speed * Time.deltaTime * direction;
transform.Translate(movementSpeed, 0, 0);
lifetime += Time.deltaTime;
if (lifetime > 5) gameObject.SetActive(false);
}
```

В методі Update() куля рухається до поки hit не стане true або час життя lifetime не буде більше 5 секунд.

Скрипт Health:

```
public void TakeDMG(float _damage)
{
HealthRN = Mathf.Clamp(HealthRN - _damage, 0, StartedHealth);
```

Mathf.Clamp стискає число і не дозволяє йому виходити за задані межі в дужках тобто між 0 і записаною змінною StartedHealth.

```
if (HealthRN > 0)
{
      SoundManager.instance.PlaySound(HurtSound);
      anim.SetTrigger("hurt");
      StartCoroutine(Invulnerability());
}
```

Кожного разу, коли метод викликається, він викликається з певним числом яке віднімається від змінної HealthRN, яка за замовчуванням дорівнює 3, і при

перевірці якщо воно все ще більше нулю то буде виконана відповідна анімація і звук. Також виконується метод Invulnerability який відключає можливість гравцем получити більше шкоди протягом заданого часу.

```
else
{
      if (!dead)
            anim.SetTrigger("die");
      GetComponent<PlayerMovement>().enabled = false;
      dead = true;
      anim.SetBool("grounded", true);
      SoundManager.instance.PlaySound(DeathSound);
}
```

В іншому випадку гравець помирає і в нього відключається скрипт PlayerMovement задається правильна анімація та звук і стан dead.

```
public void GetHp(float _toadd)
{
HealthRN = Mathf.Clamp(HealthRN + _toadd, 0, StartedHealth);
}
```

Простий метод, який додає здоров'я до HealthRN, використовуючи той самий метод Mathf.Clamp.

```
public void Respawn()
{
dead = false;
GetHp(StartedHealth);
anim.ResetTrigger("die");
anim.Play("Idleplayer");
StartCoroutine(Invulnerability());
GetComponent<PlayerMovement>().enabled = true;
```

Цей код буде працювати в тандемі з скриптом PlayerRespawn, і буде відміняти все що було виконано при смерті гравця.

```
private IEnumerator Invulnerability()
{
Physics2D.IgnoreLayerCollision(8,9,true);
```

Дозволяє предмету з слоєм 8, в цьому випадку це слой Player або гравець, ігнорувати слой 9, в цьому випадку слой Enemies або вороги. Створено для того щоб гравець мав пару секунд після получення шкоди в безпеці від ворогів.

```
for (int i = 0; i < Flashes; i++)
{
   SpriteRen.color = new Color(1, 0, 0, 0.5f);
  yield return new WaitForSeconds(iFrameDuration / (Flashes * 2));
     SpriteRen.color = Color.white;
  yield return new WaitForSeconds(iFrameDuration / (Flashes * 2));
}
```

Код для того щоб це показати гравцю візуально. Персонаж буде мигати червоним пару раз до поки метод працює.

```
Physics2D.IgnoreLayerCollision(8, 9, false);
```

Після пари секунд відключається перша лінія коду.

Скрипт Healthbar:

```
private void Awake()
{
          healthtotal.fillAmount = playerHealth.HealthRN / 10;
}
```

Беруться дві картинки healthtotal та healthcurrent, одна з пустими іконками здоров'я, та інша замальована червоним. І методи в цьому скрипті будуть міняти

видимість замальованої картинки для того щоб показати гравцю скільки залишилось очків здоров'я (рис 3.1).

```
private void Update()
{
          healthcurrent.fillAmount = playerHealth.HealthRN / 10;
}
```

Ця лінія перевіряє кожного кадру чи змінилась змінна HealthRN і змінює її видимість на відповідну.

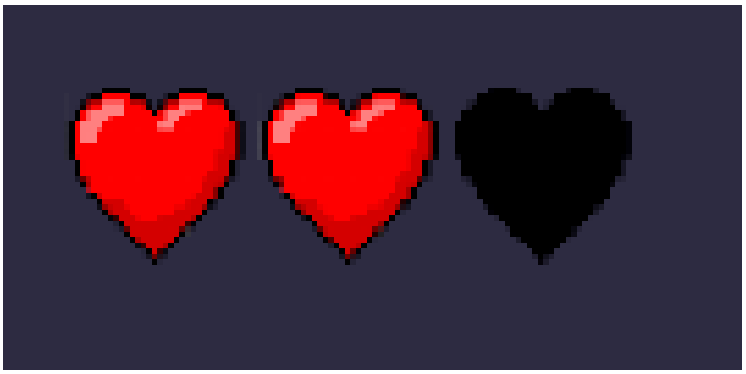


Рисунок 3.1 – Візуальний дисплей очок здоров'я гравця

Скрипт Collectible:

```
private void OnTriggerEnter2D(Collider2D collision)
{
     if(collision.tag == "Player")
     {
          SoundManager.instance.PlaySound(pickup);
          collision.GetComponent<Health>().GetHp(value);
          gameObject.SetActive(false);
     }
```

Цей код доволі простий і він буде спрацьовувати коли користувач буде підходити до об'єкту Collectible, і викликати метод GetHp з скрипту Health і добавляти йому задану кількість очок здоров'я.

Скрипт PlayerRespawn:

```
public void CheckForRespawn()
{
            if( CurrentCheckpoint == null )
      {
            uiManager.GameOver();
            return;
}
```

Ця частина коду перевіряє чи гравець має активовані контрольні точки і якщо ні то виконує метод GameOver і показує користувачу екран кінця гри. Також записано return, щоб пропустити наступний код і не виконувати його.

```
transform.position = CurrentCheckpoint.position;
      playerHealth.Respawn(); }
```

В іншому випадку використовується код Respawn, який знаходиться в скрипті Health та переносить гравця на те саме місце, на якому знаходиться остання активована контрольна точка.

```
private void OnTriggerEnter2D(Collider2D collision)
{
      if     (collision.transform.tag     ==     "Checkpoint"     &&
playerHealth.HealthRN != 0)
```

Цей код працює тільки у випадку якщо тег речі яку задів гравець “Checkpoint” і якщо очки здоров'я персонажа не дорівнюють 0. Причина перевірки очок здоров'я я знайшов при тестуванні гри, в випадку якщо цієї частини не має, гравець може активувати контрольну точку навіть після того як він загинув.

```
            CurrentCheckpoint = collision.transform;
            SoundManager.instance.PlaySound(checkpointSFX);
```

```
            collision.GetComponent<Collider2D>().enabled = false;
            collision.GetComponent<Animator>().SetTrigger("Appear");
        }
```

Получає інформацію де знаходиться активована контрольна точка, грає відповідний звук, відключає Collider2D, щоб гравець не мав можливості активувати її більше ніж раз та відтворює анімацію.

Скрипт SelectionArrow який буде прив'язаний до стрілки вибору яка буде використовуватись в головному меню, в меню паузи, при появі екрану поразки.

```
private void ChangePositions(int _change)
currentPos += _change;
if (_change != 0)
SoundManager.instance.PlaySound(changeSFX);
if (currentPos < 0)
currentPos = options.Length - 1 ;
else if (currentPos > options.Length - 1)
currentPos = 0 ;
rect.position = new Vector3(rect.position.x,
options[currentPos].position.y, 0);
```

Скрипт бере массив options, в яких знаходяться опції по яким буде ходити стрілки в любому меню і міняє їх позицію в залежності від числа _change.

```
private void Interact()
{
SoundManager.instance.PlaySound(interactSFX);
options[currentPos].GetComponent<Button>().onClick.Invoke();
}
```

Виконує заданий метод в настройках Button, який буде прив'язаний в наступному скрипті і відтворює звук interactSFX.

```
private void Update()
{
```

```
if (Input.GetKeyDown(KeyCode.W) || Input.GetKeyDown(KeyCode.UpArrow))
          ChangePositions(-1);
if (Input.GetKeyDown(KeyCode.S) || Input.GetKeyDown(KeyCode.DownArrow))
          ChangePositions(1);
if (Input.GetKeyDown(KeyCode.E) ||
Input.GetKeyDown(KeyCode.KeypadEnter))
          Interact();
     }
```

Використовує попередньо створені методи і дозволяє користувачу рухати стрілкою та нажимати на них за допомогою заданих кнопок.

Скрипт UiManager який буде відповідати за всі кнопки в меню:

```
     private void Awake()
     {
     GameOverScreen.SetActive(false);
     PauseScreen.SetActive(false);
     }
```

Про всяк випадок для того, щоб уникнути багів при запуску гри, необхідно виключити екрани кінця гри та паузи.

```
     private void Update()
     {
     if (Input.GetKeyUp(KeyCode.Escape)){
          if (PauseScreen.activeInHierarchy)
               PauseMenu(false);
          else
               PauseMenu(true);
     }
     }
```

Гравець може нажати кнопку Escape для того щоб запустити меню паузи, проте, якщо вона вже запущена то її необхідно виключити.

```
     public void GameOver()
```

```
{
GameOverScreen.SetActive(true);
SoundManager.instance.PlaySound(GameOverSFX);
}
```

Метод створено публічним для того щоб мати можливість його викликати в інших частинах скрипту.

```
public void Restart()
{
SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}
```

Відповідає за кнопку “Заново” в меню кінця гри (рис 3.2) і відправляє користувача на початок активного рівня.

Рисунок 3.2 – Меню кінця гри

```
public void MainMenu()
{
SceneManager.LoadScene(0);
}
```

Метод, який відправляє гравця в меню, меню було поставлено першим елементом в списку рівнів тому дано номер 0.

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

Цей код дозволяє гравцю вийти з гри.

```
public void PauseMenu(bool status)
{
PauseScreen.SetActive(status);
if (status)
      Time.timeScale = 0;
else
      Time.timeScale = 1;
}
```

В додаток до того, щоб гравець бачив меню паузи, необхідно поставити Time.timeScale до 0, щоб гра припинила рух загалом і персонаж зовсім не міг рухатися і залишався на одному місці.

```
public void SoundVolume()
{
SoundManager.instance.ChangeSoundVolume(0.2f);
}
```

Використовує попередній код ChangeSoundVolume, і додає йому 0.2f до гучності, так само працює метод MusicVolume тільки він минає тільки музика на відміну від усіх звуків гри.

Скрипт VolumeText буде використаний для того щоб показати користувачу інформацію про те наскільки гучна музика та звуки в грі.

```
private void UpdateVolume()
{
float volumeValue = PlayerPrefs.GetFloat(volumeName) * 100;
txt.text = textIntro + volumeValue.ToString();
}
```

Цей код бере задане користувачем в минулому скрипті і перетворює його в те, що користувач зможе побачити на екрані і зрозуміти.

```
private void Update()
{
UpdateVolume();
}
```

Попередній код вставляється в метод Update, для його виконання при кожній маніпуляції користувачем.

Скрипт TrapDamage буде батьківським класом всіх наступних пасток і матиме простий код:

```
protected void OnTriggerEnter2D(Collider2D collision)
{
if(collision.tag == "Player")
      collision.GetComponent<Health>().TakeDMG(damage);
}
```

Код полягає в тому, що він буде спрацьовувати коли гравець наступає на них, буде забрано певна кількість очок здоров'я damage.

Скрипт SawTrap - це спеціальна пастка, яка рухається від одного місця до іншого по одній осі:

```
private void Awake()
 {
      leftEdge = transform.position.x  - movementdistance;
      rightEdge = transform.position.y + movementdistance;
 }
```

Визначає між якими точками буде подорожувати пастка.

```
private void Update()
 {
      if (MovingLeft)
      {
      if( transform.position.x > leftEdge)
      {
            transform.position = new Vector3(transform.position.x -
speed * Time.deltaTime, transform.position.y, transform.position.z);
      }
```

Рухає пастку в задану позицію якщо її координати більші за позицію лівого краю.

```
      else
            MovingLeft = false;
      }
      else
      {
      if (transform.position.x < rightEdge)
```

В іншому випадку рухає її в проположну сторону тим самим чином заставляє її рухатися між двома точками rightEdge та leftEdge.

```
      {
transform.position = new Vector3(transform.position.x + speed *
Time.deltaTime, transform.position.y, transform.position.z);
      }
      else
            MovingLeft = true;
      }
 }
```

Скрипт FireTrap це пастка яка працює наступним чином - гравець наступає на її модель, пастка спрацьовує і розпочинає таймер, коли таймер спливає модель пускає вогонь, і якщо модель гравця все ще знаходиться в радіусі вогню.

```
private IEnumerator ActivateFiretrap()
{
      triggered = true;
      spriteRend.color = Color.red;
      yield return new WaitForSeconds(activationDelay);
      SoundManager.instance.PlaySound(FireSFX);
```

Працює як попередження для гравця міняє колір моделі на червоний відтворює звук та чекає певний час activationDelay для того, щоб користувач міг відреагувати та ухилитися.

```
spriteRend.color = Color.white;
      active = true;
      anim.SetBool("activated", true);
      yield return new WaitForSeconds(activeTime);
```

Активовує пастку на певний час activeTime, та вказує на це гравцю з допомогою заданих анімацій activated.

```
      active = false;
      triggered = false;
      anim.SetBool("activated", false);
}
```

Змінює всі статусі на замовчувані після того як все було виконано.

```
private void OnTriggerEnter2D(Collider2D collision)
{
      if (collision.tag == "Player")
```

```
        {
            if (!triggered)
                StartCoroutine(ActivateFiretrap());
            player = collision.GetComponent<Health>();
        }
    }
```

При вході в визначену зону BoxCollider, якщо тег предмету Player використовується метод ActivateFiretrap.

```
    private void OnTriggerExit2D(Collider2D collision)
    {
        player = null;
    }
```

Цей код був написаний після знаходження багу при якому коли гравець знаходиться на краю BoxCollider’а спрайту то він буде получати шкоду без зупинки і зразу загине не залежно від його очок здоров'я.

```
    private void Update()
    {
        if (active && player != null)
            {
            player.TakeDMG(damage);
            player = null;
            }
    }
```

Додає шкоди до гравця на задаче число damage.

Скрипт SpikeHead найбільш унікальна пастка, вона активовується коли гравець стає в один з її шляхів, і продовжує його переслідувати до поки гравець все ще є на її шляху.

```
    private void Stop()
    {
```

```
destination = transform.position;
attacking = false;
}
private void OnEnable()
{
Stop();
}
```

На початку гри необхідно щоб пастка стояла нерухомо, і гравець зрозумів де вона знаходиться і що вона взагалі існує.

```
private void Update()
{
if (attacking)
      transform.Translate(destination * Time.deltaTime * speed);
```

Рухати пастку на до користувача тільки якщо вона атакує.

```
else
{
      checkTimer += Time.deltaTime;
      if (checkTimer > checkDelay)
            CheckForPlayer();
}
}
```

В іншому випадку продовжувати шукати ціль.

```
private void CheckForPlayer()
{
CalculateDirections();
for (int i = 0; i < directions.Length; i++)
{
Debug.DrawRay(transform.position, directions[i], Color.red);
RaycastHit2D hit = Physics2D.Raycast(transform.position,
directions[i], range, playerLayer);
```

RaycastHit2D створює 4 лінії у сторонах directions від предмету та перевіряє для подальшої перевірки.

```
if (hit.collider != null && !attacking)
{
      attacking = true;
      destination = directions[i];
      checkTimer = 0;
}
```

Якщо знаходьться ціль то пастка направляється в її сторону, і ставить таймер checkTimer для того щоб перестати атакувати на певний час.

```
private void CalculateDirections()
{
directions[0] = transform.right * range;
directions[1] = -transform.right * range;
directions[2] = transform.up * range;
directions[3] = -transform.up * range;
}
```

Цей код використовується для визначення шляху по якому буде подорожувати пастка.

```
private void Stop()
{
destination = transform.position;
attacking = false;
```

Використовується для зупинки пастки

```
private void OnTriggerEnter2D(Collider2D collision)
{
SoundManager.instance.PlaySound(impact);
```

```
base.OnTriggerEnter2D(collision);
Stop();
```

Метод бере наслідовані від TrapDamage параметри та добавляє новий звук, потім зупиняє її для подальшого використання. Скрипт ArrowTrap та EnemyProjectile працює по такому ж самому принципу як і вогняні кулі, створюються масив з 10 та ставиться точка з якої вони будуть надсилатись.

3.2 Аніматор Unity

Аватар визначає структуру скелета об'єкта, але контролер аніматора (Animator Controller) також потрібен для застосування анімації до скелета. Контроллер аніматора створює Unity і дозволяє розробникам керувати набором анімацій для персонажа і переключатися між ними при виконанні іншої умови.

Наприклад, розробники можуть переключитися від анімації походки до прижки при нажаті клавіші пробела. Контролер керує переходами між анімаціями за допомогою так званої машини стану (State Machine), - родом з іншої програми, написаної мовою візуального програмування в Unity.

Animator Controller створюється в меню Assets або через меню Create у Project View. В меню контролера створюються анімації з Unity Asset Store для оточення гравця [19] та його персонажа [20].

У більшості ситуацій нормально мати кілька анімацій і перемикатися між ними, коли виникають певні умови гри. Наприклад, ви можете перемикатися з анімації ходьби на стрибок щоразу, коли натискаєте пробіл. Однак, навіть якщо у вас є лише один анімаційний кліп, вам все одно потрібно розмістити його в контролері аніматора, щоб використовувати його на ігровому об’єкті.

Висновки до розділу 3

В даному розділі були описані конкретні кроки, в вибраній програмі для кваліфікаційної роботи Unity, для створення відео гри в жанрі 2Д платформер.

Обрані популярні методи які використовуються в більшості платформерах і графічні методи які дозволяють грі бути доступній на більшості ПК.

Спочатку були описані позитивні сторони використання Unity Asset Store. Після був створений базовий скрипт для рухання користувача, на основі якого були створені всі рухи гравця та його атаки. Пастки грають велику роль в цій грі тому їх створено було 4, для цього використано основу об'єктно-орієнтованого програмування а саме наслідування.

Присвоєно всім необхідним предметам анімації з допомогою аніматор контролера і раніше згаданих ассетів з Unity Asset Store. Обрані предмети мають всі потрібні кадри для створ

ВИСНОВКИ

В рамках кваліфікаційної роботи було вивчено предметну область проекту, а саме створення відео гри в жанрі 2Д платформер, з допомогою мови програмування C# в програмі Unity. Дослідження існуючих аналогів допомогло вказати на недоліки та переваги різних ігрових двигунів, та остаточно вирішити чому Unity найбільше підходить для цього проекту.

При виборі ігрових рушіїв для розробки відеогри було взято до уваги їхню доступність, надійність та популярність, що дозволило створити стабільну працюючу програму. Рушій Unity зробив розробку швидкою та ефективною, в той час як використання Unity Assets задовільнило потреби в графічних елементах і зменшило загальний час створення гри.

Була створена система руху, яка інтересна гравцю та є доволі швидкою, стрільба яку можливо використати в дуже специфічних моментах та пастки для того щоб гравцю дати гравцеві свого роду виклик. Також створено меню паузи, для того щоб користувач міг змінювати гучність звуків та музики в грі.

Результатом є створена повноцінна гра яку можна запустити на інших персональних комп'ютерах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Статистика використання ігрової індустрії та доходів за 2023 рік : веб-сайт. URL: https://helplama.com/game-industry-usage-revenue-statistics/ (дата звернення 12.02.2024)

2. Головний посібник з входу в індустрію розробки ігор : веб-сайт. URL: https://tripleten.com/blog/posts/your-ultimate-guide-to-breaking-into-the-game-development-industry (дата звернення 15.02.2024)

3. Відеоігри Це Хобі? (Що Говорить Наука?): веб-сайт. URL: https://www.gamedesigning.org/gaming/hobby/ (дата звернення 16.02.2024)

4. Статистика Steam за 2024 рік (користувачі, популярні ігри та ринок) : веб-сайт. URL: https://www.demandsage.com/steam-statistics/ (дата звернення 18.02.2024)

5. Ігри Metroidvania: 5 характеристик ігор Metroidvania : веб-сайт. URL: https://www.masterclass.com/articles/metroidvania-definition (дата звернення 20.02.2024)

6. Створення Hollow knight : веб-сайт. URL: https://www.gameinformer.com/2018/10/16/the-making-of-hollow-knight (дата звернення 22.02.2024)

7. Огляд Ori and the Blind Forest: «складна та красива подорож» : веб-сайт. URL: https://www.theguardian.com/technology/2015/apr/01/ori-and-the-blind-forest-review-a-challenging-and-beautiful-journey (дата звернення 24.02.2024)

8. Dead Cells — це груба, захоплююча та швидка гра, у яку вам потрібно зіграти цього тижня : веб-сайт. URL: https://www.gq.com/story/dead-cells-review (дата звернення 26.2024)

9. Dead Cells речі, які початківці повинні зробити в першу чергу : веб-сайт. URL: https://www.thegamer.com/dead-cells-starter-tips/ (дата звернення 26.02.2024)

10. Як вибрати найкращий ігровий двигун для вашої гри? : веб-сайт. URL: https://pinglestudio.com/blog/co-development/how-to-choose-the-best-gaming-engine-for-your-game (дата звернення 03.03.2024)

11. Який ваш процес вибору ігрового рушія?: веб-сайт. URL: https://www.linkedin.com/advice/3/what-your-process-choosing-game-engine-skills-gaming-industry (дата звернення 09.03.2024)

12. GameMaker: Studio відгуки 2024, перевірені відгуки, плюси та мінуси : веб-сайт. URL: https://www.capterra.com/p/158594/GameMaker-Studio/reviews/ (дата звернення 15.03.2024)

13. Різні художні стилі та як вибрати підходящий для гри : веб-сайт. URL: https://dreamfarmstudios.com/blog/game-art-styles-with-guide/ (дата звернення 23.03.2024)

14. Як вибрати художній стиль який підходить вашій грі : веб-сайт. URL: https://kevurugames.com/blog/choosing-an-art-style-for-your-video-game/ (дата звернення 24.03.2024)

15. Переваги використання ассетів з Unity Asset Store : веб-сайт. URL: https://www.gamingdebugged.com/2023/01/10/benefits-of-using-assets-from-the-unity-asset-store/ (дата звернення 01.04.2024)

16. Розпочніть свої творчі проекти та завантажуйте Unity : веб-сайт. URL: https://unity.com/download (дата звернення 02.04.2024)

17. Безкоштовне програмне забезпечення та послуги для розробників – Visual Studio : веб-сайт URL: https://visualstudio.microsoft.com/free-developer-offers/ (дата звернення 02.04.2024)

18. Просте рухання користувача в Unity 2D : веб-сайт. URL: https://gameartist86.medium.com/simple-player-movement-in-unity-2d-c9bfd46f8837 (дата звернення 03.04.2024)

19. Піксельна Пригода 1 з Unity Asset Store : веб-сайт URL: https://assetstore.unity.com/packages/2d/characters/pixel-adventure-1-155360 (дата звернення 02.05.2024)

20. Дракон Воїн 2Д Персонаж з Unity Asset Store : веб-сайт URL: https://assetstore.unity.com/packages/2d/characters/dragon-warrior-free-93896 (дата звернення 02.05.2024)

метадані

Заголовок
Створення відеогри

Автор
Шкорута В. Р.

Науковий керівник / Експерт
кандидат технічних наук Сергій Ващишак

підрозділ
King Danylo University

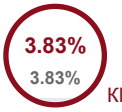
Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

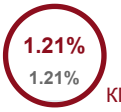
Заміна букв	ß	0
Інтервали	A→	0
Мікропробіли	·	1
Білі знаки	ß	0
Парафрази (SmartMarks)	a	16

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КП 2

25
Довжина фрази для коефіцієнта подібності 2

6965
Кількість слів

50924
Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	Розробка веб-сайту агрегатора новин з елементами веб-скрапінгу [redacted] 6/12/2023 King Danylo University (King Danylo University)	29	0.42 %
2	Розробка веб-сайту агрегатора новин з елементами веб-скрапінгу [redacted] 6/12/2023 King Danylo University (King Danylo University)	28	0.40 %
3	http://pdf.lib.vntu.edu.ua/books/2022/Knysh_2019_20.pdf	27	0.39 %