МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ СЕМЕНА КУЗНЕЦЯ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАТИКИ ТА КОМП'ЮТЕРНОЇ ТЕХНІКИ

Рівень вищої освіти	Перший (бакалаврський)
Спеціальність	Інформаційні системи та технології
Освітня програма	Інформаційні системи та технології
Група	6.04.126.010.18.1

ДИПЛОМНИЙ ПРОЕКТ

на тему: «Розроблення двовимірної мобільної гри в жанрі Arcade із використанням Unity 3D»

Виконав: студент Федір ДУБІНЕВИЧ

Керівник: к.т.н., доцент Олексій ГОРОХОВАТСЬКИЙ

Консультант: -

Рецензент: к.т.н, доц. кафедри Інформатики
Харківського національного університету радіоелектроніки
доц. Валентин ЛЮБЧЕНКО

Харків – 2022 рік

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

Факультет Інформаційних технологій
Кафедра Інформатики та комп'ютерної техніки
Освітній ступінь Бакалавр
Спеціальність 126 "Інформаційні системи та технології"

ЗАТВЕРДЖУЮ
Завідувач кафедри
інформатики та комп'ютерної техніки
_____ проф. Сергій УДОВЕНКО
«__»_____20 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ
Дубіневичу Федору

1. Тема проекту: «Розроблення двовимірної мобільної гри в жанрі Arcade із використанням Unity 3D».

Керівник проекту: Гороховатський Олексій Володимирович, кандидат технічних наук, доцент кафедри інформатики та комп'ютерної техніки.

Затверджені наказом ректора від "01" лютого 2022 р. № 178-С.

2. Строк подання студентом проекту: «09» червня 2022 р.

3. Вихідні дані до проекту: методи розроблення ігрового програмного забезпечення, Unity 3D, середовище розробки Visual Studio, програмний застосунок для роботи із графікою Inkscape.

4. Зміст розрахунково-пояснювальної записки:

Розділ 1. Історія ігор в жанрі "Аркада"

Розділ 2. Проектування гри в жанрі "Аркада"

Розділ 3. Розробка гри у жанрі “Аркада”

5. Перелік графічного матеріалу: актуальність проблеми (1 слайд), мета та завдання проектування (1 слайд), проектування гри (3-4 слайди), розроблення ігрового застосунку (3-4 слайди), висновки (1 слайд).

6. Консультація розділів дипломного проекту: Консультант відсутній.

7. Дата видачі завдання: «01» лютого 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
1	Розроблення плану дипломного проекту, ознайомлення з літературними джерелами за темою	01.02.2022-28.02.2022	
2	Аналіз історії жанру “Аркада”, оформлення першого розділу	01.02.2022-28.02.2022	
3	Проектування гри, оформлення другого розділу	01.03.2022-01.04.2022	
4	Перевірка дипломного проекту керівником	02.04.2022	
5	Розроблення гри, оформлення третього розділу	03.04.2022-31.05.2022	
6	Перевірка дипломного проекту керівником, перевірка на плагіат	31.05.2022	
8	Подання Голові Екзаменаційної комісії щодо захисту дипломного проекту	09.06.2022	

Студент Федір ДУБІНЕВИЧ

Керівник проекту Олексій ГОРОХОВАТСЬКИЙ

РЕФЕРАТ

Дипломний проект містить: 67 сторінок, 23 рисунки, 1 додаток, 28 джерел.

Об’єктом дослідження є метод розроблення гра жанру “аркада”.

Метою роботи є розроблення двовимірної гри в жанрі аркада із використанням рушія Unity 3D для мобільних пристроїв з прагненням позбутися основних недоліків жанру.

Методи розроблення включають метод аналізу методів розробки комп’ютерних ігор, що існують, метод комп’ютерного моделювання ігрового середовища.

В результаті виконання дипломного проекту розроблено двовимірну комп’ютерну гру, яка може бути використана у розважальному бізнесі та для зняття психологічного навантаження.

МОБІЛЬНІ ІГРИ, АРКАДИ, 2D ІГРИ, UNITY 3D, C#, ГЕЙМПЕЙ, ІГРОВИЙ РУШІЙ

ABSTRACT

Diploma project contains: 67 pages, 23 pictures, 1 supplement, 28 sources.

Object of the research is the method of the Arcade genre game development.

The goal of the work is the development of the two-dimensional Arcade game using Unity 3D engine for mobile platforms with intention to get rid of main genre problems.

Development methods include the analysis of existing methods for the development of computer games, computer modeling and implementation of gaming environment and gameplay.

As a result of diploma project the 2D game has been developed, that may be used in entertainment business or for the removal of psychological stress.

MOBILE GAMES, ARCADE, 2D GAMES, UNITY 3D, C#, GAMEPLAY, GAME ENGINE

ЗМІСТ

ВСТУП

Гра жанру «аркада» - це гра зі спеціально примітивним ігровим процесом. Іграми цього жанру зазвичай є такі, що напряму імпортовані з аркадного автомату, або схожі з ними за концепцією. Цей жанр ігрової індустрії вважається загальним, до нього часто відносять багато ігор інших жанрів, наприклад, “бійка” (fighter), симулятори перегонів, частково “перестрілка” (shooter). Аркадні ігри мають простий геймплей (ігровий процес, взаємодія гравця і ігрового всесвіту, гравцю не потрібно вивчати налаштування гри, у більшості випадків він може просто почати грати. До аркадних ігор можна віднести популярні Shadow Fighter, Vector, розвиток ігор цього жанру постійно продовжується та еволюціонує.

РОЗДІЛ 1. ІСТОРІЯ ІГОР В ЖАНРІ “АРКАДА”

1.1 Історія виникнення жанру

Аркадні ігри часто мають такі характерні особливості:

– гра на одному екрані - цю особливість мало дуже багато ігор, особливо в “золоту епоху аркад”. Але з часом в іграх ставало більше різноманітних рівнів, тому в сучасних іграх зараз така особливість зустрічається нечасто;

– безкінечна гра - дуже частою особливістю аркади є створення безкінечного рівня, який буде або основою гри, або додатковим викликом для гравців, які будуть намагатися обіграти самих себе, ставити рекорди, та ін.;

– безліч життів - ця механіка з’явилася завдяки походженню аркад, адже на аркадних автоматах грали по 1 разу, за жетон, тому для того, щоб новачки могли ознайомитися з правилами і механікою гри, в аркадах було передбачено декілька життів;

– ігровий рахунок - майже в кожній аркаді є різноманітні очки (бали), які нараховуються за різноманітні задачі, які вже залежать від геймплею гри. В деяких іграх очки даються за час виживання, в деяких – за збирання різноманітних елементів, або вбивства ворогів, або ж за декілька одразу;

– швидке навчання і простий геймплей - цією особливістю аркади також зобов’язані своєму походженню. Ніхто не хотів, щоб гравець годинами вивчав правила гри, стоячи в черзі на автомат.

Завдяки простоті геймплею розробка аркадних ігор не вимагає величезних бюджетів, так само, як і професійних команд розробників. Якщо зайти в магазини застосунків, може скластися враження, що аркади це і є весь ринок ігор, до того ж зроблений за одним шаблоном. Таке враження складається тому, що багато починаючих розробників ігор починають з простого – аркад, та і деякі великі

компанії намагаються не ризикувати і отримувати найбільшу вигоду при найменшій кількості витрачених ресурсів. Також аркади часто плутаються з іграми з іншим жанром з простим геймплеєм - казуальними іграми. Через такий наплив одноманітних, простих, і неоригінальних ігор, багато дійсно цікавих проектів залишається невідомими.

Останніми роками ігри настільки стрімко розвиваються і впроваджуються у наше життя, що ми цього майже не помічаємо. Навіть якщо говорити про ігри для розваг, то можна згадати такі новини як “Minecraft почали використовувати для навчання у школах” [1], та інше. Це може бути пов’язане з тим, що у створенні і популяризації ігор широкого використання набула психологія, адже основний заробіток розробника - це час, який користувач провів у грі, тим самим піднявши її активність, подивився рекламу, частота повернень гравця у гру і т.д. Саме за цією причини молодь часто проводять багато часу в іграх і гаджетах – над розважальним програмним забезпеченням працюють спеціалісти, які розуміють, як заволодіти часом користувача. Вчителям, викладачам доводиться боротися за час учнів і студентів з цілими командами розробників. І аби зрівняти шанси, дехто вдається до хитрощів, таких як використання ігрових засобів, які вже завоювали увагу аудиторії, в своїх інтересах.

Використання ігор для навчання також є достатньо популярним у засвоєнні професійних навичок, наприклад, існують симулятори для навчання водіїв, солдатів, пілотів. Безумовно, слід відзначити, що не можна опанувати всі професії із використанням віртуального моделювання, наприклад, професію медичного працівника. Тим не менш, використання віртуальної реальності знаходить застосування останнім часом і у медицині також.

Окрім того ігри з 2011 року США і Американським Національним Фондом було визнано окремим видом мистецтва, в деяких країнах їх прирівняли до

спорту. Отже, потрібно визнати - ігри з нами надовго, а може і назавжди, а сфери їх використання обмежуються лише нашою фантазією, і бажанням.

Ігри підрозділяються за такими класифікаціями як :

- Жанр
  - Аркада
  - Пригодницька
  - Екшн
  - Рольові ігри
  - Стратегія
  - Симулятор
  - Головоломка
  - Навчальна
  - І безліч змішаних жанрів, та інших під жанрів.
- Платформа
  - Крос-платформова гра (Гра доступна на різних типах пристроїв)
  - Для мобільних телефонів
  - Для персональних комп’ютерів
  - Для консолей
- Візуальний вигляд
  - Текстова
  - 2D
  - 3D
  - VR
  - AR
- Кількість гравців
  - Для одного гравця
  - Багатокористувацька

- Необхідність інтернет з’єднання
  - Онлайн
  - Офлайн

Це лише основні категорії класифікації, окрім них є класифікація за рейтингами, методами поставки та інші. Останнім часом в більшості категорій помічено тенденцію до стирання кордонів між собою, саме тому варто розуміти, що часто нові ігри дуже важко класифікувати.

Зупинимося більш детально на жанрі “аркада”. Історія жанру починається в далеких 1970-х роках. Перші ігри жанру були створені для ігрових автоматів, пізніше ці автомати почали називати “Аркадними” звідси в подальшому і пішла назва жанру.

Першою грою, яка поклала основу виникненню жанру стала Computer Space. Її геймплей був до неможливості простим - за 99 секунд гравцеві потрібно було знищити якомога більше космічних кораблів, накопичуючи очки. Часто в гру грали компаніями, хто набрав більше очок - перемагав.

Першими іграми на комп’ютерах також були саме ігри даного жанру, через те, що тогочасні обчислювальні можливості комп’ютерів були досить слабкими, а аркади через простоту геймплею не мали високих вимог. Йшов час, індустрія ігор розвивалась, аркади на комп’ютерах втрачали популярність, росли обчислювальні можливості, з’являлись нові жанри, з більш різноманітним геймплеєм, сюжетами, графікою.

Але із поширенням мобільних телефонів почалося відродження популярності жанру, адже інші, більш сучасні жанри, потребували більше часу для гри, а на телефонах люди могли скоротити час в простому геймплеї. Так в жанрі аркад почали з’являтися безліч піджанрів. З появою смартфонів і набуття їх популярності почалось масштабне зростання кількості аркадних ігор завдяки їх

доступності. Саме аркадні ігри можна побачити майже у кожного у метро, черзі і тд.

Окрім комп'ютерів і мобільних пристроїв варто згадати портативні приставки, в свій час для деяких ігор аркадного жанру вони створювались спеціально, приставка підтримувала лише гру для якої була створена. В якості прикладів можна назвати Mario Adventure.

1.2 Відомі проекти у жанрі "аркада"

Жанр існує дуже давно, тому безліч легендарних проектів (наприклад Pac-man, Super Mario Bros, Mortal Kombat та багато інших) відноситься до жанру «аркад». "Золота епоха аркад" була в 1970-1980х роках [2]. Більшість ігор, створених у цих роках, переносилися спочатку на персональні комп'ютери, потім на мобільні телефони. І продовжували приносити задоволення користувачам і фанатам протягом десятиліть. Нижче наведено найпопулярніші проекти цього жанру.

Pac-man - перше гра з'явилась на аркадному автоматі в кінотеатрі міста Токіо в 1980 році. Створений він був філіалом компанії Namco. Через майже сорок років та ж сама компанія випустила гру на мобільні телефони.

Геймплей гри за цей час не зазнав змін. Жовтий "колобок" рухається лабіринтом, збирає точки, і тікає, або ж в певні моменти з'їдає привидів. В класичній грі було 255 рівнів, на 256 гра ламалась, розробники не очікували, що хтось зможе до нього дійти. Складність гри повинна була підвищуватись з кожним рівнем, що робило б неможливим зайти занадто далеко. Але деякі гравці розгадали алгоритм і змогли легко пройти гру. На основі цієї гри для мобільних телефонів було створено безліч схожих ігор, просто з іншим візуальним

відображенням. Гру було додано до схованих ігор Google, в неї можна пограти не завантажуючи її, написавши її назву в пошуковій стрічці (рис. 1.1).

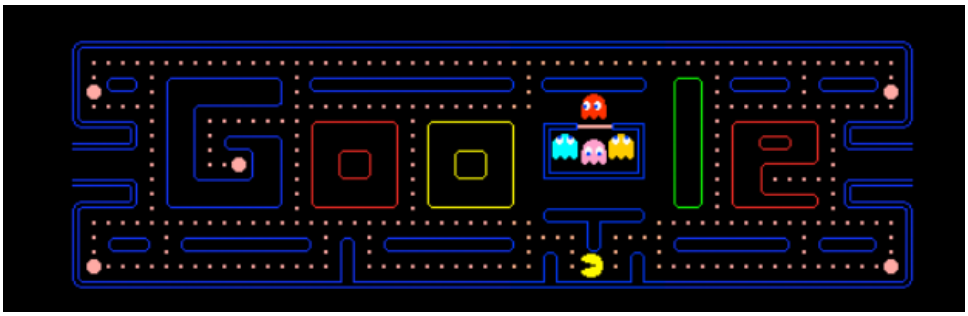

Рисунок 1.1 – Вигляд гри Pac-man в Google

Pinball - це одна з найстаріших аркад у світі, вона була придумана в 18 сторіччі при дворі французького короля Людовика XIV [3]. З того часу правила гри майже не змінились - є нахилене пласке поле з лузами, на полі встановлені різноманітні перепони. З вищої частини поля запускаються кульки, які відштовхуються, стрибають, але все одно котяться в лузи і приносять очки.

Ця гра існувала також і на ігрових автоматах, де була однією з найпопулярніших, гравцю потрібно було якомога довше утримати кульку на полі. В 90-х і на початку 2000 Microsoft через шалену популярність гри на автоматах створила віртуальну копію гри і вбудувала її в Windows NT 4.0. Пізніше гра з’явилася в магазинах застосунків, де в неї з’явилось безліч ігрових столів, через що вона почала набридати не так швидко. Навіть зараз ігри такого формату можна часто зустріти в магазинах ігор. На рисунку знизу зображено сучасну адаптацію гри Pinball – 3D Pinball Space Cadet (рис. 1.2).

Subway Surfer - ця гра в далекому 2012 році спричинила фурор, майже неможливо було не знайти людину, яка не грала в цю гру. Простота і складність геймплею одночасно змогли приманити безліч гравців. Навіть зараз, у 2022 році ця гра знаходиться у топ-10 найбільш популярних додатків магазину мобільних

застосунків Google Play (рис. 1.3). Сюжет гри - художник вирішив намалювати графіті в депо, і його побачив охоронець, ну а далі гравець намагається від нього втекти. Чим довше він тікатиме, тим складнішим буде ставати геймплей, та тим більше очок він отримуватиме. Проте теоретично геймплей може бути нескінчений, як і в багатьох аркад.

Рисунок 1.2 – Вигляд 3D Pinball Space Cadet

На основі механіки гри було створено тисячі копій, якісних і не дуже, деякі з них теж пробились в лідери рейтингу.

Red Ball 4 - на старих кнопкових, чорно-білих мобільних телефонах дуже популярною була гра, в якій гравець проходив рівні, керуючи м’ячиком. Судячи з геймплею гри Red Ball 4, її розробники надихались саме вищезгаданою грою. Окрім простого та цікавого геймплею в Red Ball 4 є цікавий та якісний сюжет, хоч цей компонент і не властивим більшості аркад.

Рисунок 1.3 – Постер-реклама гри Subway Surfer з сторінки в Google Play

Гравець у грі керує червоним м’ячем з обличчям, катаючи його по 2-д простору та підстрибуючи, знищуючи таким чином ворогів, стрибаючи на них з вищих точок, збираючи зірки та долаючи перешкоди (рис. 1.4). В грі присутні безліч головоломок, ворогів і боси, більшість ворогів - це різноманітні ворожі квадрати з обличчями. Гра складається з багатьох рівнів, і в деяких частинах серії є безкінечні рівні.

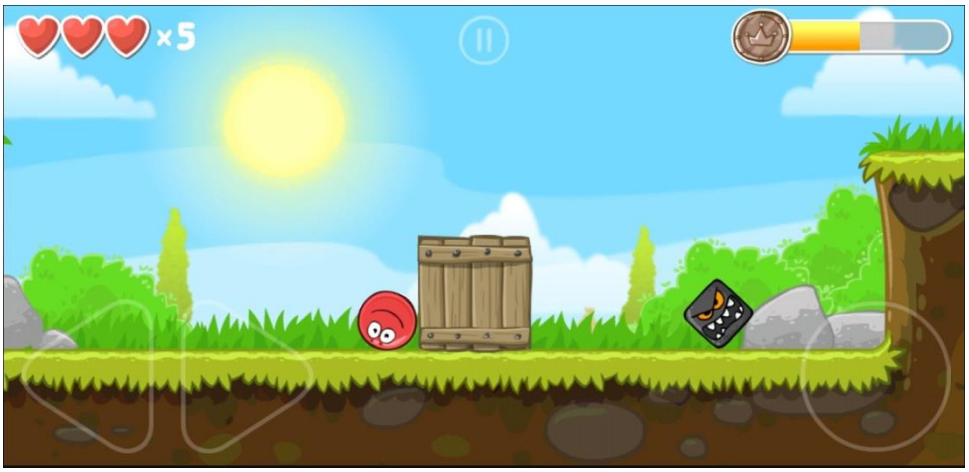

Рисунок 1.4 – Внутрішній інтерфейс гри Red Ball 4

Після проведеного аналізу можна зробити висновок, що гравці, які обирають жанр аркада полюбляють долати труднощі, але в той самий час люблять простий геймплей, притаманніий жанру, завдяки існуючій у сучасній ігровій індустрії розмитості кордонів , при доданні цікавих механік, їх ускладнені, та інших поліпшеннях. Можуть допомогти не просто зберегти аудиторію, яку приваблює аркадний жанр, а й залучити гравців, яким подобаються інші жанри.

1.3 Недоліки відомих ігор у жанрі “аркада”

Хоч аркада здобула друге дихання на мобільних пристроях.ї, все одно із появою таких жанрів, як казуальні ігри, підвищенням потужностей мобільних пристроїв, аркадні ігри почали займати дедалі менше відсотків ринку ігор. Проте, через зростання ринку, кількість прихильників аркад не зменшується, просто більша частина нових користувачів захоплюються іншими жанрами.

Нестача великих розробників і видавців

Аркади зазвичай являють собою невеликі проекти, тому великі студії не беруться за їх розробку. Кваліфіковані спеціалісти, які мають потужну техніку, не

часто беруться за такі проекти. Та і великі видавці не поспішають зв’язуватися з інді-студіями, які зазвичай займаються розробкою аркад.

Великі студії розробки часто орієнтуються за популярність серед користувачів. В наші часи жанр «аркада» і його піджанри не так популярні, як великі 3D ігри, які вимагають величезного часу розробки і великих зусиль, але все ж, у порівнянні з аркадами, мають значну аудиторію на одному проекті. Невеликі казуальні проекти, які робляться швидко, мають більшість витрат при їх розробці і випуску на рекламу та швидко окупаються також часто є більш популярними за сучасні аркади.

Можна сказати, що ігри жанру “аркада” не витримують конкуренції жанрів серед великих розробників і великих видавців. Інколи певний великий видавець може звернути увагу на цікавий проект в жанрі “аркада” від інді-студії, але, на жаль, це відбувається не часто.

Проблема маленьких розробників і видавців

Здавалося б, що ігри жанру аркада за притаманними їм характеристиками чудово підходять для починаючих студій, невеликих компаній розробників і маленьких видавців. І це дійсно так, але конкуренцію на цьому сегменті ринку складає порівняно з жанром “аркада” молодий жанр ігор - “казуальний”. Обидва жанри досить прості в розробці проекту, мають високу популярність, але відсоток сегменту, який тримає жанр казуальних ігор та його під жанри, поступово росте.

Іграм жанру “аркада” притаманна певна складність геймплею, як називають це гравці - “хардкорність”. Саме завдяки цьому ігри цінуються, гравці полюбляють долати складнощі, ставати краще і сильніше, нехай навіть у грі. А жанру “Казуальні ігри” притаманна абсолютна простота та банальність. Саме завдяки цьому в 2000-х роках вони різко почали набирати популярності. В них немає зростаючої складності геймплею, вони швидко і дешево розробляються, більша частина їх бюджетів витрачається на просування продукту. Через їх

однотипність і схожість зазвичай просуванням казуальних ігор займаються маленькі видавці, адже не кожен видавець хоче випускати десяток одноманітних ігор.

Аркадам же хоч і притаманний простий геймплей, він поступово ускладнюється, ставить нові виклики гравцям, і дуже часто в основу гри закладений так званий “нескінчений рівень”, коли гравець може грати настільки довго, наскільки йому дозволять навички і вміння, адже складність буде рости з часом гри. Тому далеко не всі гравці вміють в них грати, багато кому складно, і вони йдуть шукати щось простіше.

Аркадні ігри також важче розробити, адже там повинний бути хоч і простий, але “штучний інтелект” у ворогів гравця, та і до того ж він повинен підвищувати складність гравцеві. Тому багато маленьких студій починають з казуальних проектів, а не з аркади. Видавці ж надають перевагу казуальним іграм, через те, що вони приносять більший прибуток порівняно з меншими витратами на рекламу і мають більшу аудиторію потенційних гравців

Старіння ігор

Багато популярних аркад є «перезапусками» ігор з “золотої епохи аркад”, деякі є новими версіями, але саме старі ігри - це та невелика частина аркад, за які беруться великі студії розробки. Для створення гри використовують ігрові рушії - але далеко не всіх спрощень, які може виконати програмне забезпечення достатньо розробникам ігор - чим більше гра, тим більше однотипових моментів у ній є, і тим більше можна спростити, оптимізувати. Тому багато компаній під час розробки гри модернізують ігровий рушій, звісно цим здебільшого займаються великі компанії, і хоч зараз жанр “аркада” серед них не є популярним, але в 1970-1990х роках, він був дуже популярним, і розробкою ігор у цьому жанрі займалися тодішні великі компанії, деякі з них з часом виросли ще більше, і вже не випускають аркади, деякі з них залишились такими ж, деякі перестали існувати.

Але деякі з компаній продовжують створювати нові аркади, або ж перезапускають старі, аби порадувати фанатів оновленою графікою, або ж можливість пограти у гру дитинства на смартфоні. І ось тут у них виникають проблеми.

Адже багато старих рушіїв давно перестали випускати оновлення і модернізуватися, програмна частина ігор створювалась на мовах які зараз можуть не використовувати, або ті які безнадійно застаріли. Але рушії на яких було створено аркаду були модернізовані певними компаніями дуже сильно, настільки, що для того щоб створити нову версію гри, або гру яку будуть підтримувати нові пристрої, на нових системах, їм потрібно провести таку ж модернізацію на новому рушії, а це займає дуже багато часу та коштів. Після цього потрібно переписати код гри на нову, оптимізовану мову. Це все потребує величезних затрат часу та коштів, настільки що випуск аркади стає неймовірно дорогим.

Але це все за умови якщо код гри і первісні файли збереглися. Digital Leisure розробник популярних ігор у жанрі “аркада” Dragon's Lair и Mad Dog McCree в розмові з блогером Джошом Андерсоном розповіли що первісний код неможливо відновити через формат носія на якому він зберігався. Адже тоді не існувало флешок, та дисків якими ми зараз користуємось. А тогочасні носії просто немає до чого підключати, якщо на них збереглась інформація. Взагалі в статті Джош Андерсон описує розмову з 14 видавцями які розповіли про проблему старіння ігор, і там йому розповіли про такі неймовірні історії як продаж цілих шаф, і складів з прототипами ігор, або їх версіями. Документами на їх та інше. [6]

Таким чином відновити стару гру дуже важко, а створити нову - за це можна накликати на студію гнів тих самих фанатів для яких гра створювалась.

“Логістичний кошмар”

В своєму дослідженні Джош Андерсон згадав також таку проблему як “логістичний кошмар” [6]. Якщо студія проіснувала майже 50 років, за цей час

створила сотні проектів, якщо вона постійно займалась перенесенням інформації з носіїв які старішали, на нові, нічого не втратила, і хоче перевипустити одну з ігор, спеціалістам потрібно розібратися в роботах людей які створювали гру 50 років тому. Знайти всі її файли, відрізнити актуальні файли від прототипів, це складно навіть для людини яка створювала гру. А новому спеціалісту потрібно це зробити, розібрати, і переписати, серед сотень проектів, тисяч файлів, і мільйонів зображень.

Загалом окрім основних вищезгаданих проектів у ігор жанру “аркада” є ще безліч незгаданих. І не зважаючи на такі масштабні і великі проблеми, деякі розробники перевипускають ігри “Золотої епохи”, випускають нові, і продовжують існування жанру “аркада”.

1.4 Постановка задачі

Для отримання готової гри в жанрі “аркада” потрібно пройти шлях від ідеї до фінального проекту. Після вибору жанру проекту, його візуального стилю та інших деталей треба виконати проектування гри.

До проектування входить: створення дизайн-документу (необхідний для того, щоб пояснити програмістам, що потрібно прописати у грі, що зможе робити гравець), створення UML-діаграми, або ж діаграми використання (на вигляді взаємозв’язків програміст побачить, які сторінки гри будуть пов’язані між собою, та як само). Після цього в проектуванні створюється прототип інтерфейсів користувача – головного меню, екрану гри, налаштувань та інших інтерфейсів за потреби.

Останнім етапом проектування гри є створення прототипу гри. Прототип може складатися з будь яких графічних елементів, його основна задача –

відпрацювати основну механіку гри, це можуть бути навіть прості фігури на зразок квадратів, кіл, та ін. Проте в ньому повинні бути усі ключові механіки. Якщо тестувальникам цікаво грати в прототип, де немає зображень, текстів, та ін., це означає, що механіка цікава і можна приступати до розробки проекту. Проте така модель абсолютно не підходить для ігор, де основою є сюжет або чудова візуальна частина.

Наступна задача в розробці гри – реалізувати та розробити застосунок, тобто ігрову програму. Тут як і в проектуванні можна поділити задачу на декілька частин, які включають розробку програмної частини, створення і додавання візуальних елементів, додавання звукових елементів, фінальне тестування застосунку. Розробка програмної частини здійснюється в спеціальному середовищі - ігровому рушії та сумісному з цим рушієм редакторі коду, у якому пишуться скрипти (команди) для гри. На цьому етапі проводиться створення і модернізація фізики гри, збереження і завантаження інформації та прогресу гравця (прогрес гравця - все чого гравець досяг у грі, отримана ним ігрова валюта, придбані ігрові персонажі, їх ступінь покращення та інше), створення елементів керування ігровим персонажем, створення ігрових ворогів, поведінку яких буде контролювати внутрішньо ігровий штучний інтелект, створення елементів керування інтерфейсом головного меню, створення налаштувань.

На етапі створення і додавання візуальних елементів(графіка гри - задній фон сцен, зображення кнопок, панелей, різноманітних ігрових елементів та ін.) створюється перелік елементів які можна створити самостійно, і ті елементи які необхідні, але не вистачає навичок для їх створення. Графічні елементи, які можна створити самостійно - створюються в спеціальних, попередньо обраних графічних редакторах, після чого імпортуються в середовище ігрового рушія, для компонування візуальних елементів з уже існуючими частинами гри (заміна

простих графічних елементів, квадратів, прямокутників, на створені графічні елементи).

Ті візуальні елементи які немає можливості створити власноруч - розшукуються на різноманітних ресурсах, наприклад на безкоштовних “стоках” - сайтах на яких зберігається безліч зображень та інших візуальних елементів які розповсюджуються згідно ліцензій які дозволяють безкоштовне комерційне використання, або ж в магазинах ігрових ресурсів - це магазини, де розповсюджуються ресурси, які можна використовувати в грі після придбання, там також зустрічаються безкоштовні елементи. Усі візуальні ресурси потрібно підбирати в одному стилі, для того щоб коли вони будуть імпортовані в гру око гравця не чіплялося за об’єкт який різко відрізняється. Така сама процедура чекає і на звукові файли - їх потрібно знайти на спеціальних платформах, обов’язково звертаючи увагу на ліцензію і на те які дозволи нею передбачені, і подальша інтеграція у гру.

Метою виконання дипломного проекту є розробка гри в жанрі «аркада» на платформі Unity 3D. Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) виконати аналіз вимог и розробити специфікації;

2) виконати проектування ігрового застосунка;

3) реалізувати ігровий застосунок;

4) протестувати розроблений ігровий застосунок.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ГРИ В ЖАНРІ “АРКАДА”

2.1 Розробка алгоритму реалізації проекту

Процес розробки гри мало відрізняється від процесу розробки будь-якого іншого програмного продукту. Він складається з 4 великих етапів:

1. Ідея
2. Проектування
3. Розробка
4. Впровадження і підтримка

Після появи ідеї треба почати над нею працювати, або занотувати її, інакше ідея може залишитися ідеєю і не наблизитися до реалізації. Після формування ідеї зазвичай складається дизайн-документ, де на основі ідеї складається сюжет гри і викладається у форматі, доступному для гравця. У командах розробників цю задачу зазвичай виконує “гейм-дизайнер”. Під час підготовки дизайн-документу, оцінки своїх можливостей, гра може змінити свою першочергову ідею і перетворитися в щось інше. Дизайн-документ дозволить навіть в одиночній розробці запобігти виникненню помилок, нічого не втратити і не забути.

Наступний крок - обрання сетингу [4] (сукупності елементів\найменувань які характеризують ігровий світ) на основі вподобань цільової аудиторії, це сильно спростить подальшу розробку. Після обрання сетингу потрібно обрати засоби розробки продукту.

Засобом розробки відеоігор є ігровий рушій. Від його вибору залежить як швидкість так і майбутня працездатність застосунку. Мова програмування раніше залежала від платформи, для якої створювався застосунок, але враховуючи, що ігровий рушій спрощує процес розробки, то вибір мови програмування, по суті, вирішується вибором рушія, адже в рушіях є обмеження, якими мовами в них

можна працювати. Ігровий рушій відповідає за низькорівневий опис фізики об’єктів, правил рендерингу графіки та ін.

Після вибору рушія і складення дизайн-документу першим кроком потрібно створити прототип майбутнього застосунку, на якому можна буде відпрацювати необхідні ігрові механіки. Зазвичай в цьому прототипі тестується чи буде дана механіка популярна, і чи є сенс доводити проект до фінального стану. Якщо в прототип цікаво грати без графіки, текстур, та іншого - значить він може стати успішним проектом, і його потрібно спробувати завершити.

Наступним етапом розробки застосунку є альфі версія - “сира” версія продукту, в якій допустимі глюки, тестуванням цієї версії займаються самі розробники. Після чого відбувається доопрацювання рівнів, механік, сюжетних елементів, відеороликів та діалогів. Окрім того проводиться виправлення помилок, які було знайдено в ході проведення альфа-тестування.

Після цього наступає час розробки бета-версії. Бета-версія створюється для того, щоб протестувати гру на несправності, фактично бета версія - це майже готова гра. В ній може не бути в наявності деяких елементів, які не впливають на геймплей. При тестуванні бета-версії перевіряється абсолютно все. Деякі продукти можна спробувати саме на цьому етапі, адже тестування бувають публічні, таким шляхом розробник і видавець можуть значно спростити вартість і швидкість тестування. Якщо гра успішно проходить тестування, відбувається доопрацювання і виправлення критичних помилок, після чого відбувається фінальна збірка ігрового проекту. В станах альфа або бета тесту деякі проекти можуть знаходитися роками, деякі з них так і не побачать релізу. Але ті які побачать реліз, оновлюються і підтримуються після нього.

2.2 Проектування гри

2.2.1 Опис геймплею

Дизайн-документ - це детальне описання гри, що розробляється. Дизайн-документ створюється командою розробників, або одним розробником, зазвичай гейм дизайнером - для організації роботи розробників. В дизайн-документі поставлені задачі, і в відповідності до нього організовується робота дизайнерів, художників і програмістів в ході створення гри. Інколи дизайн-документ називають концепт документом. Дизайн-документ дописується на протязі усього проекту. Він повинен складатися з тексту, зображень, діаграм, концепт-артів, написаний не в художньому стилі.

Дизайн-документ повинен містити:

- схему гри - що повинен робити гравець, яка кінцева мета, що заважає її досягненню;
- інтерфейс - детально описаний функціонал інтерфейсу, що можна робити, як, які клавіші та кнопки що виконують;
- ігрову механіку - опис ігрового всесвіту, його об’єкти, їх характеристики, формули пересування, бою та ін.;
- програмні механізми та алгоритми - характеристики ігрового рушія, штучного інтелекту, мережевого коду, інтерфейсу, редактора карт, звуків;
- графіку - перелік необхідних моделей, анімацій, відеороликів, фонів;
- музику і звуки - музикальна тема, спосіб відображення звуків, необхідні набори звукових ефектів;
- сюжет - план сюжету, сюжетні завдання гравця, заплановані карти для рівнів;
- ігровий всесвіт - перелік персонажів, монстрів, їх параметри, і можливі розташування, механіки виробництва та ін.;

- співробітників, їхню заробітну плату, терміни виконання проекту, і план робіт.

Геймплей гри є глобальним та часто суб’єктивним поняттям, але для спрощення його можна умовно поділити на базовий і глобальний.

Базовий геймплей - це частина гри, в якій гравець буде проводити найбільшу кількість часу, і одночасно - основна мета, заради якої він запускає гру. Базовий геймплей для застосунку, який розробляється в цьому дипломного проекті - бій обраним гладіатором на арені. Ціль в будь якому з обраних режимів одна - перемогти і отримати якомога менше ран. У базовому геймплеї на арені гравцю буде доступне керування гладіатором, його переміщеннями, атаками та захистом. Керування переміщенням гладіатора в чотирьох напрямках буде відбуватися лівою рукою за допомогою елементів управління у вигляді стрілок. Для керування атакою і захистом передбачено 2 сенсорних клавіші в правому нижньому куті екрану.

В лівому верхньому куті знаходиться елемент, який відображає запас здоров’я гравця. Якщо здоров’я закінчиться до того, як гравець знищить усіх ворогів - гравець програє. Якщо знищить усіх ворогів - виграє.

Камера сцени прикріплена до гладіатора, яким керує гравець, і переміщується за ним по ігровому полю.

Після завершення ігрової сесії - одного бою, з’являється вікно результату, яке відрізняється зовнішнім виглядом в залежності від виграшу або програшу. У цьому вікні відображаються результати про бій: скільки та яких ворожих суб’єктів було знищено, кількість отриманої ігрової валюти (досвіду, трофеїв, срібних монет, золотих монет, квитків турніру, імператорських квитків). Ігрову валюту гравець може витрачати в глобальному геймплеї.

Глобальний геймплей включає налаштування для вибору персонажів, тобто геймплей, який гравцю доводиться проходити, аби зіграти в базовий геймплей. В

глобальному геймплеї буде створено механіки, які більше притаманні не жанру Аркада (жанр -метод систематизації ігрових світів, і проектів), а жанру Стратегія, наприклад, прокачка бази гладіаторів - лодосу, тренування, найми, оснащення самих гладіаторів, та спорядження для них.

Як перехід до базового геймплею і вибору його режиму, так і перехід до глобального геймплею відбувається з головного екрану. На ньому розташовується 2 сенсорних кнопки, ліворуч кнопка “Лодос”, праворуч - “Грати”. Перша відправляє гравця у візуальне зображення лодосу. Друга - до вибору режимів, після чого і запускається базовий геймплей.

Коли гравець відправляється у “Лодос”, йому відкривається можливість редагувати власний дім гладіаторів. В залежності від того, наскільки відомий його “Лодос”, йому відкривається можливість побудувати кімнати певної спеціалізації - наприклад, лікарню або тренувальний зал. Вибравши кімнату, гравець може розташувати її на вільному місці екрана, після чого вона з’явиться на схемі будинку-Лодосу. Гравець також може обрати спеціаліста, який буде керувати кімнатою, цей спеціаліст зможе покращувати властивості і характеристики, які надаються кімнатою. Від рівня слави Лодосу залежить наскільки великий буде будинок лодосу. Чим більше слави - тим більше місця для кімнат. Кількість гладіаторів залежить від кімнат, в спеціалізованих кімнатах гладіаторів можна тренувати, змінювати їх зовнішній вигляд та лікувати після боїв. Також саме в лодосі можна проводити найм нових гладіаторів та купляти і ставити спорядження на гладіаторів.

2.2.2 Діаграма використання

В процесі аналізу вимог до застосунку було створено UML-діаграму (рис. 2.1)). Гра розрахована на одного гравця, який здійснює керування гладіатором.

При вході в гру кожен користувач стає Гравцем. Гравець потрапляє в Головне меню в якому є наступні дії:

- Відкриття налаштувань - гравець може провести такі дії як:
  - Налаштування звуку - гучність, наявність музики, звуку, ефектів.
  - Перегляд інформації про гру - короткий опис про правила гри, творців і т.д.;
  - Авторизація - Доступ до :

    Реєстрації - створення нового акаунту

    Вхід в акаунт - доступ до збережених даних і збереження нових.
- Вибір режиму гри - гравець може вибрати режим серед поданого переліку, після чого проводиться гра:
  - В грі гравець може керувати персонажем, гладіатором.
- Керування лодосом - гравець може здійснювати такі дії для керування власним лодосом:
  - Лікування гладіаторів - вибір гладіатора, запуск його лікування для відновлення здоров’я.
  - Купівля\найм гладіаторів - набір гладіаторів для лодоса, якими в подальшому гравець зможе керувати, битися, тренувати і т.д.
  - Купівля спорядження для гладіаторів - вибір спорядження з кращими характеристиками для збільшення шансів на перемоги.
  - Вибір спорядження гладіатору - вибір активного спорядження, яке гладіатор буде використовувати в бою.

- Тренування гладіатора - підвищення його характеристик.

- Вихід з гри - закриття гри.

Гладіатор - об’єкт яким керує гравець.

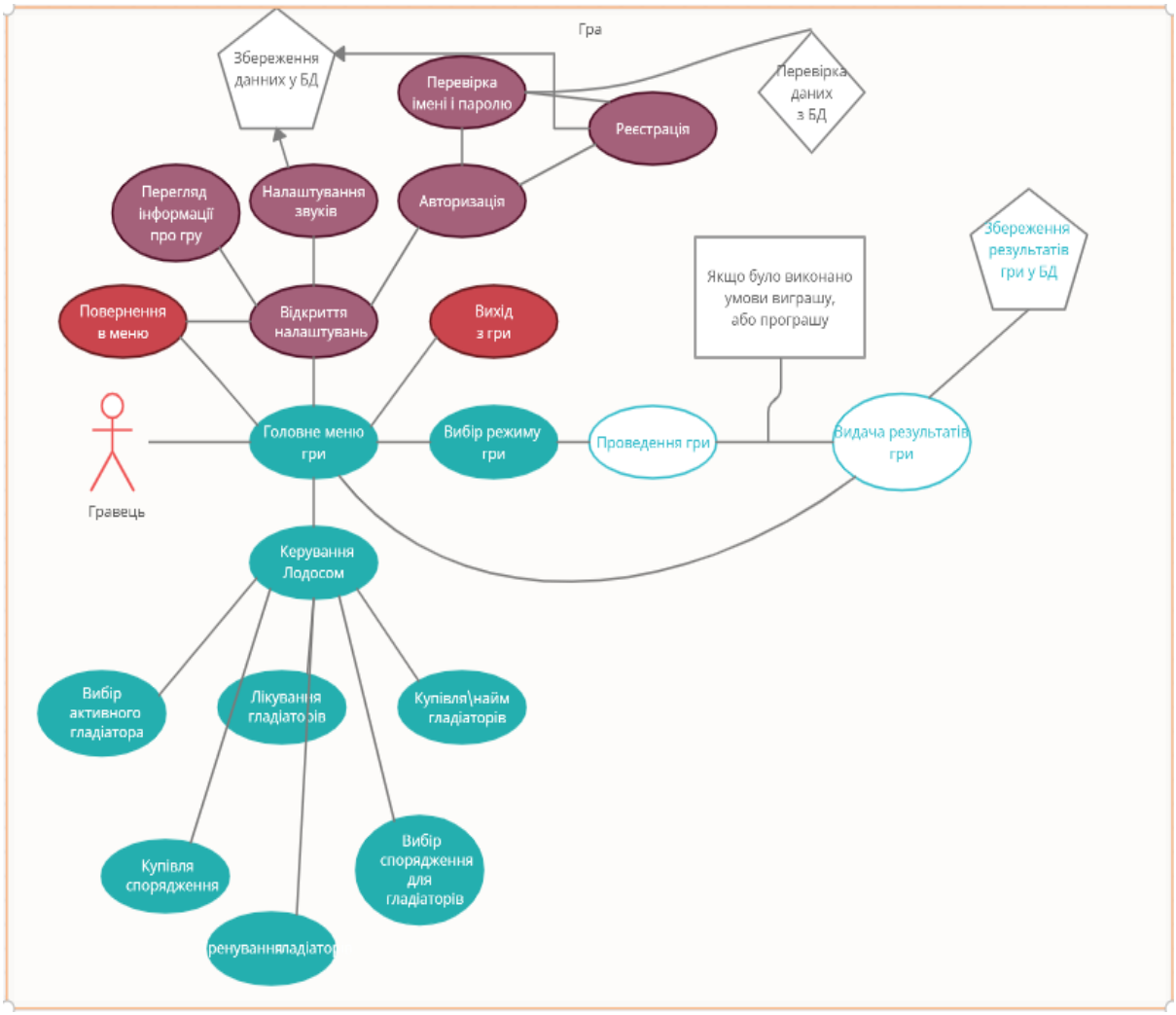

Рисунок 2.1 – UML-діаграма для гри яка створюється в якості дипломного проекту

2.3 Створення прототипів гри

2.3.1 Прототипування інтерфейсу користувача

Інтерфейс гри повинен бути зручним для користувача, потрібно знайти баланс між навантаженням елементами керування інтерфейсом та їх доступністю. Для того, щоб не загромаджувати інтерфейс користувача, певні групи елементів керування можна сховати в окремі “вторинні інтерфейси”, наприклад, дуже часто саме так ховають всі елементи налаштувань (звуки, мову, та інші). За часи існування інтерфейсів користувача спеціалістами було визначено характеристики, які притаманні досконалому інтерфейсу користувача:

- ясність - інтерфейс уникає двозначності, і робить все зрозумілим доступними методами;
- виразність - легко зробити інтерфейс прозорим завдяки надмірному уточненню та позначенню всього, але це призводить до його “роздування”, де одночасно на екрані є занадто багато елементів. В цьому випадку знайти те, що ви шукаєте, важко, і через це інтерфейс стає нудним у використанні. Справжнє завдання у створенні досконалого інтерфейсу полягає в тому, щоб одночасно зробити його стислим і зрозумілим;
- відповідність - інтерфейс повинен мати якісний зворотній зв’язок, відображати чи опрацьовуються дані, які ввів користувач;
- послідовність - інтерфейс повинен відповідати сценарію всього продукту;
- знайомство - легке розуміння навіть для користувачів, які перший раз користуються продуктом;
- естетика - це не обов’язкова деталь, але чим приємніше користувачеві користуватися інтерфейсом, чим більше він йому подобається, відповідно, тим більше шансів на те, що в користувача залишиться в цілому приємне враження про продукт;
- продуктивність - “час це гроші”, тому досконалий інтерфейс повинен зберігати час користувача;

- “пробачення” - інколи користувачі допускають помилки в використанні інтерфейсу, досконалий інтерфейс не повинен їх за це карати, а повинен надавати засоби для усунення цих помилок.

Прототип інтерфейсу можна зробити навіть у Paint, проте в спеціалізованих додатках, таких як Figma, ProtoPie, Marvel є можливість зробити функціональний інтерфейс, тобто є можливість заздалегідь продумати переходи від інтерфейса до інтерфейса, в деяких спеціалізованих додатках також можна налаштувати ефекти для кнопок інтерфейсу, та ін.

Перший інтерфейс, який зустріне користувач при вході в гру - інтерфейс головного меню (рис. 2.2), саме він буде поєднувати “базовий” і “глобальний” геймплей, в цьому інтерфейсі повинні знаходитися налаштування.

Рисунок 2.2 – Прототип інтерфейсу головного меню:
1 - кнопка налаштувань і захований інтерфейс налаштувань; 2 - кнопка Грати і захований інтерфейс вибору режимів

Як можна побачити на прототипі, інтерфейс налаштувань (рис.2.3) було спеціально “заховано”, щоб не перевантажувати екран великою кількістю кнопок.

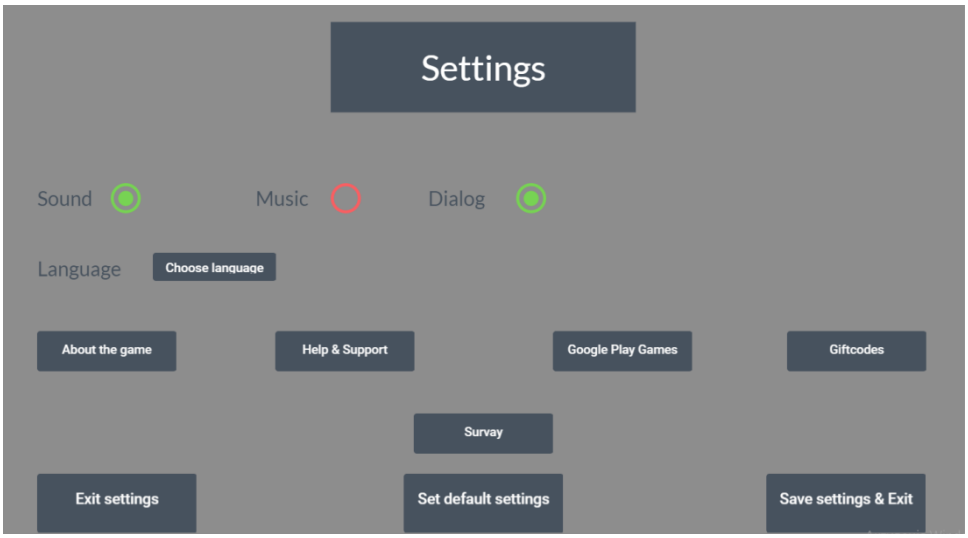

Рисунок 2.3 – Прототип інтерфейсу налаштувань

Перед переходом до “базового” геймплею гравцю необхідно обрати режим гри, але через велику кількість режимів правильним здається не виносити цей вибір у інтерфейс головного меню, а сховати його так само, як і налаштування (рис. 2.4).

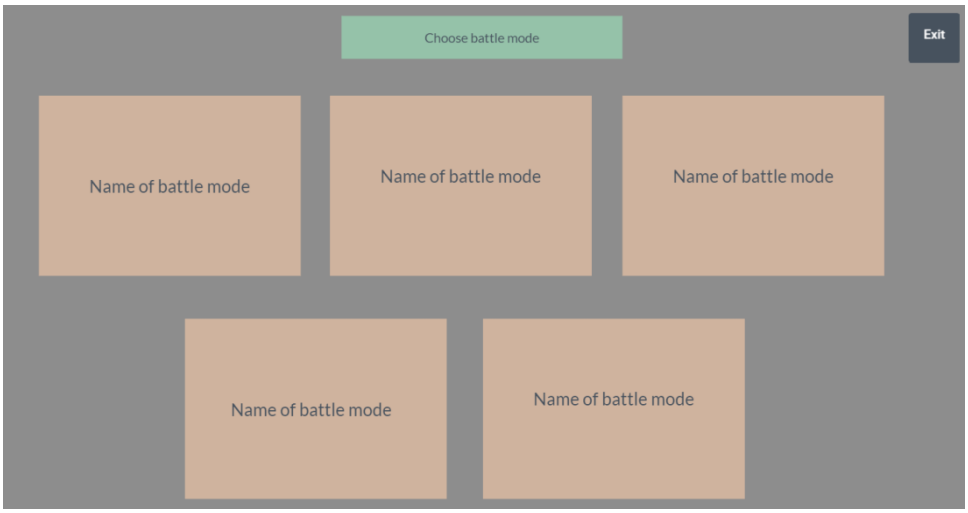

Рисунок 2.4 – Прототип інтерфейсу вибору режимів

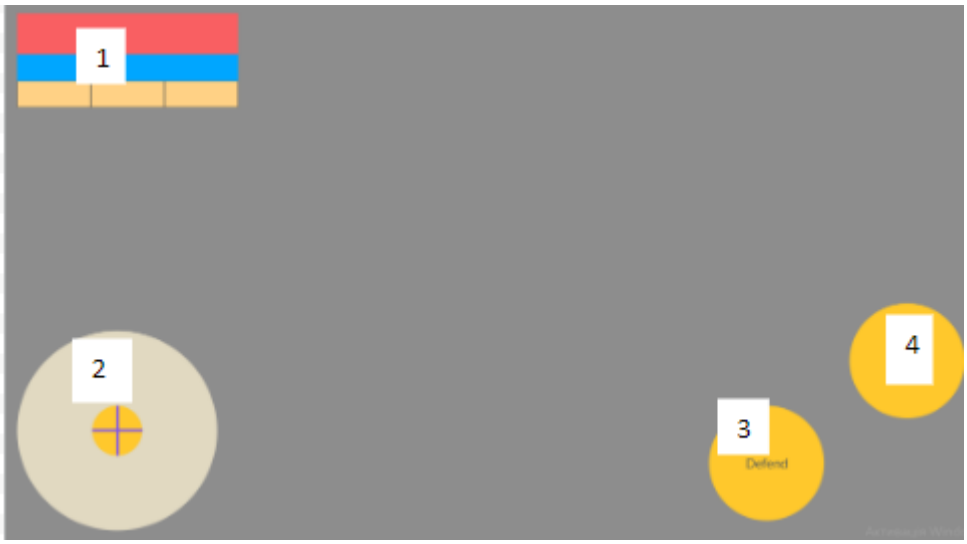

Рисунок 2.5 – Прототип “бойового” інтерфейсу: 1 - Панель стану персонажа; 2 - “Стік” віртуального джойстика; 3 - кнопка захисту; 4 - кнопка атаки

На прототипі “бойового” інтерфейсу або ж інтерфейса “базового” геймплею (рис. 2.5) можна побачити елементи керування гравцем - ліворуч “стік” віртуального джойстику, яким гравець керує напрямком руху, праворуч - кнопки атаки і оборони, а в верхньому лівому куті - панелі стану здоров’я, броні, виснаження персонажа.

На прототипі головного інтерфейсу “Лодоса” (рис. 2.6) в верхньому правому куті гравець бачить “прогрес слави” “Лодоса” - коли шкала прогресу буде заповнена, він зможе змінити будівлю “Лодосу”. Головним же об’єктом інтерфейсу є сам “Лодос” - будівля з якої гравець може керувати спорядженням гладіаторів, частково самим гладіаторами. Зеленими елементами позначено зайняті кімнати, помаранчевими - ті які будуються, темно сірими - вільні, світло-сірий кольором позначено двір. Натискання на будівництво кімнат відкриває для гравця інтерфейс побудови нових кімнат (рис. 2.7) на вільному місці, або дозволяє виконати руйнування/перенесення старих, якщо гравець натисне на існуючу кімнату в режимі будівництва.

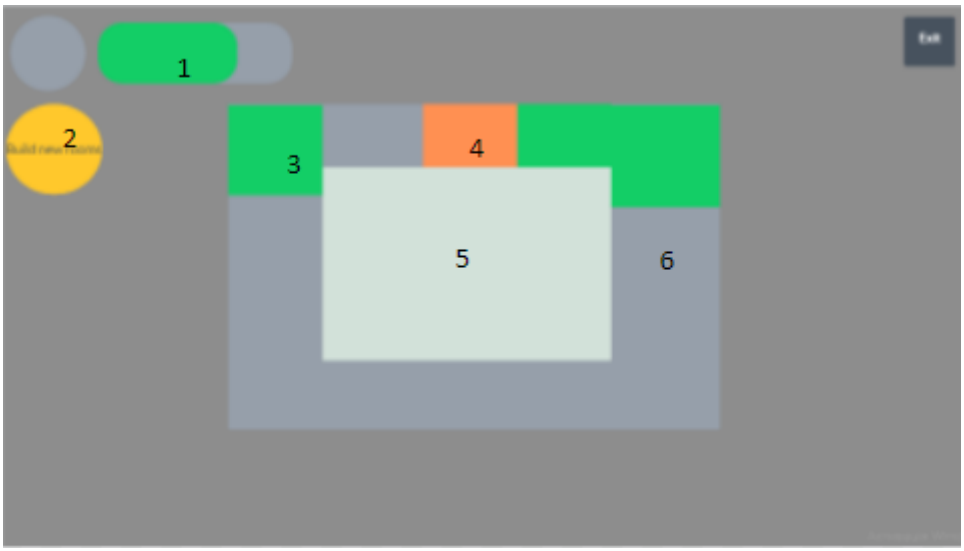

Рисунок 2.6 – Прототип головного інтерфейсу “Лодоса”: 1 - шкала “прогресу слави”; 2 - кнопка переходу в режим будівництва; 3 - зайнята кімната; 4 - кімната що ще будується; 5 - двір “Лодосу”; 6 - вільне місце для кімнат

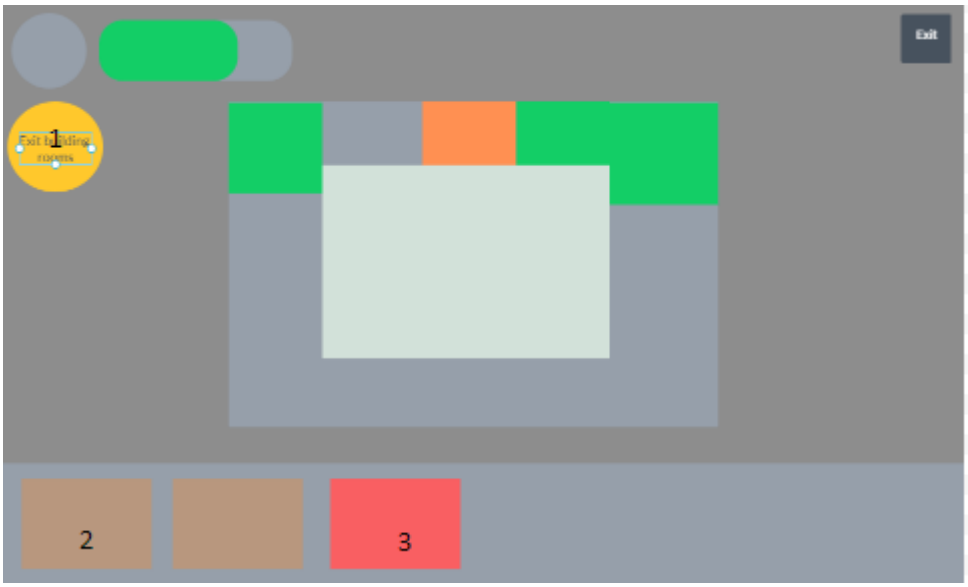

Рисунок 2.7 – Прототип інтерфейсу “Лодоса” в режимі будівництва: 1 - кнопка виходу з режима будівництва; 2 - кімнати які гравець може побудувати; 3 - кімнати які гравець поки що не може побудувати

Одночасно гравець може бачити і кімнати, які йому доступні для побудови, і ті, які він поки що не відкрив - на них зображуються умови для відкриття.

Якщо ж на головному інтерфейсі “Лодосу” гравець натисне на певну кімнату, в нього відкриється випадаюча панель (рис. 2.8). Прототипи інтерфейсів кімнат схожі, відрізнятися вони будуть вже в самому прототипі гри своїм функціоналом.

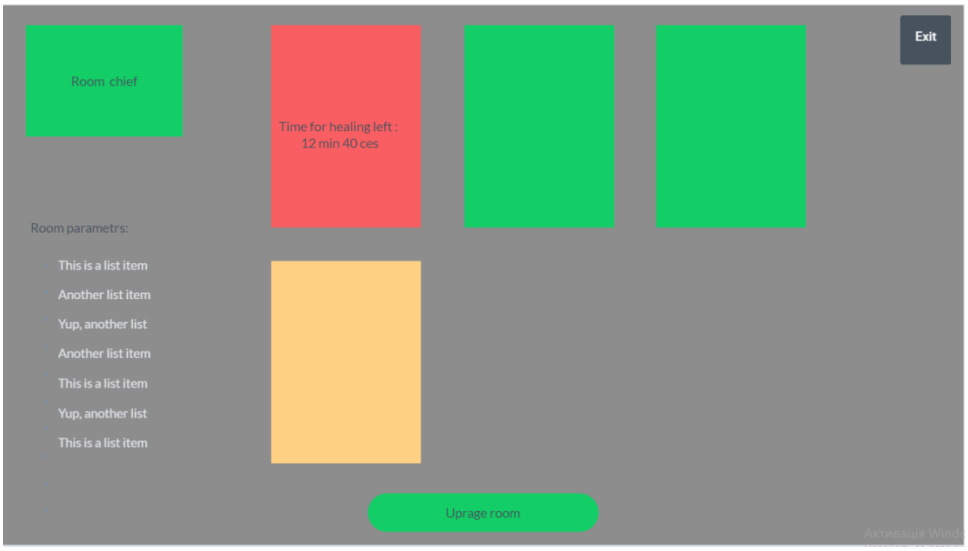

Рисунок 2.8 – Прототип випадаючої панелі з інтерфейсом кімнати

В якості прикладу було приведено інтерфейс медичного кабінету. При натисненні на зайняте місце на фоні інтерфейсу кімнати з’являється інтерфейс ігрового персонажа (рис. 2.9).

Прототипування інтерфейсу користувача є важливою частиною розробки проекту, адже вона дозволяє запобігти помилкам в подальшій розробці або виявити певні недоліки на ранніх етапах.

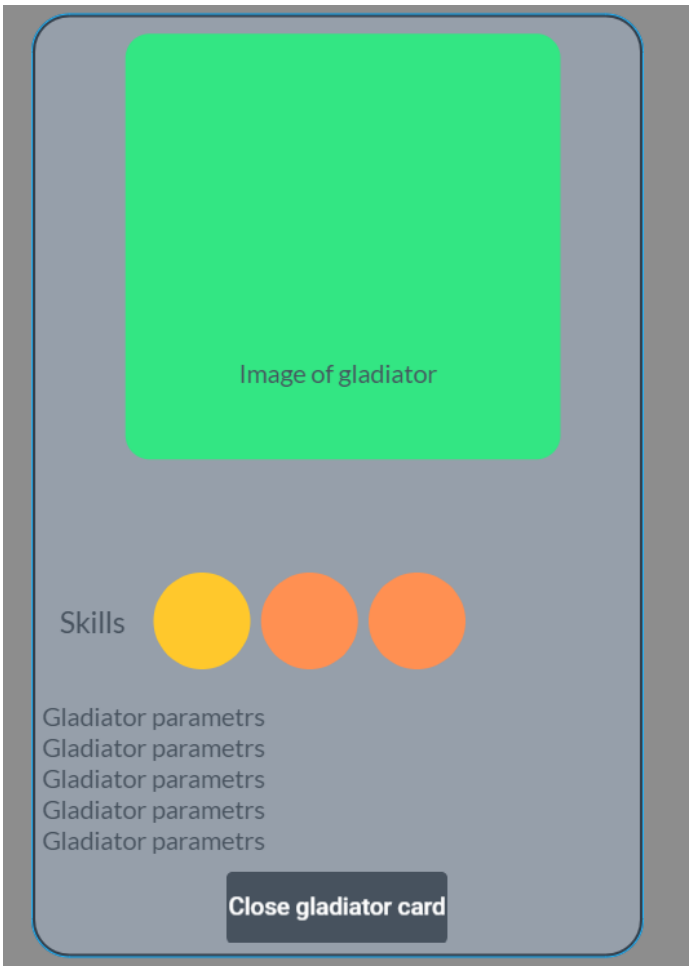

Рисунок 2.9 – Прототип картки ігрового персонажа

2.3.2 Прототип гри

Прототип гри створюється для того, щоб мати можливість оцінити шанс на комерційний успіх проекту. Адже прототип має містити в собі всі основні механіки, які заплановано в проекті. Проте в прототипі дуже часто немає зображень, анімацій, звуків та інших незначних моментів, які обов’язково повинні бути в фінальній версії гри. Часто прототипи складаються з найпростіших елементів - персонажів заміняють кубами, квадратами, кулями.

Прототип самої гри уже необхідно виконувати в ігровому рушії. Перш за все туди переноситься повноцінний прототип інтерфейсу, пишуться скрипти для взаємозв’язку інтерфейсів різних сцен між собою. Після чого додаються прототипи персонажа, яким керуватиме гравець та ворогів цього персонажа (рис. 2.10).

Для створення прототипу було використано безкоштовні моделі з магазину моделей Unity Asset Store. Вони неанімовані, але їх вистачило для того, щоб розробити функціонал гри і перевірити його перші версії.

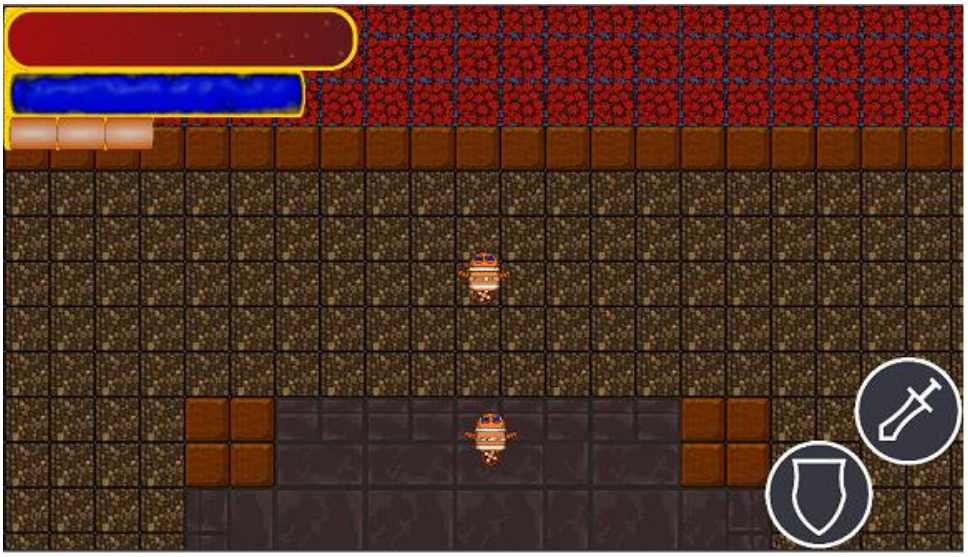

Рисунок 2.10 – Вигляд арени у прототипі гри

РОЗДІЛ 3. РОЗРОБКА ГРИ У ЖАНРІ “АРКАДА”

3.1 Огляд програмних продуктів для розробки ігор

Основним засобом для розробки комп’ютерних ігор є програмний ігровий рушій (software engine). Функціонал для розробки ігор містить необхідні алгоритми для правильного функціонування гри і її розробки. В наш час програмних рушіїв для розробки ігор дуже багато, вони відрізняються за мовами програмування, функціоналом, і вартістю використання. Кожен з них має певні недоліки і переваги в різноманітних напрямах. Багато з них можна модернізувати під спрощення нових функцій, саме тому найбільш поширені рушії мають найбільш широкий функціонал.

Для вибору програмного засобу для реалізації дипломного проекту вибір стояв між Unity 3D і Unreal Engine. Це обумовлено високою функціональністю цих програмних продуктів в першу чергу для 3D-моделювання, але завдяки високій популярності ці програми мають також широкий 2D-функціонал. Програмні засоби спеціально для роботи із 2D (наприклад Game Maker Studio 2, та Clickteam Fusion) мають більш обмежений функціонал. Окрім цього, в вищезазначених програмних засобів досить прості умови комерційного використання у проектах, а також дуже великі бібліотеки ресурсів - як платних, так і безкоштовних. Крім того, обидва рушії мають величезні навчальні бази. Для Unity 3D - існує офіційний ресурс з ресурсами для навчання - Unity Learn [8], а для Unreal Engine - створено базу з офіційними ресурсами для навчання на сайті програмного засобу. Окрім офіційних ресурсів також існують курси від сторонніх компаній, книжки, електронні ресурси, та офіційна документація.

Unreal Engine [21]– ігровий рушій розроблений и підтримуваний компанією Epic Games. Перша гра була створена на цьому рушії в в 1998 році. З тих пір цей

рушій було використано в тисячах інших ігор та проектів. Рушій було створено мовою програмування C++, на ньому можна створювати ігри для таких пристроїв та платформ:

- персональних комп'ютерів на Microsoft Windows, Linux, Mac OS та Mac OS X;
- консолей Xbox, Xbox 360, Xbox One, PlayStation 2, PlayStation 3, PlayStation 4, PSP, PS Vita, Wii, Dreamcast, GameCube та ін.,
- пристроїв на iOS и Android;
- AR,VR.

У рушія є модульна система залежних компонентів, яка використовується для спрощення портування. Він підтримує різноманітні способи рендерингу (Direct3D, OpenGL, Pixomatic), відтворення звуку (EAX, OpenAL, DirectSound3D), засоби голосового відтворення тексту, розпізнавання мови, модулі для роботи з мережею і підтримки різноманітних пристроїв введення даних. 2 березня 2015 року Unreal Engine 4 став безкоштовним. Але розробники ігор, як і раніше, повинні передавати 5 % від прибутку Epic Games за умови, що дохід гри більше $12000 за рік.

Unity 3D – це програмне середовище для розробки 2D і 3D ігор, яке дозволяє створювати застосунки, що працюють на таких операційних системах і платформах як: Windows, OS X, Windows Phone, Android, Apple iOS, Linux, консолях Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One, VR, AR та MotionParallax3D дисплеях. Окрім того, цей програмний засіб надає можливість створювати застосунки для запуску у браузерах за допомогою Unity Web Player, а також за допомогою реалізації технології WebGL. У редактора Unity 3D є Drag&Drop-інтерфейс, який легко налаштовується та складається з різних вікон, завдяки чому можна калібрувати гру прямо в редакторі. Програмний рушій Unity 3D підтримує мову програмування C# та модифікацію JavaScript. Ігрова фізика

розраховується за допомогою фізичних рушіїв PhysX від NVIDIA. Проект в Unity 3D розподілено на сцени (рівні) — окремі файли, в яких зберігаються власні ігрові світи зі своїми наборами об’єктів, сценаріїв і налаштувань. На сценах знаходяться порожні або фізичні об’єкти. В кожної групи об’єктів є набір компонентів з яким взаємодіють скрипти. Програмний засіб Unity 3D є умовно безкоштовним, якщо дохід від гри, зробленої на ньому, не перевищує 100 000$ у рік.

Створювати графічні об’єкти прямо у рушії Unity 3D можливості немає, тому для створення візуальної частини проекту потрібно також обрати спеціалізований програмний продукт. 2D графіка буває векторною і растровою (піксельною), перша з яких складніше в освоєнні, але має свої переваги, друга – простіша в створенні, вимагає меншого рівня навичок, але не завжди забезпечує потрібну якість.

Серед популярних графічних редакторів можна відзначити Adobe Photoshop, CorelIDRAW, Inkscape.

Adobe Photoshop - один із найпопулярніших і найпотужніших графічних редакторів для обробки зображень. В ньому можна працювати з растровою графікою, проте в пакеті Adobe також є продукт Adobe Illustrator, в якому можна працювати з векторною графікою. Існує також такий програмний продукт, як Adobe Animator, в якому присутні якісні засоби для створення анімацій зображень, які також можна імпортовувати в Unity 3D. Для програмних продуктів Adobe існує багато навчальних матеріалів, але головним недоліком для початкового розробника є їх висока вартість.

CorelIDRAW - векторний графічний редактор, створений однойменною компанією. В пакеті Corel також існують інші програмні продукти, один з яких дозволяє працювати з растровою графікою. Так само, як і попередній пакет, цей продукт також має високу вартість.

Inkscape - векторний графічний редактор, має широкий набір функцій. Потребує ознайомлення, проте функціонал майже аналогічний схожим продуктам. Розповсюджується за ліцензією GNU General Public License, що дозволяє використовувати продукт для створення зображень безоплатно, якщо вони будуть розповсюджуватися за такою ж ліцензією.

3.2 Розробка сцени головного меню

Розробку гри здійснено в програмному засобі Unity 3D із використанням середовища розробки Visual Studio та мови C#. Unity 3D було обрано через те, що для нього існує більше навчальних матеріалів з різним рівнем пояснення матеріалу. Крім того, Unity 3D також має підтримку розробки мовою C# (в Unreal Engine розробляти треба мовою C++), що є перевагою через простіше засвоєння C#.

Гра складається зі сцени меню, сцени лодосу, і сцен, які потрібно виділити в окрему категорію - сцени арен. Вони відокремлені у зв'язку з тим, що кількість цих сцен в процесі розробки і оновлення гри буде збільшуватись, а також через те, що вони всі є однотипними і мають схожу структуру.

Сцена меню - перша сцена, на яку потрапляє гравець після запуску гри. Вона структурно складається з контролера сцени і 3 груп елементів - групи, яка містить усі елементи меню (усі кнопки, обраний гравцем персонаж, елементи, що візуалізують прогрес гравця, ігрова валюта, яка є в наявності у гравця та ін.), група панелі налаштувань та всіх її елементів, група панелі вибору режиму гри та всіх її елементів. Всі ці елементи можна побачити в частині інтерфейсу Unity 3D, яка називається ієрархією та відображає всі об'єкти, які знаходяться на сцені (рис. 3.1). Окрім вікна з ієрархіями і всіх об'єктів на сцені, які можна побачити в

Unity 3D, інтерфейс Unity 3D складається зі сцени (на ній доступне переміщення, обертання об’єктів сцени), екрану (відображення того вигляду екрану, який побачить гравець), інспектора (надає можливість детального редагування об’єкту), консоль (відображає критичні помилки і попередження, які існують в проекті), провідник (відображає всі об’єкти, які є в проекті, з файлами і папками), аніматор (вікно для створення моделей анімації з готових анімаційних блоків) та інші. Інтерфейс Unity 3D (рис. 3.2) є модульним, тому його можна налаштувати так, як зручно розробнику, всі вікна можуть змінювати своє розташування

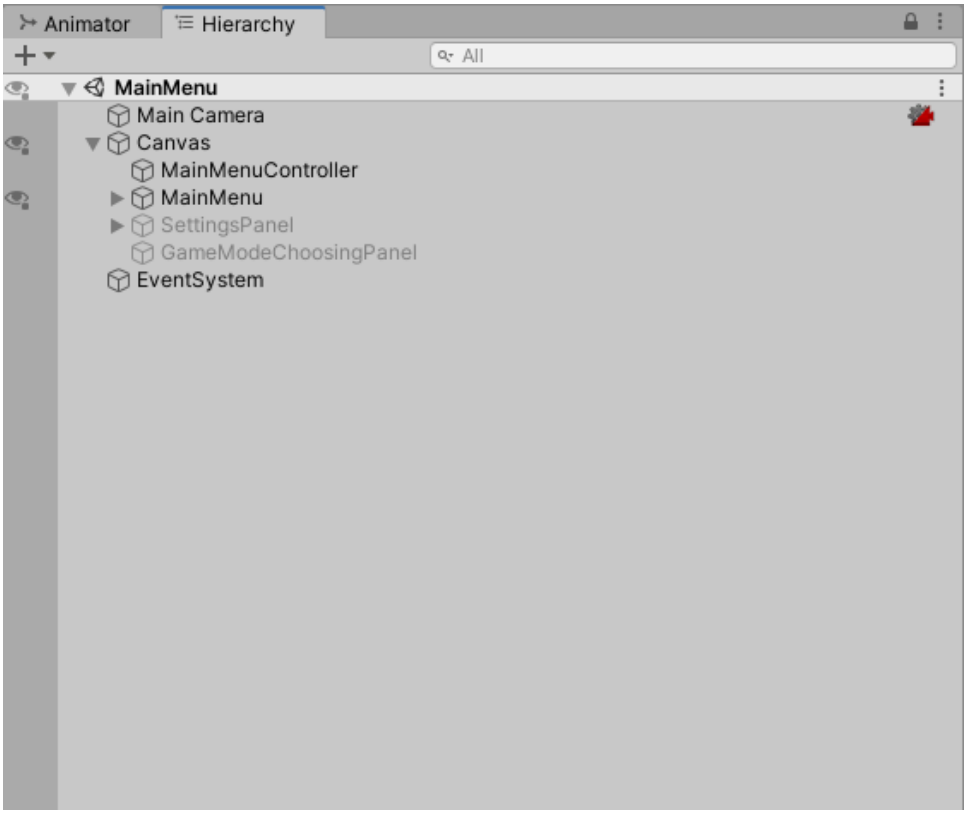

Рисунок 3.1 – Вікно ієрархії в Unity 3D

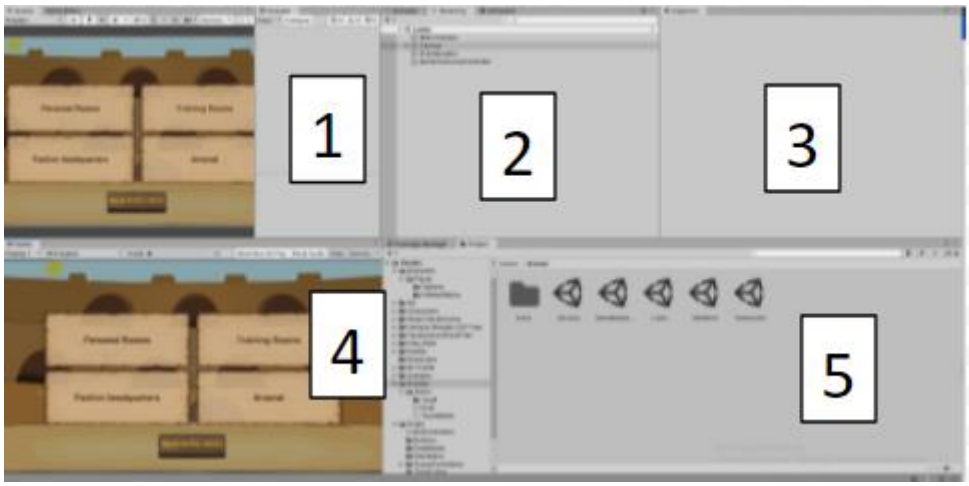

Рисунок 3.2 – Інтерфейс Unity 3D: 1 - Сцена; 2 - Ієрархія; 3 - ”Інспектор”; 4 - Екран; 5 - “Провідник”

Одночасно на екрані гравця відображається лише єдина група елементів - контроль усіх процесів (відкриття інших панелей, перехід на іншу сцену) здійснює контролер головного меню, для нього було написано спеціальний скрипт. Також спеціальний скрипт, який керує анімацією, було створено для моделі головного персонажа (додаток А).

Власні графічні елементи для гри було створено в графічному редакторі Inkscape, його було обрано через тип ліцензії, за яким він розповсюджується, що дозволяє використання редактора в комерційних проектах. Для головного меню в редакторі було створено фон і панелі вибору режиму гри, зображення шкали прогресу, деякі зображення кнопок і зображення ігрових валют. Модель головного персонажа з його анімаціями, іконки деяких кнопок, було взято з Unity Asset Store.

Рисунок 3.3 – Головна сцена гри після компонування всіх готових елементів

3.3 Розробка сцени лодосу

Сцена лодосу містить в собі великий геймплей, який менш динамічний у порівнянні з геймплеєм бою на аренах. Для його функціонування було написано скрипт контролю сцени, який керує відображенням таких елементів, як панель непобудованих кімнат - вона повинна з’являтися тільки тоді, коли натискається кнопка будівництва, і при її натисканні при відчиненій панелі - панель повинна закритися. Окрім панелі непобудованих кімнат, скрипт контролю сцени керує відображенням карток з інформацією про гладіаторів.

Після скрипта контролю сцени лодосу найважливішим компонентом сцени є будівля лодосу - саме через неї гравець може здійснювати відповідне керування лодосом, наймати гладіаторів, вибирати того гладіатора з яким гравець піде у бій, лікувати поранених гладіаторів. Більшість об’єкту будівлі є функціональним - при

натисненні вже побудованих кімнат відкриваються відповідні кімнати та меню. Для цього кожна з кімнат має компонент Button, в якому при натисканні виконується одна з функцій в скрипті контролера сцени. Кімнати-спальні після відкриття їх із застосуванням меню відображають отриманий прогрес гравця, а саме – скільки місць в кімнаті ким зайнято, скільки вільно, відповідний скрипт автоматично отримує дані з файлу збережених даних. Всі карти гравців, які відображають, ким зайняте місце, також мають компонент Button, по натисканню на який відкривається вікно з детальними даними про гладіатора. Окрім кімнати-спальні також на картку ігрового персонажа можна натиснути і в кімнаті-лікарні та тренувальному залі. На сцені лодосу дуже багато різноманітних елементів - кнопок, зображень, панелей, проте всі вони схожі за функціоналом, тому для їх роботи підходять ті самі скрипти, в яких відрізняються лише посилання на різні об’єкти.

3.4 Розробка шаблону сцен арен

Сцени арен відрізняються одна від одної лише контролером - для кожного з режимів гри було розроблено окремий контролер, і штучними ворогами, а в певних режимах це ще і штучні союзники, для них створено декілька шаблонів для кожного з режимів гри. Останній важливий компонент рівня арени - сама арена та її компоненти, їх також розроблено декілька, але одна і та сама арена може бути використана в різних режимах з відповідними контролерами і штучними ворогами створеними для обраного режиму.

Контролер арени це порожній об’єкт [9] (в Unity 3D об’єкт, в якого немає форми на сцені, найчастіше використовується як контейнер), до якого було

прикріплено скрипт, який керує ареною. Скрипт контролера арени виконує такі функції:

- керує відображенням поточного інтерфейсу рівня арени;
- керує відображенням результатів гри на рівні;
- завершує гру при досягненні передбачених умов програшу або виграшу;
- зберігає досягнення і отримані ігрові предмети в базу даних гри.

Штучні вороги – це шаблон ворога для гравця, в кожному режимі він свій, тому що в скрипті ворога передбачено, яким саме ворогам він може наносити пошкодження (гравцю, союзникам гравця або ж всім ігровим персонажам арени, окрім себе самого), а в кожному з режимів вони відрізняються. Шаблон ворога складається з вищезгаданого скрипта, який контролює параметри, вигляд (які зображення використані для “частин тіла”, спорядження, зброї), рух, атаку і захист ворога, колайдеру, прикріпленого об’єкту з аніматором (керує умовами програвання анімацій і переходами між ними) і прикріплених об’єктів із зображеннями і координатами розташування різних “частин тіла” ворога. Шаблон союзника також схожий за складом на шаблон штучного ворога (рис. 3.4).

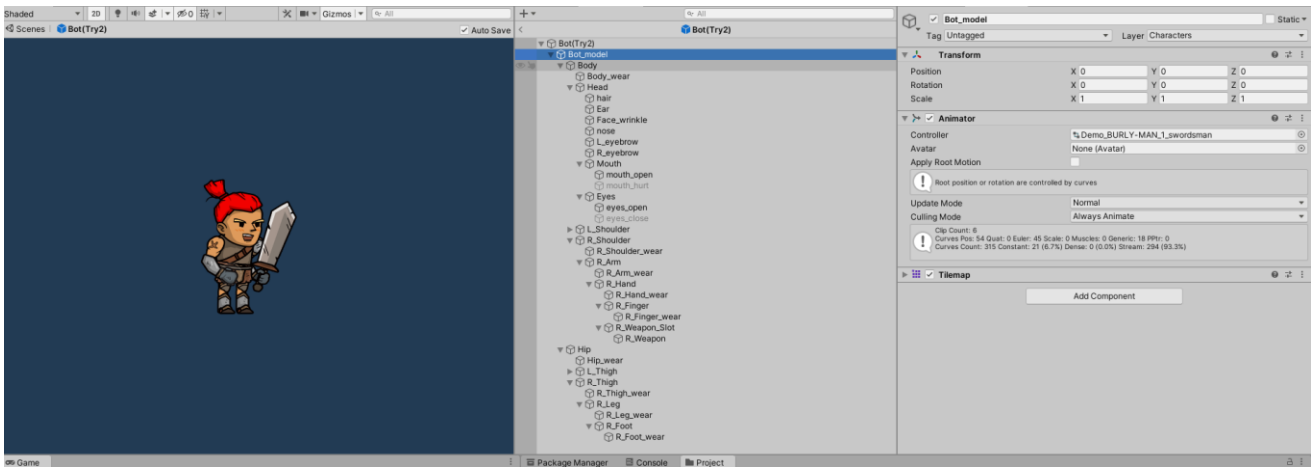

Рисунок 3.4 – Відображення елементів шаблону ворога на сцені в ієрархії

Персонаж гравця, якого він обрав і спорядив для гри в інших сценах (сцені лодосу), складається з таких основних компонентів:

- тверде тіло (Rigidbody [11]) - об’єкт який дозволяє взаємодію із фізикою;
- скрипт-контролер персонажа, який завантажує з бази даних параметри обраного гравцем гладіатора, отримує і використовує для руху та бою вхідні параметри з контролера переміщенням, кнопок атаки і захисту, контролює, яких ігрових персонажів гравець може атакувати, контролює анімації;
- скрипт для взаємодії з інтерфейсом, для відображення рівня здоров’я та наснаги персонажа на інтерфейсі;
- колайдер [10];
- аніматор, який контролює умови ввімкнення анімацій і переходи між ними;
- “порожній” об’єкт з прикріпленими зображеннями “частин тіла” і їх розташуванням відносно до основи персонажа.

Рисунок 3.5 – “Бойовий” інтерфейс

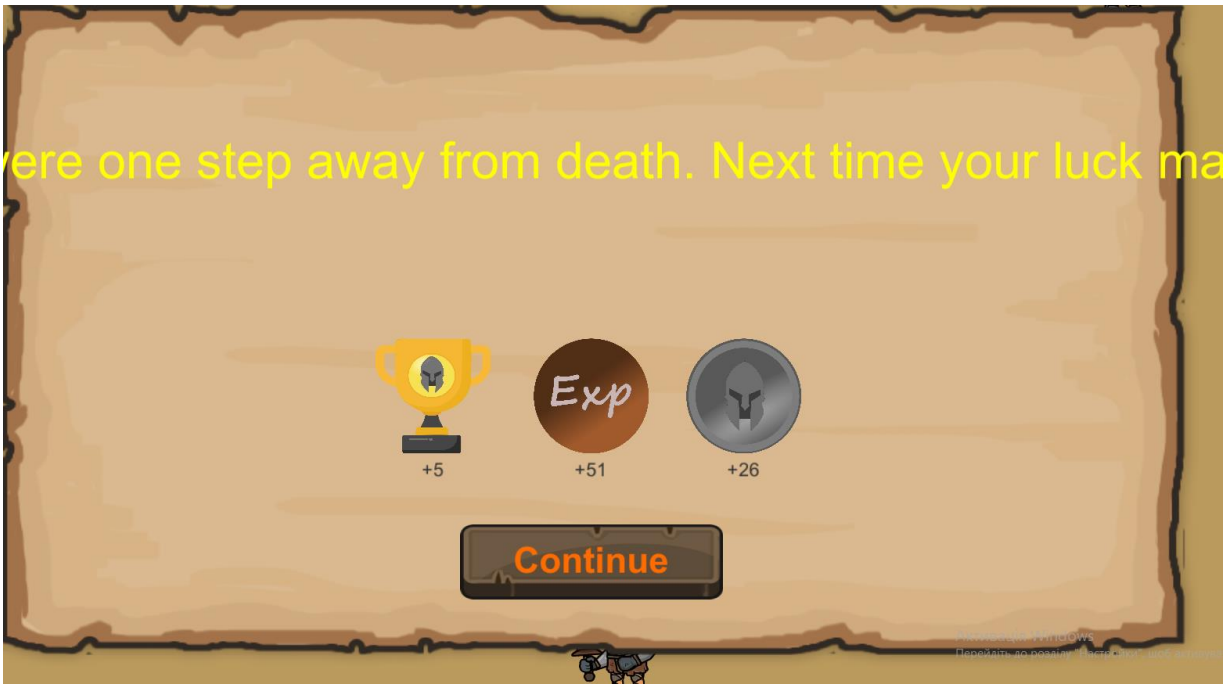

Рисунок 3.6 – Інтерфейс результату

У сцен арени передбачено два інтерфейси - один з інтерфейсів можна назвати “бойовим” інтерфейсом (рис. 3.5). Він складається зі шкали здоров’я, шкали броні, шкали виснаження в лівому верхньому куті, віртуального геймпаду керування рухом в нижньому лівому куті, двома кнопками - атаки, і захисту в нижньому правому куті. Другий інтерфейс - інтерфейс результату (рис. 3.6), він викликається контролером рівня, коли виконуються умови завершення гри. На цьому інтерфейсі відображається результат бою - поразка або виграш, відображаються елементи, які гравець виграв в бою та кнопка виходу на головне меню.

Усі арени складаються з “плиток” (tiles) (рис. 3.7) однакового розміру. Ними “викладено” вигляд арени, деякі плитки мають власні колайдери - так створюється межа, за яку не можуть заходити персонажі і перепони, якими персонажі не зможуть рухатися. Саме таким чином відрізняються карти арен: зовнішнім виглядом, перепонами, розмірами і кордонами.

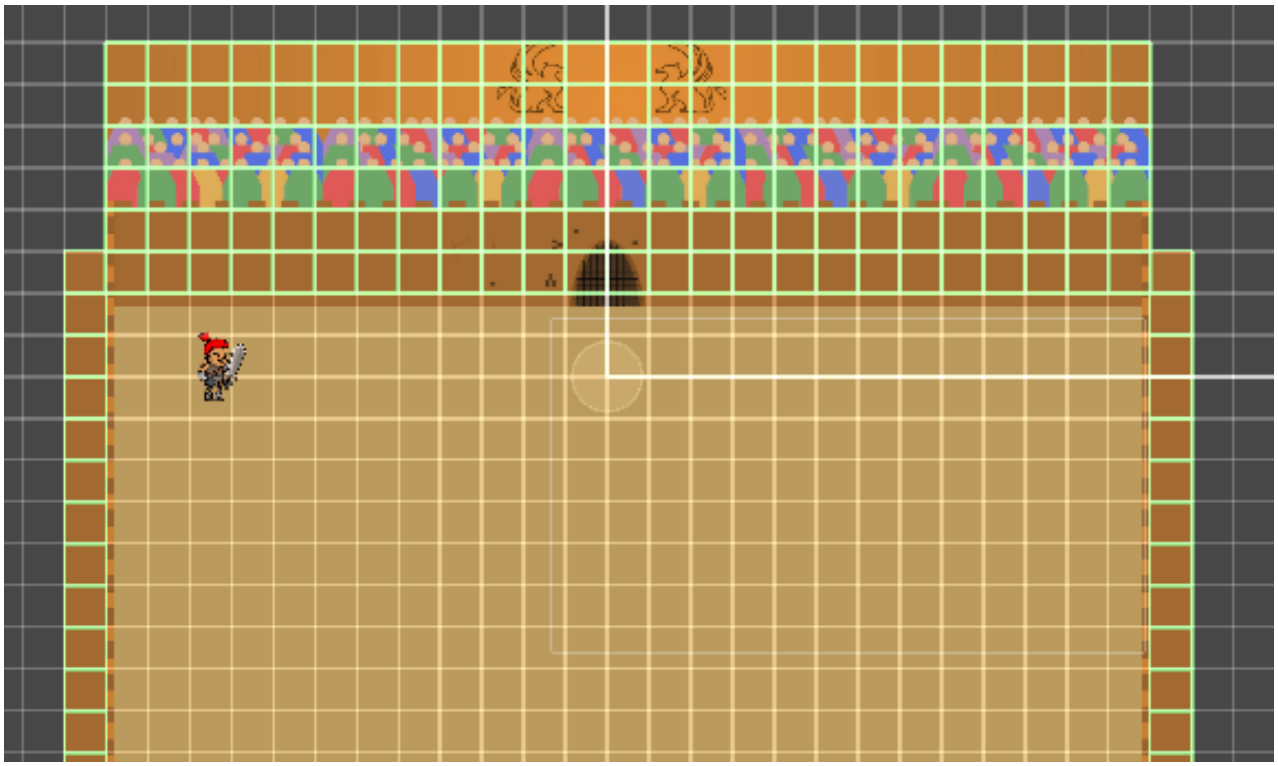

Рисунок 3.7 – Вигляд арени із ввімкненою сіткою, для видимості окремих плиток

3.5 Збереження ігрових даних

До ігрових даних можна віднести ігрову валюту, поранення, отримані під час бою, а також інформацію про купівлю, найм, тренування та спорядження гладіаторів. Всі ці дані повинні зберігатися для того, щоб коли гравець вийде з гри і знову її запустить або перейде зі сцени на сцену його ігровий прогрес було збережено. Для збереження прогресу було створено багато технологій - найпопулярніші з них:

- PlayerPrefs;
- використання Google Play Games;
- серіалізація з збереженням в хмарному сховищі;
- вбудована серіалізація із використанням JSON / XML-файлів;
- сторонні ресурсні рішення.

Метод, який використовується для збереження прогресу гравця, називається серіалізацією. Серіалізація [12] - це перетворення обраних даних в формат, який Unity 3D може зберегти і відтворити в подальшому

PlayerPrefs [13] - це вбудований у Unity 3D метод, який призначений для збереження невеликих об’ємів даних, наприклад, внутрішньоігрових налаштувань гравця (гучність звуку, обрана мова гри і т.п.), збереження та використання налаштувань при переходах між сценами гри та вході/виході з гри. Серед недоліків цього методу збереження даних можна відмітити можливість зміни даних користувачем, неможливість збереження прогресу гравця для використання на декількох пристроях, тобто збережені дані зникнуть також і при видаленні гри. Через це таким методом не рекомендується зберігати прогрес гравця, кількість ігрової валюти, щоб запобігти нечесному геймплею та втраті ігрового прогресу.

Метод збереження ігрового прогресу із використанням Google Play Games [16] - широко поширений метод, який використовується тільки в проектах для мобільних пристроїв на операційній системі Android. Цей метод дозволяє зберегти будь-які необхідні дані. Також завдяки застосуванню цього метода гравець може переглянути певні елементи ігрового прогресу в програмі Google Play Games. Вона дозволяє реалізувати не тільки збереження прогресу гравця, а також такі досягнення, дошки лідерів, ігрових друзів, вхід в власний акаунт. Серед недоліків такого методу можна зазначити невеликий виділений обсяг для даних на хмарному диску Google (3 мегабайти для даних і 800 кілобайт для зображень). Серед переваг - можливість гри зі своїм ігровим прогресом на різних пристроях, неможливість зміни файлів гравцем (що не дає гравцям вести нечесну гру), можливість збереження прогресу навіть без з’єднання з інтернетом з подальшим завантаженням на хмарне сховище Google, прогресу при відновленні доступу до інтернету.

Збереження ігрового прогресу із використанням збереження в хмарному сховищі (в якості прикладу було розглянуто популярну систему Firebase) в платній версії позбавлене основного недоліку попереднього методу - обмеженого обсягу. В безкоштовній версії існують певні обмеження на кількість користувачів, загальний обсяг та інше [18]. Перевагами методу є великий функціонал налаштування збереження даних, доступ до прогресу з різних пристроїв і платформ. Недоліком є складність технічного оволодіння цим методом збереження прогресу.

В Unity 3D існує вбудована серіалізація, яка дозволяє зберігати ігровий прогрес в файлах для локального зберігання, наприклад, скриптах. Але гравець не зможе продовжити гру на іншому пристрої. Серіалізація із використанням JSON / XML файлів - це збереження інформації в файли обраного формату (json, або XML) та відповідне завантажування звідти за необхідності. За методами використання цей метод є ідентичним до вбудованої серіалізації Unity 3D. Проте json та XML файли можна зашифрувати, що не дозволить недобросовісним користувачам вести гру не за правилами.

Сторонні ресурсні рішення - це готові ресурси, які дозволяють зберегти прогрес гравця. Всі вони відрізняються між собою якістю, ціною (в Unity Asset Store є безкоштовні та небезкоштовні готові рішення). Готові ресурси відрізняються складністю використання, складністю навчання, наявністю навчальних матеріалів, функціоналом, який надають ресурси (підключення до хмарних сховищ, можливість шифрування збережених даних, швидкість процесу, кількість платформ, на яких можливе збереження, ціна самого ресурсу). В кожного з ресурсів є свої переваги і недоліки (в залежності від необхідного функціоналу).

Для збереження прогресу гравця в цьому проекті було використано декілька методів. Серед яких - збереження з методом PlayerPrefs, який зберігає

налаштування гравця, всі вони знаходяться в окремій панелі налаштувань на сцені головного меню. Збереження відбувається, коли гравець натискає кнопку “Зберегти”, для якої в скрипті створено спеціальний метод (рис. 3.8).

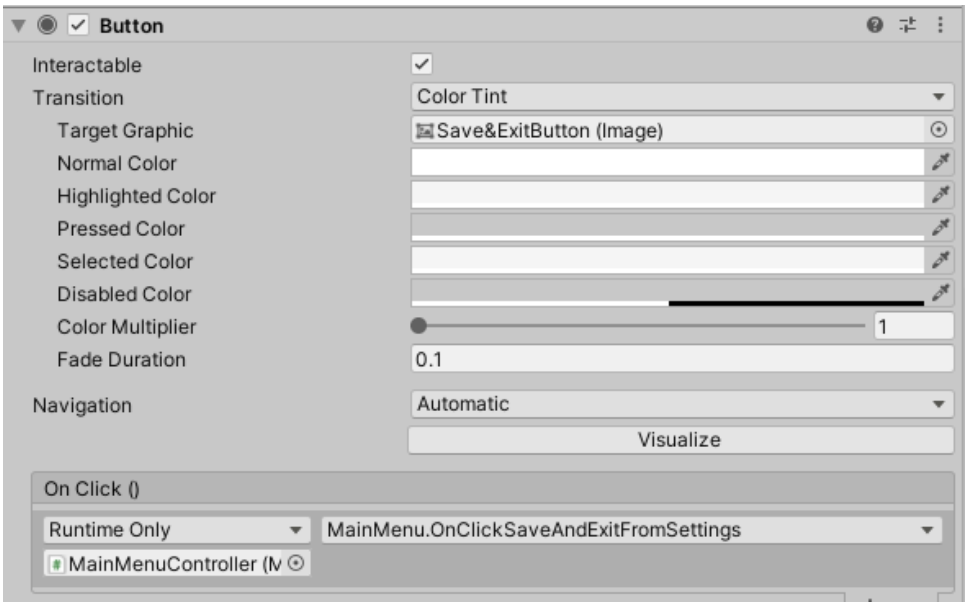

Рисунок 3.8 – Компонент Button приєднаний до кнопки збереження і функція, яка виконується при натисканні на кнопку

Наступним використаним методом було збереження в json-файл (рис. 3.9), його було обрано через простоту навчання для подальшого використання і можливість подальшої модернізації методу для “хмарних” збережень.


```
{"totalExpirience":242,"currentPlayerLevel":1,"trophies":14,"c
urrentFreeUpPoints":0,"currentSilverCoins":85,"currentGoldCoin
s":0,"currentGems":0,"currentCoinsForPlay":0,"playerTotalStren
ght":1,"PlayerAttackDamage":{"x":0.0,"y":0.0},"playerSpeed":3.
0,"playerMaxHealth":3,"playerHealthRegeneration":0.0,"playerMa
xArmor":0,"playerArmorRecovering":0.0,"dmgAbsorptionPercentage
":0,"playerMaxStamina":1,"playerStaminaRecovering":0.300000011
92092898,"playerStaminaUsing":0.30000001192092898,"playerViewR
adius":10.0,"playerAttackRadius":0.0,"playerStrenghtfromLvl":0
,"playerSpeedfromLvl":0,"playerHealthfromLvl":0,"playerStamina
fromLvl":0,"helmet":0,"weapon":0,"brigandine":0,"boots":0}
```


Рисунок 3.9 – Вигляд файлу json із даними до шифрування

В файлі загального прогресу зберігаються всі внутрішньо-ігрові валюти (срібло, золото, квитки звичайної і імператорської арен), рівень гравця, загальна кількість накопиченого досвіду, загальна кількість програних і виграних боїв на арені, рівень слави, популярність лодосу, загальна кількість гладіаторів, найпопулярніший гладіатор і його популярність.

В файлі “будівля лодосу” зберігаються побудовані кімнати та їх наповнення, їх координати, керівники кімнат, їх параметри (рівень персонажу, ім’я, бонус, який він додає та інше) і особливі навички, загальні параметри кімнат (бонус до швидкості функціонування, кількість місць, та інше).

В файлі “лодос” зберігаються усі гладіатори, їх параметри і навички, збережене на кожному з них спорядження.

Збереження усіх цих даних дозволить зберігати і відображати весь прогрес гравця. Крім прогресу гравця, який необхідно зберігати в іграх, також існують внутрішні об’єкти, параметри яких не генеруються під час створення об’єкту (наприклад, як генеруються параметри гладіатора під час його появи на ринку, або керівника кімнати), а є постійними. До таких об’єктів можна віднести, наприклад, деяке спорядження гладіатора. Параметри цього спорядження є сталими і зберігаються в спеціальних ScriptableObject - об’єктах Unity 3D, які створені для збереження великих об’ємів даних. Також зручністю ScriptableObject є те, що скрипт, в якому його описано, є одночасно макетом для інших, тому після його створення можна легко створити об’єкти за макетом, після чого заповнити їх даними. Було створено 4 класи спорядження (шоломи, броня, взуття, зброя).

ВИСНОВКИ

В результаті виконання дипломного проекту було розроблено гру в жанрі «аркада» на платформі Unity 3D.

В ході виконання дипломного проекту було виконано детальний аналіз жанру “Аркада”. Був виконаний аналіз ринку мобільних ігор і визначення найбільш популярних ігор жанру, що дозволило виявити спільні недоліки жанру і спробувати позбутися їх в своєму проекті.

Після детального аналізу жанру було проведення проектування ігрового застосунку, створено дизайн-документ для гри і розроблено діаграму використання і перші прототипи, після цього було додано функціонал до прототипу, і проведено перші тестування. Після вирішення найбільш критичних помилок було проведено аналіз стану графічних елементів, які вже існують, та відібрано елементи зі схожим стилем. Було розроблено та імпортовано в гру власні графічні елементи.

В результаті виконання дипломного проекту було розробено цікавий ігровий застосунок, для якого було створено десятки графічних елементів і написано більше п’ятнадцяти скриптів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Коваленко К. Вчити по-новому: гру Minecraft почали використовувати у школі. URL: https://vn.20minut.ua/Podii/vchiti-po-novomu-gru-minecraft-pochali-vikoristovuvati-u-shkoli-10562386.html (дата звернення: 06.04.2022).
2. Contributors to Wikimedia projects. Golden age of arcade video games - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Golden_age_of_arcade_video_games (дата звернення: 10.04.2022).
3. Contributors to Wikimedia projects. Pinball - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Pinball#Late_18th_century:_Spring_launcher_invented (дата звернення: 12.04.2022).
4. Учасники проектів Вікімедіа. Сетинг – вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Сетинг (дата звернення: 31.05.2022).
5. A look at the global gaming industry in 2021. VCBay News. URL: https://www.vcbay.news/2021/04/01/the-global-gaming-industry-in-2021/ (дата звернення: 15.04.2022).
6. Andersen J. Where Games Go To Sleep: The Game Preservation Crisis, Part 1. Game Developer. URL: https://www.gamedeveloper.com/business/ where-games-go-to-sleep-the-game-preservation-crisis-part-1 (дата звернення: 28.04.2022).
7. Hosking J. Unity in action: multiplatform game development in C# with unity 5. Manning Publications, 2015. 352 с (дата звернення: 31.05.2022).
8. Learn game development w/ Unity | Courses & tutorials in game design, VR, AR, & Real-time 3D | Unity Learn. Unity Learn. URL: https://learn.unity.com (дата звернення: 31.05.2022).

9. What is the purpose of the Empty GameObject?. GameDev.tv. URL: https://community.gamedev.tv/t/what-is-the-purpose-of-the-empty-gameobject/33846 (дата звернення: 31.05.2022).
10. Unity - manual: colliders. Unity - Manual: Unity User Manual 2021.3 (LTS). URL: https://docs.unity3d.com/2018.4/Documentation/Manual/Colliders Overview.html (дата звернення: 31.05.2022).
11. Unity - Manual: Rigidbody component reference. Unity - Manual: Unity User Manual 2021.3 (LTS). URL: https://docs.unity3d.com/2019.4/Documentation/ Manual/class-Rigidbody.html (дата звернення: 31.05.2022).
12. Unity - manual: script serialization. Unity - Manual: Unity User Manual 2021.3 (LTS). URL: https://docs.unity3d.com/Manual/script-Serialization.html (дата звернення: 31.05.2022).
13. Unity - Scripting API: PlayerPrefs. Unity - Manual: Unity User Manual 2021.3 (LTS). URL: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html (дата звернення: 1.06.2022).
14. Tank V. Introduction to unity serialization and game data. Game Developer. URL: https://www.gamedeveloper.com/business/introduction-to-unity-serialization-and-game-data (дата звернення: 1.06.2022).
15. John. How to use Player Prefs in Unity - Game Dev Beginner. Game Dev Beginner. URL: https://gamedevbeginner.com/how-to-use-player-prefs-in-unity/ (дата звернення: 1.06.2022).
16. Saved Games | Play Games Services | Google Developers. Google Developers. URL: https://developers.google.com/games/services/common/concepts/ savedgames (дата звернення: 1.06.2022).
17. Martin P. Using Firebase Cloud Storage with Unity to share user generated content. Medium. URL: https://medium.com/firebase-developers/ using-firebase-

cloud-storage-with-unity-to-share-user-generated-content-25136cd3771d (дата звернення: 1.06.2022).
18. Firebase price-list. Firebase. URL: https://firebase.google.com/pricing (дата звернення: 1.06.2022).
19. Lesson 19.4 – Saving and loading the player information – ScottLilly.com. ScottLilly.com – C# programming tips, tutorials, and techniques. URL: https://scottlilly.com/learn-c-by-building-a-simple-rpg-index/lesson-19-4-saving-and-loading-the-player-information/ (дата звернення: 1.06.2022).
20. Unity - Manual: ScriptableObject. Unity - Manual: Unity User Manual 2021.3 (LTS). URL: https://docs.unity3d.com/Manual/class-ScriptableObject.html (дата звернення: 1.06.2022).
21. Учасники проектів Вікімедіа. Unreal Engine – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Unreal_Engine (дата звернення: 1.06.2022).
22. Учасники проектів Вікімедіа. Ігровий процес – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Ігровий_процес (дата звернення: 1.06.2022).
23. Учасники проектів Вікімедіа. Прототип – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Прототип (дата звернення: 1.06.2022).
24. Учасники проектів Вікімедіа. Сеттинг – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Сетинг (дата звернення: 1.06.2022).
25. Учасники проектів Вікімедіа. Жанр – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Жанр (дата звернення: 1.06.2022).
26. 2D Game Kit - Unity Learn. Unity Learn. URL: https://learn.unity.com/project/2d-game-kit (дата звернення: 1.06.2022).
27. Animator Controllers - Unity Learn. Unity Learn. URL: https://learn.unity.com/tutorial/animator-controllers-2019-3 (дата звернення: 1.06.2022).
28. Ruby's Adventure: 2D Beginner - Unity Learn. Unity Learn. URL: https://learn.unity.com/project/ruby-s-2d-rpg (дата звернення: 1.06.2022).

ДОДАТКИ

Додаток А

Скрипт контролю сцени головного меню

```
using System.Collections;
using static System.Collections.IEnumerable;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using System;
using System.IO;
public class MainMenu : MonoBehaviour
{
   private SaveSettings sv = new SaveSettings();
   private SavePlayerProgress sPP = new SavePlayerProgress();
   private PlayerLvlRequirements PLR = new PlayerLvlRequirements();
   private string path1,path2,path3;
   public GameObject button1Text, button2Text, button3Text;
   public Image button1, button2, button3;
   public Sprite On, Off;
   public GameObject curentLevelText,currentTrophiesNumber, ExpText;
   public Image LvlProgressBar;
   public GameObject SettingsScreen,MainMenuScreen, MainHero;
   public void Start()
   {
      GetPath();
      GetCurentLevel();
      MadeExpBar();
   }
   private void Update()
   {
```

```
    ButtonController();
}
public void QuitGame()
{
    Debug.Log("Quit");
    Application.Quit();
}
public void GoToGameModeSelector()
{
    SceneManager.LoadScene("GameModeSelector");
}
public void GoToBarracks()
{
    SceneManager.LoadScene("Barracks");
}
public void OnClickSettings()
{
    if (!SettingsScreen.activeSelf)
    {
        SettingsScreen.SetActive(true);
        MainMenuScreen.SetActive(false);
        MainHero.SetActive(false);
    }
}
public void OnClickExitFromSettings()
{
    MainMenuScreen.SetActive(true);
    MainHero.SetActive(true);
    SettingsScreen.SetActive(false);
}
public void OnClickSaveAndExitFromSettings()
```

```
{
    File.WriteAllText(path1, JsonUtility.ToJson(sv));
    MainMenuScreen.SetActive(true);
    MainHero.SetActive(true);
    SettingsScreen.SetActive(false);
}
public void OnClickSoundOn()
{
    if (sv.sound == true)
    {
        sv.sound = false;
    }
    else if(sv.sound == false)
    {
        sv.sound = true;
    }
}
public void OnClickMusicOn()
{
    if (sv.music == true)
    {
        sv.music = false;
    }
    else if (sv.music == false)
    {
        sv.music = true;
    }
}
public void OnClickDialogOn()
{
    if (sv.dialog == true)
    {
```

```
        sv.dialog = false;
    }
    else if (sv.dialog == false)
    {
        sv.dialog = true;
    }
}
public void OnClickSetDefaultSettings()
{
    sv.sound = true;
    sv.music = true;
    sv.dialog = true;
    sv.language = "english";
}
public void ButtonController()
{
    if (sv.sound == true)
    {
        button1Text.GetComponent<Text>().text = "ON";
        button1.sprite = On;
    }
    else if (sv.sound == false)
    {
        button1Text.GetComponent<Text>().text = "OFF";
        button1.sprite = Off;
    }
    if (sv.music == true)
    {
        button2Text.GetComponent<Text>().text = "ON";
        button2.sprite = On;
    }
    else if (sv.music == false)
```

```
    {
        button2Text.GetComponent<Text>().text = "OFF";
        button2.sprite = Off;
    }
    if (sv.dialog == true)
    {
        button3Text.GetComponent<Text>().text = "ON";
        button3.sprite = On;
    }
    else if (sv.dialog == false)
    {
        button3Text.GetComponent<Text>().text = "OFF";
        button3.sprite = Off;
    }
}
public void GetCurentLevel()
{
    curentLevelText.GetComponent<Text>().text = sPP.currentPlayerLevel + " Level";
    currentTrophiesNumber.GetComponent<Text>().text = sPP.trophies + "";
}

public void CheckName(string name)
{
    if (!string.IsNullOrEmpty(name) && name.Length >= 3)
    {
        sv.name = name;
        Debug.Log("Your name: " + name);
    }
    else Debug.Log("You think I'm kidding with you? Say me your normal name!");
}
public void GetPath()
{
```

```
#if UNITY_ANDROID && !UNITY_EDITOR
        path1 = Path.Combine(Application.persistentDataPath, "SaveSettings.json");
        path2 = Path.Combine(Application.persistentDataPath, "SavePlayerProgress.json");
        path3 = Path.Combine(Application.persistentDataPath, "PlayerLvlRequirements.json");
#else
        path1 = Path.Combine(Application.dataPath, "SaveSettings.json");//Шлях до файлу де будуть зберігатися наші налаштування
        path2 = Path.Combine(Application.dataPath, "SavePlayerProgress.json");
        path3 = Path.Combine(Application.dataPath, "PlayerLvlRequirements.json");
#endif
        if (File.Exists(path1))
        {
            sv = JsonUtility.FromJson<SaveSettings>(File.ReadAllText(path1));
        }
        else
        {
            sv.sound = true;
            sv.music = true;
            sv.dialog = true;
            sv.language = "english";
            File.WriteAllText(path1, JsonUtility.ToJson(sv));
        }
        if (File.Exists(path2))
        {
            sPP = JsonUtility.FromJson<SavePlayerProgress>(File.ReadAllText(path2));
        }
        else
        {
            sPP.totalExpirience = 0;
            sPP.currentPlayerLevel = 1;
            sPP.trophies = 0;
            sPP.currentFreeUpPoints = 0;
```

```
    sPP.currentSilverCoins = 0;
    sPP.currentGoldCoins = 0;
    sPP.currentGems = 0;
    sPP.currentCoinsForPlay = 0;
    sPP.playerTotalStrenght = 1;
    sPP.PlayerAttackDamage = new Vector2(0, 0);
    sPP.playerSpeed = 3f;
    sPP.playerMaxHealth = 3;
    sPP.playerHealthRegeneration = 0f;
    sPP.playerMaxArmor = 0;
    sPP.playerArmorRecovering = 0f;
    sPP.dmgAbsorptionPercentage = 0;
    sPP.playerMaxStamina = 1;
    sPP.playerStaminaRecovering = 0.3f;
    sPP.playerStaminaUsing = 0.3f;
    sPP.playerViewRadius = 10f;
    sPP.playerAttackRadius = 0f;
    sPP.playerStrenghtfromLvl = 0;
    sPP.playerSpeedfromLvl = 0;
    sPP.playerHealthfromLvl = 0;
    sPP.playerStaminafromLvl = 0;
    sPP.helmet = 0;
    sPP.brigandine = 0;
    sPP.boots = 0;
    sPP.weapon = 0;
    File.WriteAllText(path2, JsonUtility.ToJson(sPP));
}
if (File.Exists(path3))
{
    PLR = JsonUtility.FromJson<PlayerLvlRequirements>(File.ReadAllText(path3));
}
else
```

```
            {
                PLR.ExpForLvl25Plus = 25000;
                File.WriteAllText(path3, JsonUtility.ToJson(PLR));
            }
        }

        public void MadeExpBar()
        {
            int NextLvl;
            string VariableName;
            NextLvl = sPP.currentPlayerLevel + 1;
            VariableName = "ExpForLvl" + NextLvl;
            int FullExpBar = PLR.ExpForLvl[NextLvl,1] - PLR.ExpForLvl[sPP.currentPlayerLevel,
1];
            int CurExpFilling = sPP.totalExpirience - PLR.ExpForLvl[sPP.currentPlayerLevel, 1];
            double CurPercentOfExp =
(Convert.ToDouble(CurExpFilling)/(Convert.ToDouble(FullExpBar) / 100))/100;
            float ExpFill = (float)CurPercentOfExp;
            LvlProgressBar.fillAmount = ExpFill;
            ExpText.GetComponent<Text>().text = sPP.totalExpirience + " / " +
PLR.ExpForLvl[NextLvl,1];
        }
    }
    [Serializable]
    public class SaveSettings
    {
        public string name;
        public bool sound ;
        public bool music;
        public bool dialog;
        public string language;
    }
```

```
public class SavePlayerProgress
{
    public int totalExpirience;
    public int currentPlayerLevel;
    public int trophies;
    public int currentFreeUpPoints;
    public int currentSilverCoins;
    public int currentGoldCoins;
    public int currentGems;
    public int currentCoinsForPlay;
    public int playerTotalStrenght;
    public Vector2 PlayerAttackDamage;
    public float playerSpeed;
    public int playerMaxHealth;
    public float playerHealthRegeneration;
    public int playerMaxArmor;
    public float playerArmorRecovering;
    public int dmgAbsorptionPercentage;
    public int playerMaxStamina;
    public float playerStaminaRecovering;
    public float playerStaminaUsing;
    public float playerViewRadius;
    public float playerAttackRadius;
    public int playerStrenghtfromLvl;
    public int playerSpeedfromLvl;
    public int playerHealthfromLvl;
    public int playerStaminafromLvl;
    public int helmet;
    public int weapon;
    public int brigandine;
    public int boots;
}
```

```
public class PlayerLvlRequirements
{
    public int[,] ExpForLvl = { {0, 0}, {1, 100},{ 2, 200}, { 3, 400 }, { 4, 800 }, { 5, 1500 }, { 6, 3000 }, { 7, 5000 }, { 8, 7500 }, { 9, 11000 }, { 10, 15000 }, { 11, 35000}, { 12, 55000 }, { 13, 75000 }, { 14, 95000 }, { 15, 115000 }, { 16, 135000 }, { 17, 155000 }, { 18, 175000 }, { 19, 195000 }, { 20, 215000 }, { 21, 235000 }, { 22, 255000 }, { 23, 275000 }, { 24, 315000 }, { 25, 340000} };
    public int ExpForLvl25Plus;
}
```