Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка мобільної шутер гри «Space confrontation»»

Виконав: студент 3 курсу
групи 1ПІ-19мс2
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Самарасінгхе Д. С. В.
(прізвище та ініціали)

Керівник: доц. каф. ПЗ Катєльніков Д.І.
(прізвище та ініціали)

Рецензент: к.т.н., ст. вик. КН Озеранський В.С.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«____» _____ 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
"_25_" __березня__2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Самарасінгхе Дмитро Сампат Вишневські

1. Тема роботи – **Розробка мобільної шутер гри «Space confrontation».**

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від "_24_" _березня_2022 року №_66._

2. Строк подання студентом роботи _____14 червня 2022 року_____

3. Вихідні дані до роботи: операційна система – Android, середовище програмування Visual Studio 2019, мова програмування – C#, технологія – .NET, ігровий рушій – Unity2D версія 2019.4.39f1, розмір вікна додатку під всі мобільни пристрої, кольоровий режим: TrueColor.

4. Зміст розрахунково-пояснювальної записки: аналіз використання систем прийняття рішень в ігрових додатках, обґрунтування доцільності розробки; теоретичні основи розробленого методу; варіантний аналіз та обґрунтування вибору засобів реалізації ігрової системи; розробка моделей і структур ігрової

системи; розробка програмного коду ігрової системи; тестування роботи ігрової системи.

5. Перелік графічного матеріалу: мета, об’єкт та предмет дослідження; задачі дослідження; методи дослідження; наукова новизна; практичне значення; обґрунтування доцільності розробки; метод високорівневого ускладнення гри; реалізації ігрової системи; програмна реалізація ігрового додатку; тестування; висновки, апробація, публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Катєльніков Д. І., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання _____25 березня 2022 року_____

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану галузі, обґрунтування доцільності розробки	26.03.2022 – 15.04.2022	Вик.
2	Розробка методу високорівневого ускладнення гри	16.04.2022 – 25.04.2022	Вик.
3	Розробка моделі внутриігрового магазину	26.04.2022 – 10.05.2022	Вик.
4	Розробка алгоритмів роботи мобільного ігрового додатку	11.05.2022 – 25.05.2022	Вик.
5	Розробка програмного коду ігрової системи	26.05.2022 – 31.05.2022	Вик.
6	Тестування роботи ігрової системи	01.06.2022 – 10.06.2022	Вик.

Студент _____ Самарасінгхе Д.С.В.
(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ Катєльніков Д. І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 157 сторінок формату А4, на яких є 49 рисунків, 3 таблиця, список використаних джерел містить 22 найменувань.

В бакалаврській дипломній роботі запропоновано ігрову систему, орієнтовану на тренування швидкості реакції гравця в ігровому процесі та тактичного мислення в ігровому середовищі. Розробленно процес розвитку та модифікацій ігрового юніта. Супротивником є запрограмовані ворожі юніти, що з кожним рівнем важкості мають нові випробування для гравця. Розроблено метод і модель ігрової системи, що реалізована в демонстраційному додатку «Space confrontation».

Спроєктовано архітектуру ігрової системи. Створено мобільний ігровий додаток відповідно до поставлених вимог, проведено його тестування. Розроблено керівництво користувача.

Програмний продукт було створено з використанням мови програмування C#. Для розробки графічного інтерфейсу було використано ігровий рушій Unity2D версія 2019. В якості середовища розробки програмного забезпечення було обрано Microsoft Visual Studio 2019. Для створення ігрових об’єктів було використано відриті ресурси з наборами спрайтів та анімацій. При необхідності об’єкти редагувалися у Adobe Photoshop.

Ключові слова: мобільна шутер гра, класична аркадна гра, області, Unity C#, 2D іграа.

ABSTRACT

The bachelor's thesis consists of 157 A4 pages, which have 49 figures, 3 tables, a list of sources used contains 22 titles.

The bachelor's thesis offers a game system focused on training the speed of the player's reaction in the game process and tactical thinking in the game environment. The process of development and modifications of the game unit has been developed. The enemy is programmed enemy units, which with each level of difficulty have new challenges for the player. The method and model of the game system, which is implemented in the demonstration application "Space confrontation", has been developed.

The architecture of the game system is designed. Created a mobile game application in accordance with the requirements, tested it. Developed user manual.

The software product was created using the C # programming language. Unity2D gaming engine version 2019 was used to develop the graphical interface. Microsoft Visual Studio 2019 was chosen as the software development environment. Open resources with sets of sprites and animations were used to create game objects. If necessary, objects were edited in Adobe Photoshop.

Keywords: mobile shooter game, classic arcade game, area, Unity C #, 2D game.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний світ цілком і повністю поглинений мобільною лихоманкою. Чи знайдеться у нашому оточенні хоча б одна людина, яка не користується смартфоном. Потреба смартфонів зросла з блискавичною швидкістю, а компанії-виробники вкладають величезні суми в розвиток мобільних технологій і популяризацію їх на ринку [1]. Мобільна розробка – стрімко зростаюча галузь програмування, адже кількість мобільних пристроїв значно перевищує кількість персональних комп'ютерів, і ця тенденція буде тільки зростати.

Багато людей користуються мобільними телефонами всюди – на роботі, на навчанні, вдома, на відпочинку. В останні роки мобільний геймінг став справжнім драйвером зростання для індустрії завдяки своїй доступності для користувачів.

Розвиток мобільних пристроїв стимулює створення відповідного програмного забезпечення. Ігрові програми зручно використовувати на мобільних пристроях через їх доступність.

Розвиток ігрової індустрії зумовлює розробку нових сучасних мобільних ігор, які можуть використовуватися в різних галузях діяльності людини [1].

Крім розважальної мети, мобільні ігрові технології можуть надавати користувачам можливість займатися інтелектуальним саморозвитком, тренувати логічне і стратегічне мислення. Тому актуальною є розробка мобільних ігор, орієнтованих на розвиток стратегічного мислення та моторики гравця.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася відповідно до плану науково-дослідних робіт кафедри програмного забезпечення.

Мета – підвищення можливостей тренування швидкості реакції та адаптивності гравця до рівня ігрової стратегії шляхом розробки та використання мобільної ігрової системи з широким функціоналом посилення юніта і розвиненою системою адаптивного рівневого ускладнення гри, що дозволить

реалізувати ігрову систему, орієнтовану на розвиток конкретного користувача.

Основними задачами дослідження є:

– розробити метод високорівневого ускладення гри та удосконалити його автоматизацією полегшеного створення наступних рівнів;

– розробити модель ігрової системи;

– розробити маніпулятивну систему керування космічним крейсером за допомогою сенсорного екрану мобільного телефону;

– спроєктувати внутрішній ігровий магазин для остащення ігрових юнітів;

– розробити основні модулі ігрового додатку;

– спроєктувати зручний користувацький інтерфейс;

– розробити та удосконалити графічні ефекти, об’єкти та прифаби для ігрового додатку;

– надати музичне супроводження під час гри чи під час знаходження в ігровому інтерфейсі;

– розробити мобільний ігровий додаток «Space confrontation»;

– провести тестування мобільної гри в програмному ігровому рушії та на мобільних пристроях.

Об’єкт дослідження – процеси розробки логічних ігор для мобільних пристроїв.

Предметом дослідження є засоби реалізації мобільної гри, орієнтованої на тренування швидкості реакції користувача у процесі прийняття тактичних рішень.

Методи дослідження:

– теорія алгоритмів для побудови алгоритмів функціонування ігрових систем;

– методи створення графічних зображень для розробки ігрових сцен;

– методи побудови компонентів гри та їхніх взаємозв’язків для створення загальної структури збереження та обробки інформації;

– методи проєктування ігрових рівнів для створення програмного

ігрового додатку;

– методи тестування для перевірки працездатності розробленого програмного продукту.

Наукова новизна одержаних результатів:

1. Подальшого розвитку дістав метод високорівневого ускладнення гри, який, на відміну від існуючих, дозволяє адаптивно посилювати ворожі юніти в ігрових рівнях з урахуванням професійних можливостей конкретного користувача, що дозволить підлаштувати зростання рівня складності гри для тренування швидкості реакції користувача.

2. Подальшого розвитку дістала модель ігрової системи, що, на відміну від існуючих, орієнтована на адаптивне високорівневе ускладнення гри, що дозволяє ігровому середовищу створити комфортні умови тренінгу для конкретного користувача.

Практична цінність отриманих результатів. Практична цінність полягає у розробці ігрового додатку «Space confrontation», який можливо використовувати для тренування швидкості реакції користувача.

Достовірність теоретичних положень підтверджена результатами тестування розробленого ігрового додатку.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У науковій роботі [2], опублікованій у співавторстві, автору належить розробка загального алгоритму роботи гри «SPACE CONFRONTATION».

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на Всеукраїнській науково-технічній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2022). Секція Інформаційні технології та комп‘ютерна інженерія».

Публікації. Основні результати дослідження опубліковані в науковій роботі – в тезах доповіді на Всеукраїнській науково-технічній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2022). Секція Інформаційні технології та комп‘ютерна інженерія» [2].

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану мобільних ігрових додатків

Аналіз передбачає ознайомлення з основними концептуальними поняттями, що використовуються при реалізації підходів до проєктування мобільних ігор, познайомити з технологією розробки мобільних ігор, з програмним забезпеченням, що застосовується для розробки окремих модулів мобільних ігор, також досліджуються інфологічні, даталогічні та фізичні моделі, які використовуються в мобільних іграх та інших технічних системах [3]. Розглянемо історію розвитку ігрових додатків та ігрових програм на основі мобільних пристроїв.

За останні 20 років мобільний телефон щільно увійшов у кругообіг сучасного життя. За ці 20 років пройдено гіганський еволюційний шлях, що призвів до перетворення стільникового телефону на ігровий, мультимедійний інтернет пристрій. Розповімо про історію платформ для телефонів, що набули найбільшої популярності серед геймерів і про шлях, який пройшли ці першовідкривачі мобільного геймінгу, а також про перспективи розвитку цього напряму.

Поговоримо про перші телефони та перші JAVA мідлети. Після сплеску мобільних телефонів наприкінці 90-х років, компанії-виробники, усвідомили, що стільникові телефони мають величезні перспективи розвитку. Основна затримка на шляху масового застосування тодішніх апаратів – це була їхня величезна ціна. Але вже на 2001-2002 років з'являються масові пропозиції із середньою ціною, це вже не 1000 доларів за телефон, а в рази менше. Саме в цей момент основні постачальники телефонів зійшлися на думці, що для подальшого розвитку телефонії була потрібна нова програмна платформа. На той момент було два явні претенденти на це місце. Це була BREW (Binary Runtime Environment for Wireless), розроблена компанією Qualcomm та Java 2 Micro Edition (J2ME). Надалі

розробка від Qualcomm поступилася J2ME, відмовою від неї послужила програмна замкнутість та ліцензійні угоди, які зобов'язували виробників платити за використання платформи. На боці ж Ява була її простота і відкритість, адже платформа спочатку писалася для малопотужних і не енергоємних пристроїв типу пейджера, також абсолютно не вимагала жодних ліцензійних відрахувань і вкладень. Та й Java вже на той час мала велику кількість розробників, які потенційно готові писати мобільні додатки.

За 2001-2002 роки на ринку з'явився розсип моделей за допомогою J2ME, деякі з них стали справжнім проривом. Наприклад компанія Motorola штурмом взяла Штати, а в Європі тріумфували Siemens і Nokia, з їх масовими апаратами Si SL45i/S45/M50 і Nokia 6310i і 3310.

Першою на мобільний геймінг кинула погляд Nokia, анонсувавши Nokia 3410, що позиціонується як «ігрова» модель. У телефоні була жменька ігор, що найбільше запам'яталося: інтерпретація відомої гри Pacman, гра Racket - симулятор тенісу, знаменита Змійка та історія Лунохода. Розташовані вони були в пункті меню «додатки», справжньою інновацією було впровадження WAP в апарати, що дозволило зробити онлайн каталог ігор, де розробники могли викладати власні проекти (зрозуміло не безкоштовно, Nokia відходили чималі відрахування). Менш ніж за півроку список доступних налічував до 300, 40% із яких були іграми. Як і слід було очікувати, фінський виробник свого домігся, мобільні ігри таки асоціювалися зі словом Нокіа, хоча були частково сумісні з апаратами інших компаній.

Переважна більшість ігор написано в жанрі «аркада» та «головоломка». зрідка зустрічаються представники інших жанрів. Основним обмеженням для «розмаїття жанрів» є пристрій введення. Наприклад, для зручної гри в 3D-шутер бажана можливість одночасного наведення на ціль і стрільби – використання двох клавіш одночасно, що утруднено на маленьких клавіатурах мобільного телефону, а стратегії в реальному часі спочатку орієнтувалися на управління джойстиком або комп'ютерною мишею.

Наприкінці 2000-х років поширилися сенсорні телефони, на них основним жанром стала фізична головоломка.

Щоб залучити потенційного покупця, часто випускаються мобільні ігри з використанням відомого бренду (наприклад, Doom RPG), або за мотивами фільму, що нещодавно з'явився [4].

Окремо виділяють аркадні ігрові автомати та мобільні пристрої, як смартфони чи планшетні комп'ютери, на яких також можна грати у відеоігри.

Деякі ігри можуть запускатися на різних платформах. Ця їх здатність називається кросплатформністю або багатоплатформністю. Для досягнення кросплатформності розробляється окрема версія гри для кожної платформи, або ця гра використовує технології, з яким працюють декілька платформ. Наприклад, браузерні ігри потребують лише наявності інтернет-браузера.

Отже сьогодні можливо виділити наступні основні жанри ігор на телефон:

– казуальні;
– королівська битва;
– головоломки;
– екшн-ігри;
– RPG.

Інформаційні технології розробки та впровадження ігрових застосувань на основі мобільних пристроїв.

Наявність операційної системи (ОС) – головна особливість, яка відрізняє смартфон від звичайного мобільного телефону. При виборі конкретної моделі 17 телефону або пристрою, операційна система часто є визначальним фактором) . Найбільш поширені операційні системи для смартфонів і платформ:

− Symbian OS – ОС займала більшу частину ринку смартфонів до кінця 2010 р На початку 2010 року на базі ОС залишається тільки 1 Платформа: Series 60, яка використовується в основному в пристроях Nokia, а також деякі моделях Samsung;

− BlackBerry OS (RIM) – система широко використовується в пристроях в першу чергу в Сполучених Штатах, так як спецслужби деяких країн не зацікавлені у використанні смартфонів в країні через те, що всі вхідні і вихідні дані зашифровані з використанням алгоритму шифрування AES;

− Windows Mobile і Windows CE – компактна операційна система Microsoft, випущені з 1996 року і займала найбільший сегмент ринку ОС для смартфонів до 2010 року, в даний час проходить поетапну відмову від підтримки і розвитку;

− Windows Phone 7 – розробка від Microsoft, радикально відрізняється від Windows Mobile;

− Palm OS – одна з популярних платформ, хоча даний час мобільні телефони на базі Palm OS малопоширені. Останній смартфон під управлінням операційної системи була випущений в кінці 2007 року (Palm Centro);

− Linux – широкого поширення ця операційна система на мобільних пристроях не отримала, проте її розвиток традиційно вважається перспективним напрямком. Смартфони на базі Linux поширюються головним чином в Азії;

− Bada – новітня мобільна платформа, розроблена компанією Samsung. Першим телефоном на новій платформі став S8500 Wave;

− Android – портативна (мережева) операційна система для смартфонів, планшетих ПК, електронні книг, цифрових плеєрів, годин і нетбуків на базі ядра Linux. Спочатку розроблена Android Inc., яку потім купив Google. Згодом Google ініціювала створення альянсу Open Handset Alliance (OHA), який зараз займається підтримкою і подальшим розвитком платформи. Android дозволяє створювати додатки на основі Java, який управляють пристроєм через розроблені Google бібліотеки. Android Native Development Kit дозволяє системі використовувати бібліотеки і компоненти додатків, написаних на C та іншими мовами;

− ОС IOS (до 24 червня 2010 року – iPhone OS) – це мобільна операційна система, розроблена і виготовлена американською компанією Apple. Вона була випущена в 2007 році; спочатку – для iPhone і IPod Touch, а пізніше – для таких

пристроїв, як IPad і Apple TV. На відміну від Windows Phone і Google Android, доступна тільки для пристроїв, вироблених Apple;

1.2 Порівняльний аналіз аналогів

Проведемо порівняльний аналіз деяких існуючих мобільних шутер ігор зі схожим принципами роботи, що передбачають боротьбу гравця з ворожими юнітами ігрового процесу, та визначимо наявні недоліки конкурентів. Аналіз існуючих програмних ігрових додатків космічних баталій.

Мобільний ігровий додаток «Galaxy Shooter: Air Force War» [5] зображенна на рисунку 1.1 – аркадна космічна шутер гра, що має такі особливості:

- розрахований на багато користувачів режим;
- просте управління з торкніться і утримуйте, не потрібен підручник;
- різні космічні кораблі з їх власних унікальних видів зброї, missles та спеціальні здібності, щоб допомогти вам перемогти всіх цих босів;
- приголомшливі графіка та звуки вишуканої якості, як ви ніколи не бачили раніше. Цілком відрізняється від старих ретро космічний шутер, грати і відчувати його;
- захоплюючі кампанії з більш ніж 70 рівнів повний ворогів простір дозволяє бачити простір від землі;
- чудовий ігровий механіці з відмінним прогресії системою.

Рисунок 1.1 – Мобільний ігровий додаток «Galaxy Shooter: Air Force War»

Мобільний ігровий додаток «Пірати Галактики» [6] зображенна на рисунку1.2, про піратів галактики, мисливців за головами які мандрують у пригодах в космічному просторі на інопланетних космічних дирижаблях.

«Пірати Галактики» – уайпопулярніша гра-стрілялка 2019 року. Створена на основі класичних ігор-стрілялок, гра має всі найкращі характеристики класики, але в неї вбудована абсолютно нова ігрова система. З чудовою графікою та привабливою історією.

На вибір пропонується безліч дирижаблів – кожен зі своїм умінням, коли ви граєте на новому кораблі, це буде новий досвід: нові персонажі та унікальна зброя, два помічники стануть твоїм супутником у цій грандіозній пригоді. Процеси механізмів ігри: торкніться екрана, щоб керувати кораблем, купівля нових кораблів.

Гра «Пірати Галактики» має такі характерні особливості:

– гарна графіка і неймовірна музика;
– плавно на кожному пристрої;
– багато режиму гри: кампанія, нескінченний, PvP;
– інтенсивна битва з босом кожні 3 рівня.

Рисунок 1.2 – Мобільний ігровий додаток «Пірати Галактики»

Мобільний ігровий додаток «Sky Force 2014» [7] зображенна на рисунку 1.3 – легендарна мобільна гра, що поєднує популярність класичних аркад зі стріляниною з новітніми технологіями смартфонів, Sky Force 2014 пропонує незвичайний шутер із неймовірним новим соціальним геймплеєм.

Десять років тому компанія IGN заявила: «Простіше кажучи, Sky Force – приголомшлива гра». Має приголомшливу 3D-графіці, інтуїтивно-зрозумілому сенсорному управлінню та потужній системі оновлення, які змусять вас захопитися грою на кілька годин поспіль.

Гра «Sky Force 2014» має такі характерні особливості:

– барвисті рівні із захоплюючими місіями, які необхідно пройти;

– численні екстремальні битви з босами;

– ви можете оновлювати свій захист, зброю, ракети, лазери, мегабомби та магніти;

– ви можете збільшувати свої кінцеві бали за допомогою численних досягнень у грі;

– повний голосовий супровід та неймовірний електронний саундтрек.

Рисунок 1.3 – Мобільний ігровий додаток «Sky Force 2014»

Результати аналізу сучасних мобільних ігорових додатків, орієнтованих на тренування швидкості реакції користувача та розвиток тактичного мислення в космічному просторі, зведемо в порівняльну таблицю (табл. 1.1), яка відображає головні характеристики мобільного грового додатку.

Таблиця 1.1 – Результати порівняльного аналізу аналогів

Функції	Galaxy Shooter: Air Force War	Пірати Галактики	Sky Force 2014	Space confrontation
Система внутриігрового магазину	-	+	-	+
Адаптація та унікальність бойових можливостей	-	+	+	+

Різні види ворожих юнитів та ігрових босів	-	+	+	+
Інтуїтивно зрозумілий інтерфейс користувача	+	-	+	+
Є реклама	+	+	+	-

Відповідно до таблиці 1.1 та результатів порівняльного аналізу аналогів розробка власної мобільної шутер гри «Space confrontation» є затребуваною. Отриманий ігровий продукт зможе покрити недоліки існуючих рішень та забезпечити новий функціонал для покрашення моторики та інтелектуального тактичного мислення гравця.

1.3 Варіантний аналіз і обгрунтування вибору засобів реалізації програми

Для розробки мобільних додатків набирає популярності і актуальності мова програмування Java, але в реалізації ігрового мобільного додатка є конкурентна мова програмування C#, так як для неї було розробленно певне середовище розробки Unity де використовується за часту мова програмування C# для створення мобільних додатків як 3D так і 2D [8].

Обидві мови, C # і Java є продоженням мови C ++. Вони були розроблені в різний час, в основному, в конкуруючому середовищі і мають свої подібності та відмінності.

Всіх мобільних розробників можна розділити на дві категорії, в залежності від програмного забезпечення, для якого вони створюють програми – iOS розробники і Android-розробники. Фахівці з першої категорії вважаються

найбільш прибутковими на ринку праці, більш того, після появи мов Swift і Objective-C створення додатків для Apple стало дуже легким і зручним. Програмісти, які створюють програми для Android, використовують у своїй діяльності мову Java, яка вважається найнадійнішою для розробки мобільних додатків для цієї операційної системи.

Проєктування програмного забезпечення (англ. Software design) – це процес визначення архітектури, компонентів, інтерфейсів та інших атрибутів (структур даних, алгоритмів і т.д.) системи або компонента програмного забезпечення. Результатом цього процесу є проект програмного забезпечення [9].

Проєктуванню зазвичай підлягають.

– Архітектура програмного забезпечення
– Компоненти ПЗ
– Користувацькі інтерфейси

В процесі проєктування ПЗ застосовують різні моделі – блок-схеми, ER-діаграми, DFD тощо.

Сучасні інструментальні методи проектування програмних засобів на основі мобільних пристроїв.

Unity – це потужний крос-платформний двигун для створення 2D і 3D ігор, інтерфейс програми зображений на рисунку 1.4. Двигун можна спробувати безкоштовно, щоб створити прототип або альфа-версію. Щоб опублікувати створену гру, потрібно купити підписку за $25 або $125 на місяць. Вартість ліцензії для компаній визначається окремо [10].

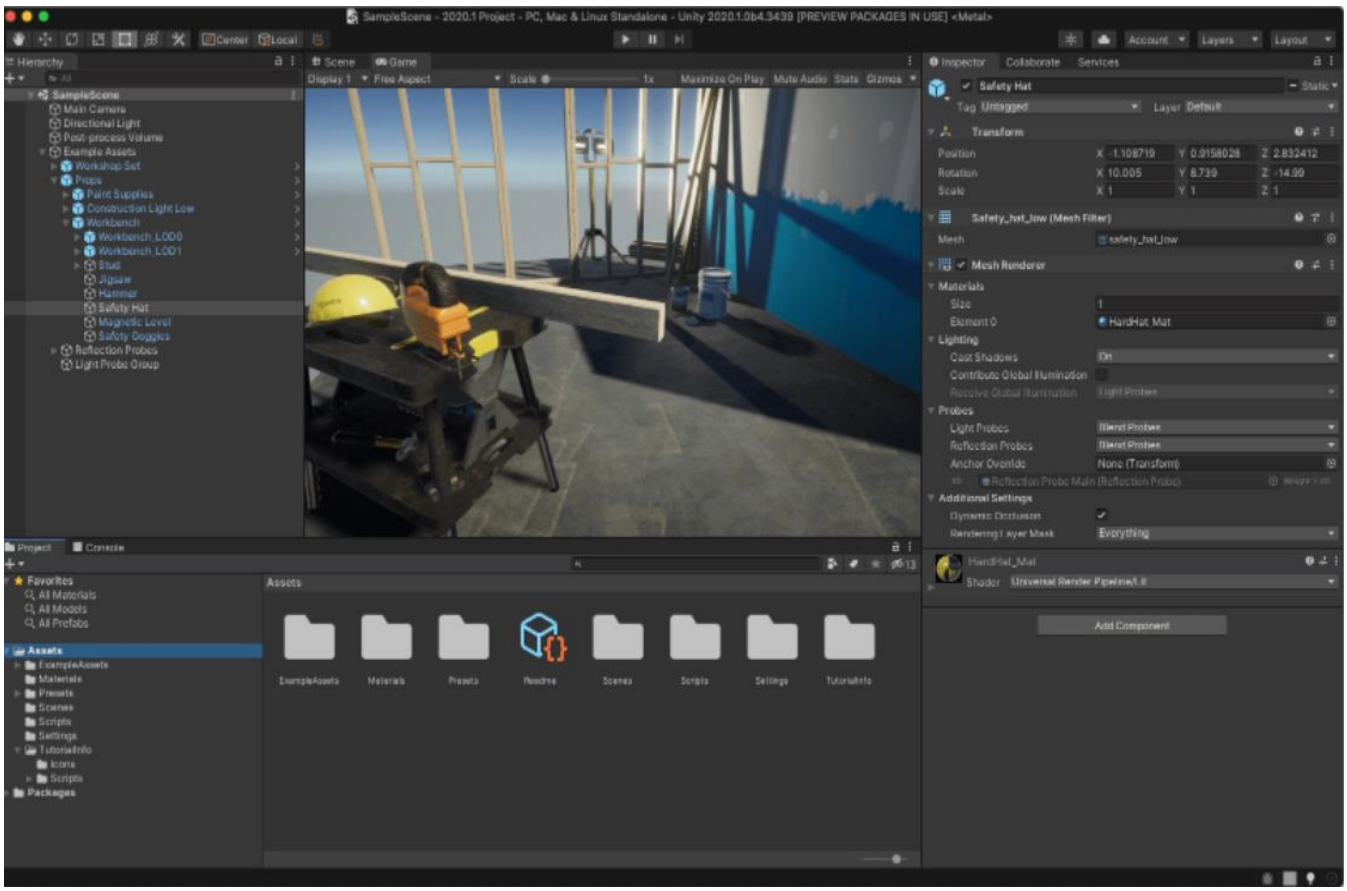

Рисунок 1.4 – Unity інтерфейс програмного середовища

На Unity створили такі популярнві за своїх часів ігри:

– Lara Croft GO;

– Angry Birds 2;

– Pokemon GO і т.д.

Платформа Unity найбільше підходить для розробки 3D ігор, але багато розробників роблять на ній ізометричні платформери або 2D ігри.

Плюси Unity:

– компонентно-орієнтований підхід – розробник прописує об'єкту компоненти на кшталт можливості управління об'єктом та моделі поведінки;

– велика бібліотека ассетів та плагінів, які можна використовувати для створення прототипу та готової гри. Наприклад, можна імпортувати модель штучного інтелекту ворогів;

– Unity підтримує новітні технології рендерингу на кшталт трасування променів, можна робити ігри з фотореалістичною графікою.

Мінуси Unity:

– потрібно багато програмувати;

– погана оптимізація «з коробки», гру доведеться оптимізувати вручну, щоб у неї було зручно грати;

– велика кількість вбудованих компонентів виливається у великий обсяг готової гри. Найпростіші проекти займають від 100 МБ і більше, а користувачі мобільних платформ не люблять ставити великі ігри.

Unreal Engine – це двигун Epic Games, розробників Fortnite. Unreal Engine можна користуватися безкоштовно [11], але за умови: якщо гра збере більше $3000 прибутку, Epic Games отримають 5% роялті, інтерфейс програми зображений на рисунку 1.5.

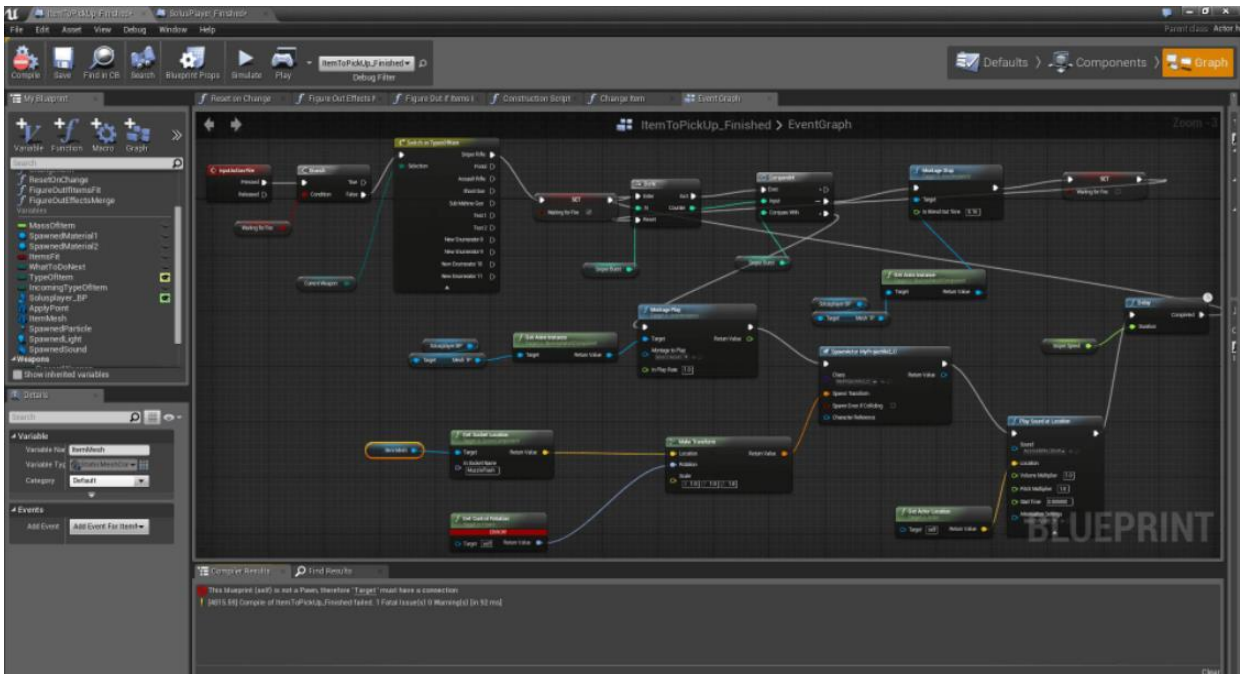

Рисунок 1.5 – Unreal Engine інтерфейс програмного середовища

Популярні мобільні ігри на Unreal Engine:

– Fortnite Mobile;

– Life is Strange;

– Mortal Kombat;

– Pro Evolution Soccer 2020.

Плюси Unreal Engine:

– можна робити ігри без програмування. Для цього в Unreal Engine є візуальний редактор Blueprints, за допомогою якого можна писати скрипти та настроювати поведінку ігрових об'єктів;

– велика кількість безкоштовних ассетів, що допоможуть у розробці;

– вбудований інструмент для оптимізації ігор для мобільних платформ.

Мінуси Unreal Engine:

– у двигуна неідеальна оптимізація. Якщо додати на карту занадто багато об'єктів або спробувати створити великий безшовний світ, така гра

гальмуватиме. Справа в тому, що Unreal Engine обраховує всі предмети незалежно від того, чи потрапляють вони у поле зору гравця;

– інтерфейс розрахований на новачків, багато кнопок швидкого доступу розташовані невдало;

– при створенні великих ігор розробникам потрібно серйозно оптимізувати.

Для розробки мобільної шутер гри «Space confrontation» було обрано мову C#, так як є певні практичні навички в розробці мобільних додатків цією мовою програмування. C# – об’єктно-орієнтована мова високого рівня. Перевагою цієї мови програмування для створення гри є те, що вона об'єднує переваги інших поширених мов програмування і має низку власних корисних особливостей, зокрема:

– підтримку інкапсуляції, наслідування і поліморфізму;

– використання обробки виключень;

– автоматичну ініціалізацію змінних;

– використання «збору сміття»;

– можливість перевантаження операторів;

– чітку типізацію змінних;

– підтримку компонентів.

В якості ігрового рушія було обрано Unity – популярне середовище для створення ігор, що має велику кількість бібліотек потужних і зручних інструментів та власний магазин з доступними прикладами різних моделей юнитів та прифабів які можна використати в безплатному доступі.

Першою перевагою є можливість писати скрипти мовою C# в середовищі програмування Visual Studio 2019, оскільки саме ця мова програмування була обрана для написання гри. Серед інших основних переваг Unity можна відзначити наступні:

– зрозумілий і доступний інтерфейс;

– можливість створення 2D ігор;

– підтримка імпорту великої кількості форматів;

– можливість тестування гри безпосередньо в редакторі.

Отже, C# і Unity 2D – найкращий вибір для написання гри «Space confrontation», оскільки ця мова програмування має перелік переваг, яких немає у інших високорівневих мовах програмування, які є попередниками мови C#.

1.4 Постановка задач бакалаврської дипломної роботи

Завданням бакалаврської дипломної роботи є створення мобільного ігрового додатку «Space confrontation», орієнтованої на тренування швидкості реакції користувача в процесі прийняття тактичних рішень в ігровому середовищі.

Для злагодженої роботи гравеця мобільної гри необхідно розробити метод високорівневого ускладнення, який, на відміну від існуючих, дозволяє адаптивно посилювати ворожі юніти в ігрових рівнях з урахуванням професійних можливостей конкретного користувача, що дозволить підлаштувати зростання рівня складності гри для тренування швидкості реакції користувача. Необхідно розробити модель ігрового додатку.

Гра буде створена в ігровому рушію Unity2D версія 2019.4.39f1 з використанням мови програмування C# в середовищі програмування Visual Studio 2019, для покращення навиків програмування і вивчення методів розробки мобільного додатку.

Буде забезпечений музичним супровідом та можливістью переривання гри з подальшим виходом у головне меню. При невдачі гравця в проходженні гри передбачається виведення відповідного тексту про можливість почати гру спочатку чи перехід у головне меню. Також в головному меню буде реалізованно магазин для того, щоб гравець використовав отримані в ігровому процесі внутрішню валюту для купівлі нового виду космічного крейсеру для полегшення проходження наступного рівня з більш важкими супротивниками та босами. Серед графічних об’єктів потрібно створити набір бойових космічних крейсерів та візуальних ефектів пострілу та знищення.

Отже, «Space confrontation» – це однокористувацький мобільний додаток (Single player game), дії якого відбуваються у космічному просторі, один бойовий космічний крейсер протистоїть цілій армаді ворожих космічний крейсерів, які з’являтимуться в певному проміжку часу, а в кінцевому рівні гравець стикатиметься с босом окремої раси космічної пригоди, набираючи бали. За отриману за перемогу внутрішню валюту можна буде придбати в магазині нові види космічних крейсерів з більш модифікованим озброєням для більш різноманітного гемплею для гравця, що дасть можливість проявити своє тактичне мислення в ігрі. Еталонним проходженням гри можна вважати перемогу над усіма рівнями важкості на максимум 100%.

Для зручності проходження гри передбачено систему можливої допомоги гравцеві:

– можливість отримувати баф у вигляді додаткого життя чи броні крейсеру;

– зміна бойових літаків, кожен з яких має унікальний вигляд і здібності в бою;

– додати музичний супровід;

– створити можливість виключення музики і звуків у випливаючому меню.

Головними задачами бакалаврської дипломної роботи є:

– розробка методу високорівневого ускладення гри та удосконалення його автоматизацією полегшеного створення наступних рівнів;

– розробка моделі ігрової системи;

– розробка маніпулятивнооїї системи керування косміним крейсером за допомогою сенсорного екрану мобільного телефону;

– проєктування внутрішнього ігрового магазину для остащення ігрових юнітів;

– розробка основних модулів ігрового додатку;

– проєктування зручного користувацького інтерфейсу;

– розробка та удосконалення графічних ефектів, об’єктів та прифаб для ігрового додатку;

– надання музичого супроводу під час гри чи під час знаходження в ігровому інтерфейсі;

– розробка мобільного ігрового додатоку «Space confrontation»;

– проведення тестування мобільної гри в програмному ігровому рушії та на мобільних пристроях.

Для запуску програми апаратне забезпечення повинно відповідати мінімальним вимогам:

- мобільної серії з частотою 233 МГц і вище;
- 1 ГБ оперативної пам’яті;
- відеокарта об’ємом пам’яті не менше 400 МБ;
- сенсорний екран;
- ОС Android;

Розмір дискового простору, що займає програма: 43 166КБ. Розмір оперативної пам’яті, що займає програма: 2 500 КБайт.

1.5 Висновки

У першому розділі було розглянуто сучасний стан мобільних ігрових додатків. Було проведено порівняльний аналіз відомих мобільних ігрових програм жанру шутер-ігор в космічному просторі, таких як: Galaxy Shooter: Air Force War, Пірати Галактики, та Sky Force 2014. У результаті порівняння було виявлено, що досліджувані ігрові програмні додатки набули значної популярності серед ринку геймерів завдяки своїй зручності та доступності. Проаналізувавши переваги та недоліки відомих ігрових мобільних шутер ігор, було виявлено, що вони не надають можливості різноманітних ворожих юнитів на різних рівнях важкості проходження гри. Таким чином було доведено доцільність розробки власного програмного рішення. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власної мобільної шутер-гри.

2 РОЗРОБКА МЕТОДУ, МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ ІГРОВОЇ СИСТЕМИ

2.1 Аналіз завдання

Розробка мобільного програмного ігрового додатку починається з визначення завдання, тому є вимоги чіткого формулювання до гри, що розробляється. Буде сформулювано вимоги до додатку та його інтерфейсу, визначимо характеристики та функціональні можливості, загальні принципи реалізації мобільного ігровогу додатку.

Мобільна шутер гра «Space confrontation» матиме за мету не тільки як теренування навичок в тактичному мисленні та рефлексів користувача, а для малого покоління може використовуватися як розважальний контент.

У розробці ігорової системи необхідно:

– розробити метод високорівневого ускладення гри, та удосконалити його з полегшим створенням наступних рівнів;

– розробити модель ігрової системи;

– розробити маніпулятивну систему керування косміним крейсером, за допомогою сенсорного екрану мобільного телефону;

– спроєктувати внутрішній ігровий магазин для полегшення проходження гри та його різнобразних можливостей гри;

– розробити основні модулі ігрового додатку;

– спроєктувати зручний користувацький інтерфейс;

– розробити та удосконалити графічні ефекти, об’єкти та прифаби для ігрового додатку;

– надати музичне супроводження під час гри чи під час знаходження в ігровому інтерфейсі;

– розробити мобільний ігровий додаток «Space confrontation»;

– провести тестування в програмному ігровому рушії та на мобільних пристроях ігрового додатку.

2.2 Розробка методу високорівневого ускладнення гри і моделі роботи ігрової системи

Сутність розробленого методу полягає у тому, щоб створювати рівні за деяким шаблоном. Кожен шаблон повинен мати в собі підшаблони.

Шаблон – в мовах програмування, специфікація форми подання та правил редагування елемента даних за допомогою рядка символів, в якій кожен символ вказує на допустимий вид символу [12], що підлягає виконанню редагування для відповідної позиції значення елемента.

Підшаблони також введені, щоб зробити два однакових шаблони несхожими один на одного. Універсальними підшаблонами є класи, структури, інтерфейси і методи, які мають прототипи (параметри типів) для одного або декількох типів, які вони зберігають або використовують.

Клас універсальної колекції може використовувати параметр типу заповнювач для типу об’єктів, які в ньому зберігаються.

Параметри типу відображаються як типи його полів і типи параметрів його методів.

Для підшаблонів також в 50% випадків робимо дзеркальне відображення зліва-направо. Чим більше шаблонів і підшаблонов буде у грі, тим більш різноманітно будуть виглядати карти рівнів.

На рисунку 2.1 використовувався один і той самий шаблон, який завдяки цифрам 7, 8, 9 виглядає кожного разу по-різному.

Метод використання шаблонів є універсальним та підходить для створення не тільки аркадних відеоігор, а й для інших жанрів.

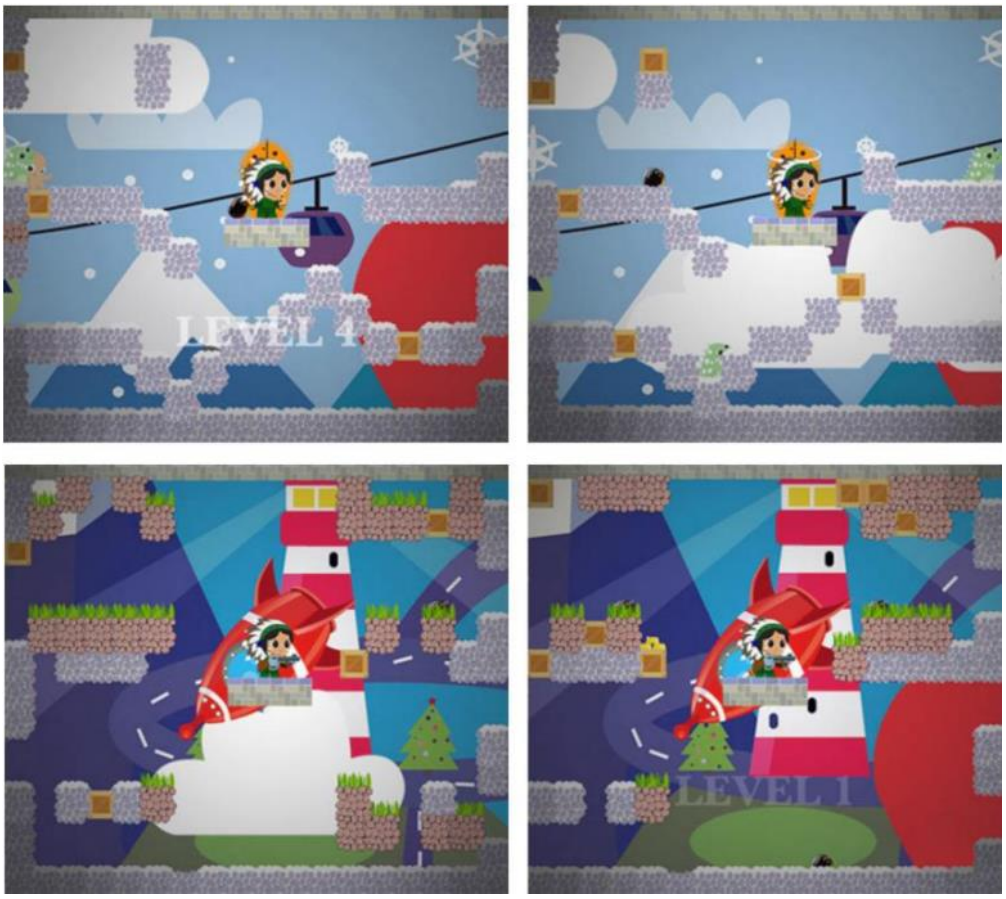

Рисунок 2.1 – Результат використання підшаблону

Сутність гри полягає в тому, щоб пережити всі хвилі з ворожими юнітами та добратися до кінця кожного рівня, переміщуючись при цьому в кінці з одного рівня в інший, де з кожним наступним рівнем будуть ускладнюватися ворожі юніти.

Уся гра має складатися з одних і тих самих елементів, які розташовуються на рівнях в різному порядку та різній кількості.

Рівень додатку, що розробляється, обов’язково має містити:

– ігрового персонажа;

– супротивників;

– задній фон планети;

– корисні предмети, що можна отримати для бонуса стрільби та ін.;

– вхід та вихід на новий рівень.

Супротивники називаються Non-Player Character (NPC) – це персонажі в рольових іграх, яким керує не гравець, а комп’ютер або майстер [13]. У комп’ютерних іграх поведінку таких персонажів визначається програмно.

У комп’ютерних і настільних рольових іграх терміном «NPC» позначаються персонажі, які взаємодіють з гравцем, незалежно від їх ставлення до ігрового персонажу. NPC можуть бути дружніми, нейтральними і ворожими. Неігрові персонажі служать важливим засобом створення ігрової атмосфери, мотивують гравців здійснювати ті чи інші дії і є основним джерелом інформації про ігровому світі і сюжеті гри.

Для процесу зацікавлення гравця з мобільним ігровим додатком було вирішено розробити метод високорівневого ускладнення гри. Даний метод наступну послідовність дій стадій:

1 Початок ігрового процесу, для ідеального проходження гравцю потрібно проходити рівні з 100% запасом життя та більше 0% запасу захисного поля, якщо ж на 2 зірки то гравець матиме 100% життя та менше 0% захисного поля.

2 Ініціалізація ігрових одиниць середнього типу – що надаватиме певний аналіз про ігрові навички гравця на протязі перших 5 рівнів.

3 Якщо гравець з легкістю пройшов 5 перших рівнів, то йому надаються наступні 5 рівнів з важкими юнітами, що матимуть більші характеристики життя, сили пошкодження по гравцю. Якщо ж гравець не зміг пройти всі перші 5 рівнів на відмінно, то йому надаються наступні п’ять легких рівнів для покращення навичок.

4 Ініціалізація легкого типу ігрових рівнів надають велику кількість ворожих юнитів, але в них менші характеристики життя, та рівень пошкодження ігрового крейсеру малий, створений для звикання до ігрового процесу та накопичення ігрової валюти для майбутньої купівлі більш модифікованого крейсеру. Після успішного проходження 5 рівнів гравцю знову надаються рівні з ігровими персонажами середнього типу.

5 Якщо гравець програв, чи то важкий рівень чи легкий, тоді гравцю надається можливіть перейти в ігровий магазин для купівлі нових крейсерів, що мають більші характеристики, у разі проходження таких рівнів гравцю надається рівень з босом.

6 Гравець може придбати новіший крейсер – в ігровому додатку є ігровий магазин, в якому можна придбати новіший крейсер, який матиме більший рівень життя, захисного поля та інших характеристик. Усього 6 ігрових крейсерів, купівля відбувається за рахунок ігрової валюти, яку гравець накопичує в кожному рівні, знищуючи ворожі юнити.

7 Ініціалізація важкого типу ігрових рівнів – надає гравцю самих складних ворожих юнитів, що можуть з легкістю знищити, якщо ігрові навички гравця замалі для проходження на три зірки кожний з рівнів.

8 Гравцю надається рівень з босом – коли гравець пройшов важкий шлях, підвищуючи свої ігрові навички для проходження гри, то йому надається рівень з босом.

9 Якщо гравець не пройшов рівень з босом, то йому надається можливість придбати в ігровому магазині новий крейсер, у іншому випадку він слідує наступним пригодам.

10 Якщо гравець пройшов всі додаткові ускладнені рівні, то він закінчує гру із задоволенням і на відмінно по 3 зірки кожний із рівнів.

11 Ініціалізація ігрових юнитів усіх типів рівнів (+50% до характеристик супротивника) – надає можливість гравцю пройти наступні типи рівнів важкості з ускладненням, так як усім ворожим юнітам надаються додаткові бонуси до характеристик +50%.

На рисунку 2.2 зображенна блок-схема алгоритму методу високорівневого ускладнення гри.

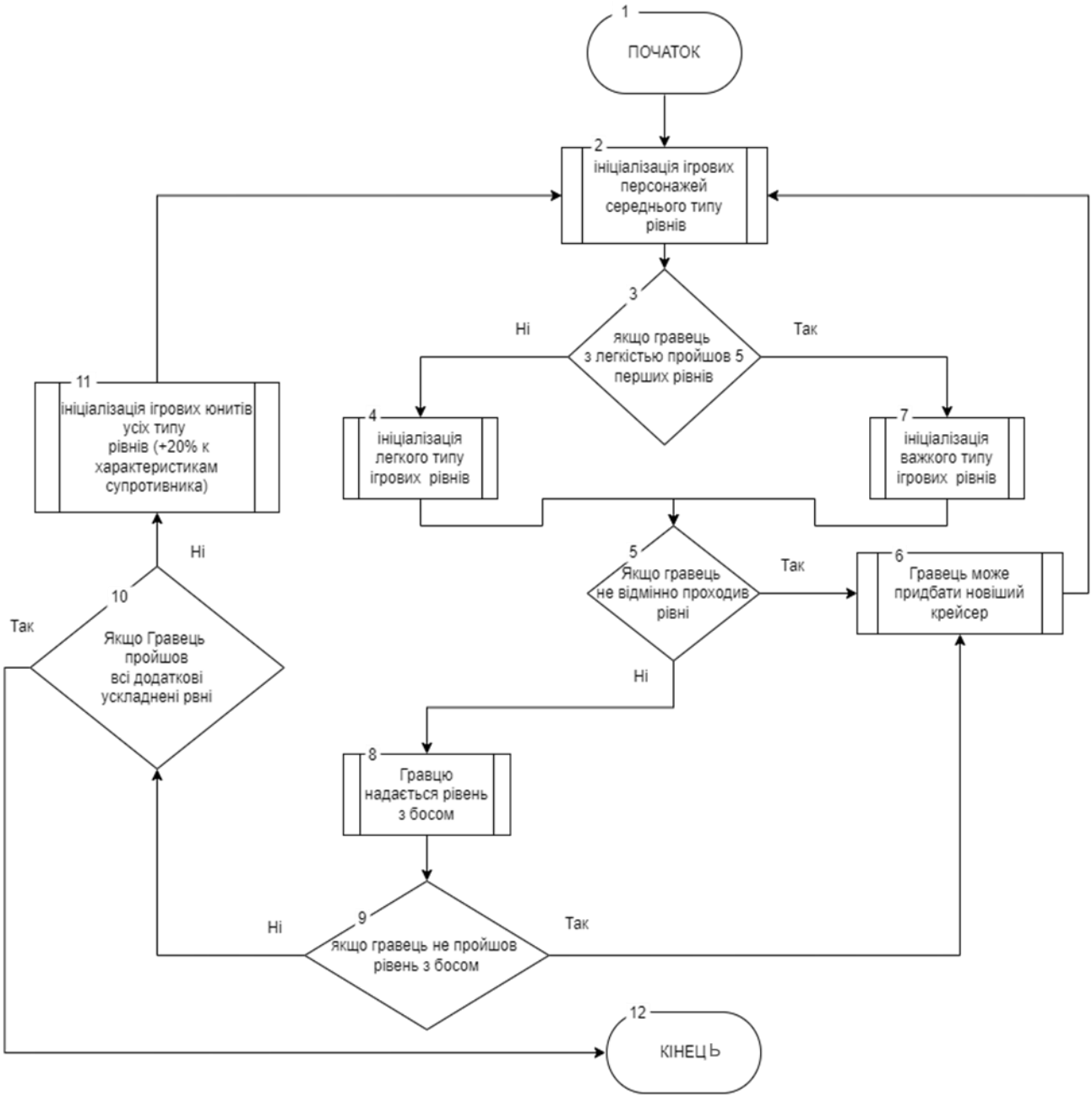

Рисунок 2.2 – Блок-схема алгоритму методу високорівневого ускладнення гри

Супротивники мають атакувати головного героя, коли той потрапляє в їх радіус атаки. Гравець повинен знищувати ворожих юнитів якнайбільше для більшої ногороди в кінці рівня та збирати усі бонуси на рівні для полегшення проходження.

Метод високорівневого ускладнення гри поданий на рисунку 2.3.

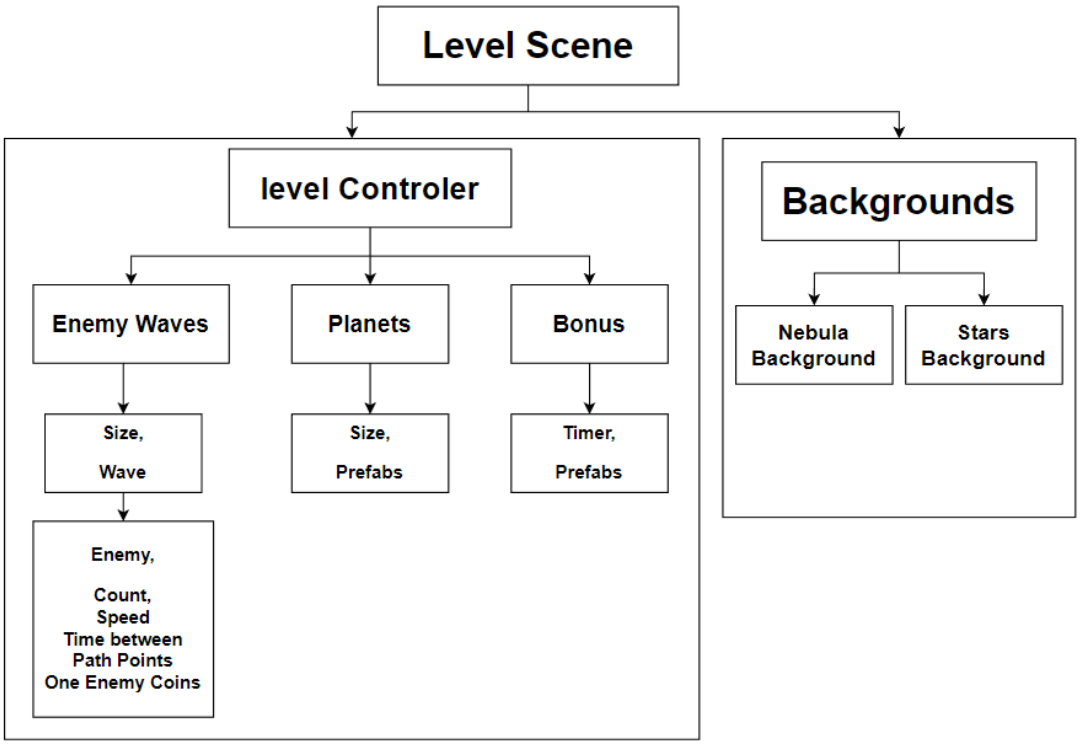

Рисунок 2.3 – Метод високорівневого ускладнення гри

Метод реазації мобільної шутер гри «Space confrontation» візуалізує процес ігрової рівневой сцени «Level Scen», що містить у собі котролер рівня «level Controler» та фон гри «Background», що змінюватиметься в залежності від рівня.

Котролер рівня містить у собі «Enemy Waves», «Planets», «Bonus», кожна із яких має свої особливості. «Enemy Waves» – це ворожі хвилі, які містять у собі кількість хвиль та їх тип з певною кількістю ворожих юнитів, які рухатимуться по певній косій та противостоятимуть гравцю. Також з певним промішком часу ці хвилі звялятимуться поступово та зникатимуть, якщо гравець їх не знищив. Кожний ворожий тип юніта має певні характеристики: вигляд, життя, швидкість, отриманням гравцем певної кількості ігрової валюти за знищення юніта. «Planets» – містить у собі контроль планет, які будуть відображатися на задньому фоні як декоративний візуальний елемент гри, має такі характеристики: час проміжку появи та певна кількість прифабів, що будуть послідовно з’являться в ігровому процесі та зникати після його завершення. «Bonus» – додатковий функціонал для

полегшення проходження рівня гри, надаватиме можливість космічному крейсеру підвишення рівня, що призводитиме до апгрейду крейсера.

«Background» – фон гри, що буде відтворюватися в залежності від положення ігрового юніта та його переміщення в процесі гри, містить в собі «Nebula Background» та «Stars Background».

Таким чином, через «level Controler» можна з легкістью налаштувати нові типи ворожих юнитів та реалізовувати нові більш складні «Level Scen» на шляху гравцю та для візуального середовищя через «Background».

2.3 Розробка моделі внутрішнього ігрового магазину

У відеоіграх мікротранзакція, також відома як покупка в грі або в додатку, є транзакцією [14], при якій споживачі обмінюють реальну валюту на віртуальну валюту або товари, такі як унікальні персонажі, зброя, засоби пересування та вихованці, які можуть мати більш високі характеристики. статистика або лише косметичні відмінності.

Наприкінці 1990-х років, зі зростанням популярності Інтернету люди почали завантажувати контент відеоігор онлайн. Розробники відеоігор та любителі будуть продавати або ділитися розширеннями для комп'ютерних ігор як завантажуваний контент (DLC) [15]. У 2002 році Microsoft випустила Xbox Live, онлайн-платформу для Xbox, яка дозволяла гравцям купувати на ній DLC. Перша мікротранзакція, продана великим видавцем, була в 2006 році, коли Bethesda продала кінську броню The Elder Scrolls IV: Oblivion за 2,50 долара. Це було зроблено як експеримент, щоб перевірити реакцію ринку на DLC. Більшість гравців відреагували негативно, заявивши, що 2,50 долара за внутрішньоігровий косметичний предмет – це дуже багато. Незважаючи на негатив, кінська броня стала дев'ятим DLC, що найбільше продається в Облівіоні, і її все ще купували більше двох років після її випуску. Bethesda та інші ігрові студії стали більше використовувати мікротранзакції як додаткове джерело доходу. У 2008 році магазин iOS був запущений на Apple iPhone, і ігри в ньому використовували

мікротранзакції як основне джерело фінансування. За перші три роки після запуску програми для iOS принесли більше 3,6 млрд доларів доходу за більш ніж 15 млрд завантажень, і 80% цього доходу припадає на мобільні ігри. Намагаючись повторити успіх мікротранзакцій у мобільних іграх, розробники додали більше мікротранзакцій у комп'ютерні та консольні ігри.

Причини, через які люди, особливо діти, продовжують платити за мікротранзакції, закладені в людській психології. Було багато дискусій про мікротранзакції та їх вплив на дітей, а також про зусилля з регулювання та законодавства. Мікротранзакції найчастіше надаються через інтерфейс магазину, що настроюється, розміщений усередині додатка, для якого продаються товари. Apple надає платформу, яка називається «покупки в додатку», для ініціювання та обробки транзакцій. Платформа Google для того ж використання називається «білінгом у додатку», назвав більше з погляду розробника. Apple і Google отримують по 30 відсотків від усіх доходів, отриманих від мікротранзакцій, проданих через покупки додатків у відповідних магазинах додатків.

Деякі ігри дозволяють гравцям купувати предмети, які можна отримати звичайними способами, але в деяких іграх є предмети, які можна отримати лише за допомогою мікротранзакцій. Деякі розробники переконайтеся, що таким чином доступні лише косметичні предмети, щоб ігровий процес був чесним та збалансованим.

В розбробленій грі буде використовуватися тільки внутриігрова валюта, тому що гра не призначенна для заробітку, а як тренувальний додаток для очіб різного віку, своєї моторики та стратегічного мислення при проходженні гри. Внутриігровий магазин матиме за мету різнообразний спосіб проходження гри, щоб гравець міш за валюту що він отримуватиме в процесі гри витрачував в магазині, для купівлі більшь модифікованного космічного крейсеру для проходження з легкістью високорівневих юнитів та босів.

Модель реалізації мобільної шутер гри «Space confrontation» візуалізує підтримку ігровому космічному крейсеру через магазин, в магазин можна зайти

через головне меню. В сомому магазині буде нада можливість покупки нового космічного крейсера с модифікаціями які краші за попередній, кожний із яких матиме певні херектеристики: рівень життя, захисний шит, швидкість стрільби, вигляд зброї, візуальний вигляд ракет, швидкість ракет, візуальний вигляд крейсеру. На рисунку 2.4 наведено модель реалізації мережевої мобільної гри-тренажера «Biplares: LastFight».

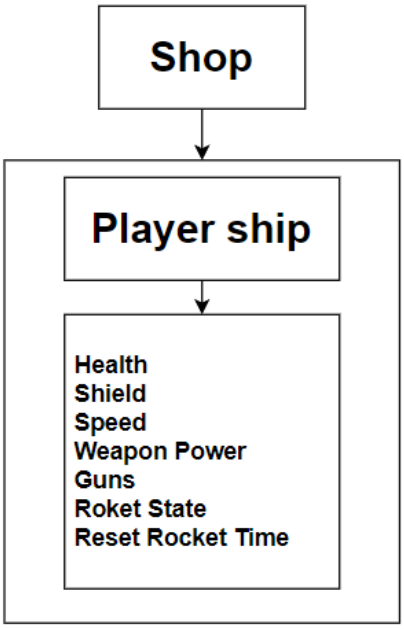

Рисунок 2.4 – Модель реалізації внутриігрового магазину

2.4 Розробка алгоритмів роботи мобільного ігрового додатку

Алгоритмів роботи мобільної шутер-гри «Space confrontation», який починається із завантаження головного меню, що надає вибір дій: «Почати грати», «Магазин», «Вихід» або «Налаштування».

Компонент «Почати грати» активує перехід на «Рівні ігри» та надає можливість обрати відкритий рівень проходження гри який здійснює перехід на ігрову сцену.

На ігровій сцені відбувається ігровий процес, де гравець притивостоють ворожим юнітам та повинен знишити їх всіх, та дібратися до кінця. Після

завершення битви активується перехід до сцени «Результат гри», де можна продовжити наступний рівне, чи повернутися в до «Меню».

Компонент меню «Налаштування» надає можливість регулювання музичного супровіда гри.

Компонент «Вихід» здійснює завершення роботи мобільного додатку.

Блок-схема загального алгоритму роботи мобільної шутер гри «Space confrontation» зображенна на рисунку 2.5.

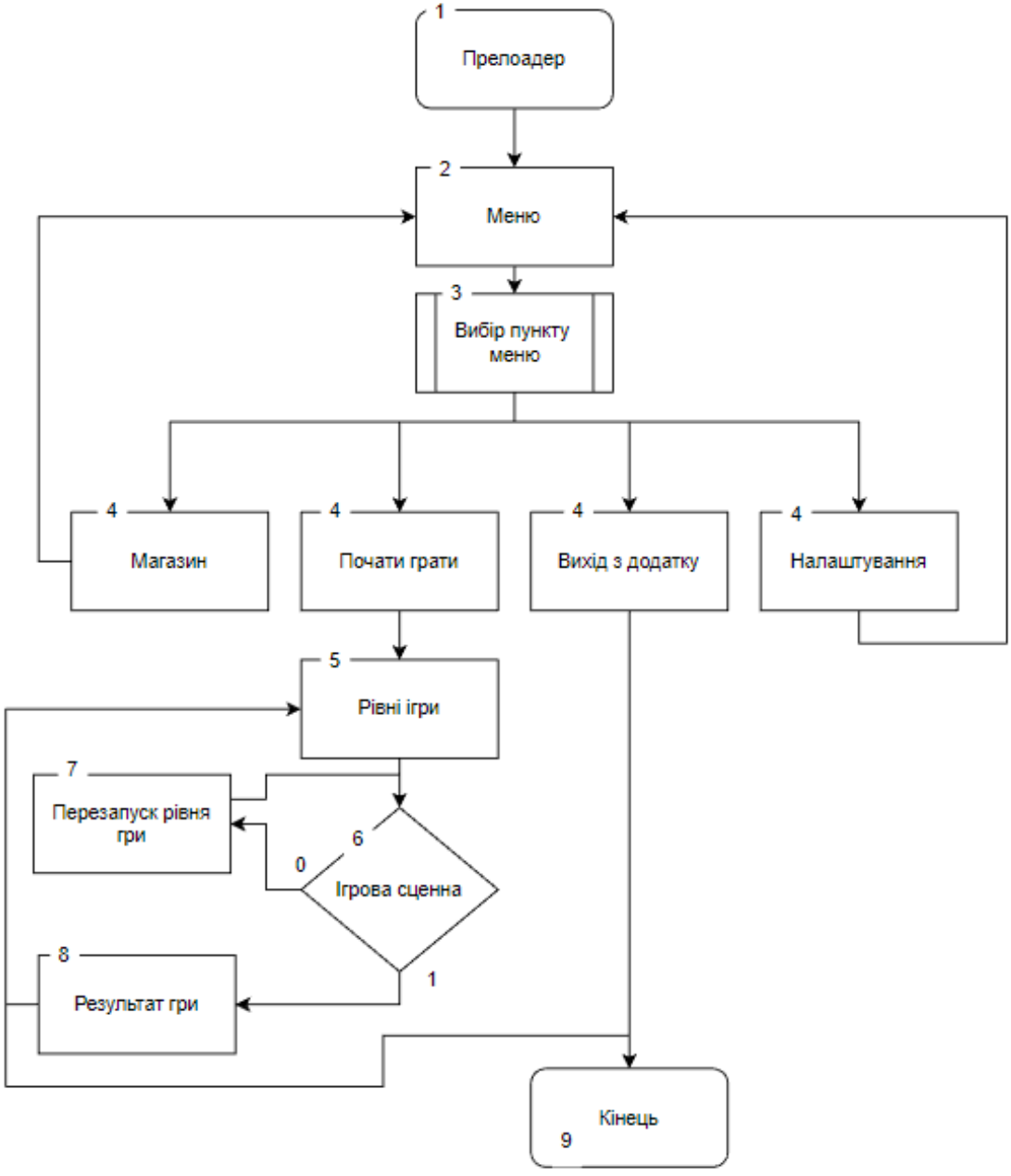

Рисунок 2.5 – Блок-схема загального алгоритму програми

2.5 Висновки

У другому розділі розглянуто структуру інтерфейсу ігрового додатку. Розроблено метод високорівневого ускладнення гри програмного продукту, який використовує різні шаблони побудовані розробником, та «Level Controler», за допомогою якого створюються нові «Level Scen». Також метод дозволяє підлаштовувати складність рівнів під конкретного гравця.

Розроблено модель роботи ігрової системи та модель реалізації внутрішнього ігрового магазину, орієнтовану на купівлю гравцем нового бойового космічного крейсеру для протистояння у важких рівнях проходження гри та для забезпечення різноманітного процесу гри. Розроблено алгоритм роботи мобільного ігрового додатку та описано процес його роботи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ІГРОВОГО ДОДАТКУ

3.1 Проєктування користувацького інтерфейсу

Користувальницькі інтерфейси складаються з одного або декількох шарів, включаючи інтерфейс людина-машина (HMI), який взаємодіє з машинами з фізичним вхідним обладнанням, таким як клавіатури, миші та вихідним обладнанням, таким як монітори комп'ютерів, динаміки, відеокамера та принтери. Додаткові шари інтерфейсу можуть взаємодіяти з одним або кількома органами чуття людини, включаючи: тактильний інтерфейс (дотик), візуальний інтерфейс (зір), слуховий інтерфейс (звук), нюховий інтерфейс (запах), рівноважний інтерфейс (баланс) та смаковий інтерфейс (смак).

Розрізняють такі види інтерфейсів користувача [16]:

– Графічний користувальницький інтерфейс (GUI) – це сукупність програмних програм, які використовують графічні можливості комп'ютера, щоб зробити програми простими у використанні. GUI використовує вказівник, вказівний пристрій та елементи інтерфейсу для управління комп'ютером та програмним забезпеченням. Вказівник – символ на екрані, який використовується для вибору піктограм та виконання завдань. Вказівний пристрій переміщує вказівник і може бути будь-яким типом пристроїв, включаючи клавіші зі стрілками на клавіатурі, миші або дотику. Елементи інтерфейсу – це піктограми, меню та вікна, які користувач вибирає для запуску програм, відкриття файлів та виконання команд.

– Інтерфейс командного рядка (ІКР) представляє собою оригінальний стиль взаємодії людини з машиною. Користувачі набирають запити або вказують дії формальною мовою, яка має власний словник, значення і синтаксис. Як правило, це набір команд, основних інструкцій до ОС. Будь- яка програма може використовувати ІКР для власного продукту. ІКР в найменшому ступені підтримує користувачів. Одна з основних проблем ІКР полягає в тому, що він не захищає ні ОС, ні програму від користувача. Для роботи з системою користувачам

потрібно знати, як комп'ютер працює і де знаходяться їх програми і дані. Модель, застосовувана в ІКР, є моделлю програміста, а не користувача. Для багатьох засвоєння ІКР схоже на вивчення іноземної мови. Комп'ютерне ПЗ та АЗ має власну мову, знайому розробникам, але комп'ютери повинні спілкуватись з користувачами мовою, доступною не лише програмістам.

Розробляючи мобільну шутер гру «Space confrontation», для забезпечення діалогу між програмою та користувачем, для програмного ігрового додатку, вирішено використати найпоширеніший тип інтерфейсу користувача – GUI, що надає можливість реалізувати всі принципи інтерфейсубуло розробленну структуру інтерфейсів додатку, що складатиметься з декількох частин.

Інтерфейс головного меню, що призначений для переходу в інші інтерфейси ігрового додатку, та под меню налаштування музики в ігрі.

Графічна схема головного меню зображена на (рис. 3.1).

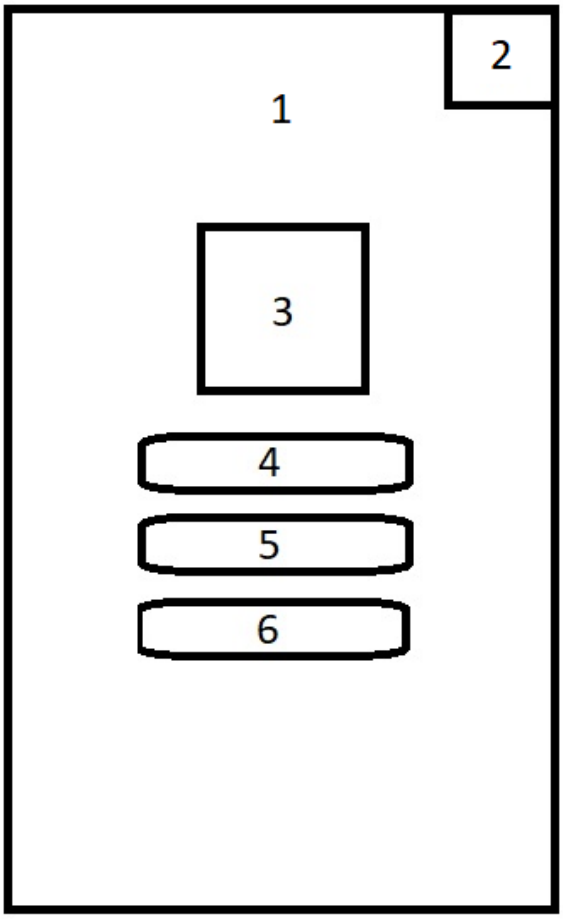

Рисунок 3.1 – Графічна схема інтерфейсу головного меню

Елементи головно меню:

1. Задній фон;
2. Кнопка налаштування;
3. Логотип ігрового додатку;
4. Кнопка переходу до інтерфейсу ігрових рівнів важкості;
5. Кнопка переходу до інтерфейсу внутриігрового мазину;
6. Кнопка виходу з ігрового додатку.

Інтерфейс ігрових рівнів важкості, що призначений для переходу в ігровий простір гравця.

Графічна схема ігрових рівнів важкості зображена на (рис. 3.2).

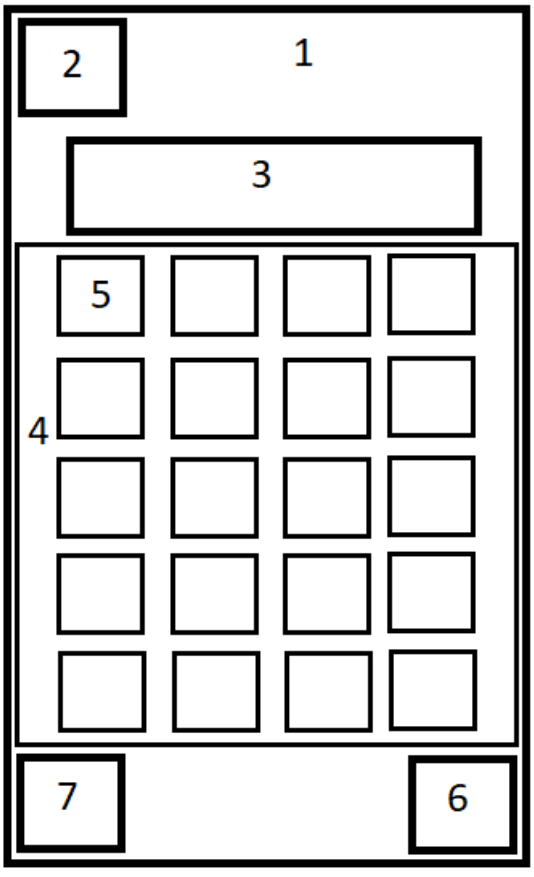

Рисунок 3.2 – Графічна схема інтерфейсу ігрових рівнів важкості

Елементи ігрових рівнів важкості:

1. Задній фон;
2. Кнопка повернення до головного меню;
3. Текст раси ворожих юнитів сюжетного рівня важкості;
4. Область кількості рівнів в даній важкості;
5. Кнопка 1 рівня ігрового простору гравця, та інших які відкриються при проходженні попереднього рівня;
6. Кнопка переходу до наступної раси ворожих юнитів та їх рівні важкості проходження;
7. Кнопка повернення до попередньої раси ворожих юнитів.

Інтерфейс ігрового простору гравця, що являється головним процесом ігрового простору де гравець проводитиме більше усього часу.

Графічна схема ігрового простору гравця зображена на (рис. 3.3).

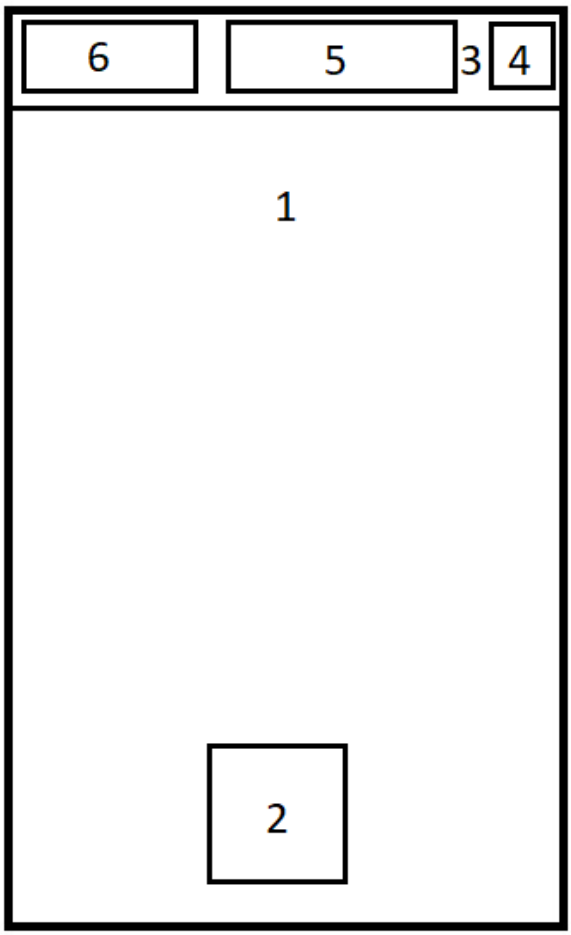

Рисунок 3.3 – Графічна схема інтерфейсу ігрового простору гравця

Елементи ігрового простору гравця:

1. задній фон, та область яка сприйметиме сенсорне переміщення крейсера гравця, і де відбуватиметься сам процес гри;
2. зображення космічного крейсеру гравця;
3. робоча область;
4. кнопка випадаючого меню, що має в собі регулювання звуку гри, кнопки повернення до головного меню, перезапуск гри, та продовження гри, так як при відкритому випадаючому меню ігровий процес стоятиме на паузі;
5. графічне зображення життя та броні ігрового юніта;
6. текстове зображення часу гри раунда та кількості очків, набраних за знищення ворожого юніта.

Інтерфейс внутрішнього ігрового магазину, що призначений для купівлі на інший бойовий космічний крейсер, більш модифікований для супротиву з більш важкими юнітами на високих рівнях важкості гри. Придбати засоби покращення юніта можна за ігрову валюту, яка отримуватиметься та накопичуватиметься від навиків гравця в ігровому процесі.

Графічна схема внутрішнього ігрового магазину зображена на рисунку 3.4.

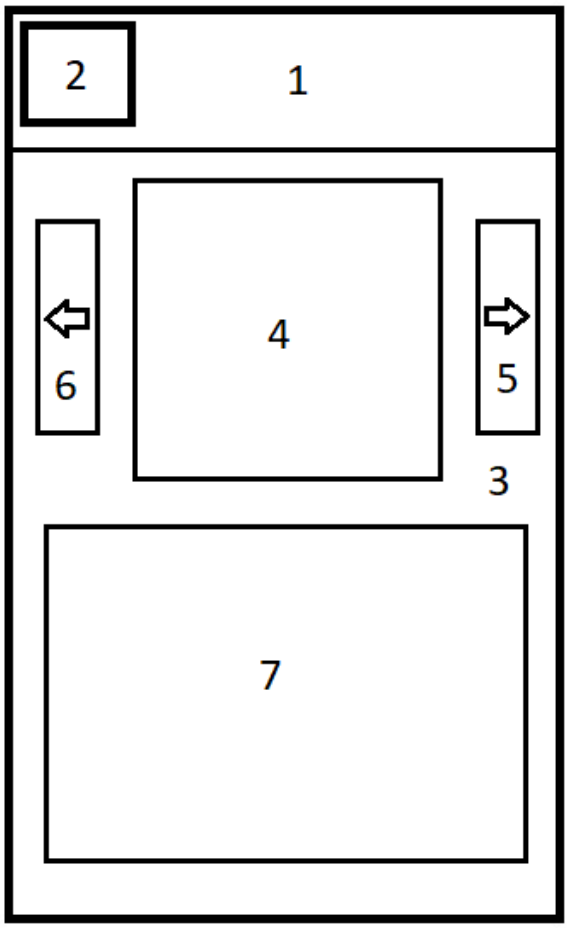

Рисунок 3.4 – Графічна схема інтерфейсу внутрішнього ігрового магазину

Елементи внутрішнього ігрового магазину:

1. задній фон;
2. кнопка повернення до головного меню;
3. область магазину;
4. зображення ігрового космічного крейсера;
5. перегляд наступного ігрового космічного крейсера;
6. повернення до минулого ігрового космічного крейсера;
7. харектеристики обраного ігрового космічного крейсера.

3.2 Розробка графічних та медіа матеріалів

Розібравшись і збудувавши план роботи за технічним завданням, пристумо до роботи над 2D-графікою для гри. Спочатку все починається зі скетчів – це начерки елементів гри, персонажів, ігрових сцен, озброєння, в загальному все, що

буде в проекті. Це робиться для того, щоб було простіше бракувати або редагувати ідеї, оскільки моделювання всього перерахованого вище зайняло б набагато більше часу.

У іграх важливими аспектами є як візуальні аспекти інтерфейсу та зображень, так і музичного супровіду, тому для розробки візуальних компонентів було обранно програму Adobe Photoshop, а музикальний супровід буде знайде в просторах інтернету.

Растрове зображення – зображення, що є сіткою (мозаїкою) пікселів – кольорових точок (зазвичай прямокутних) на моніторі, папері та інших пристроях, що відображають [17].

Растровий графічний редактор – спеціалізована програма, призначена для створення та обробки растрових зображень, тобто графіки, яка на згадку про комп'ютер записується як набір точок, а не як сукупність формул геометричних фігур [18].

Подібні програмні продукти знайшли широке застосування в обробці цифрових фотографій та застосовуються в роботі художників-ілюстраторів, під час підготовки зображень до друку друкарським способом або на фотопапері, публікації в інтернеті.

Adobe Photoshop – багатофункціональний графічний редактор , що розробляється і розповсюджується компанією Adobe Systems [19]. В основному працює з растровими зображеннями, однак має деякі векторні інструменти. Продукт є лідером ринку в області комерційних засобів редагування растрових зображень та найвідомішою програмою розробника.

Розробка сцени буде складатися з побудови векторних та текстових об’єктів. Першим було створено фонове зображення, яке складається з групи векторних об’єктів.

На рисунку 3.5 наведено візуалізацію процесу створеної групи фонових об’єктів.

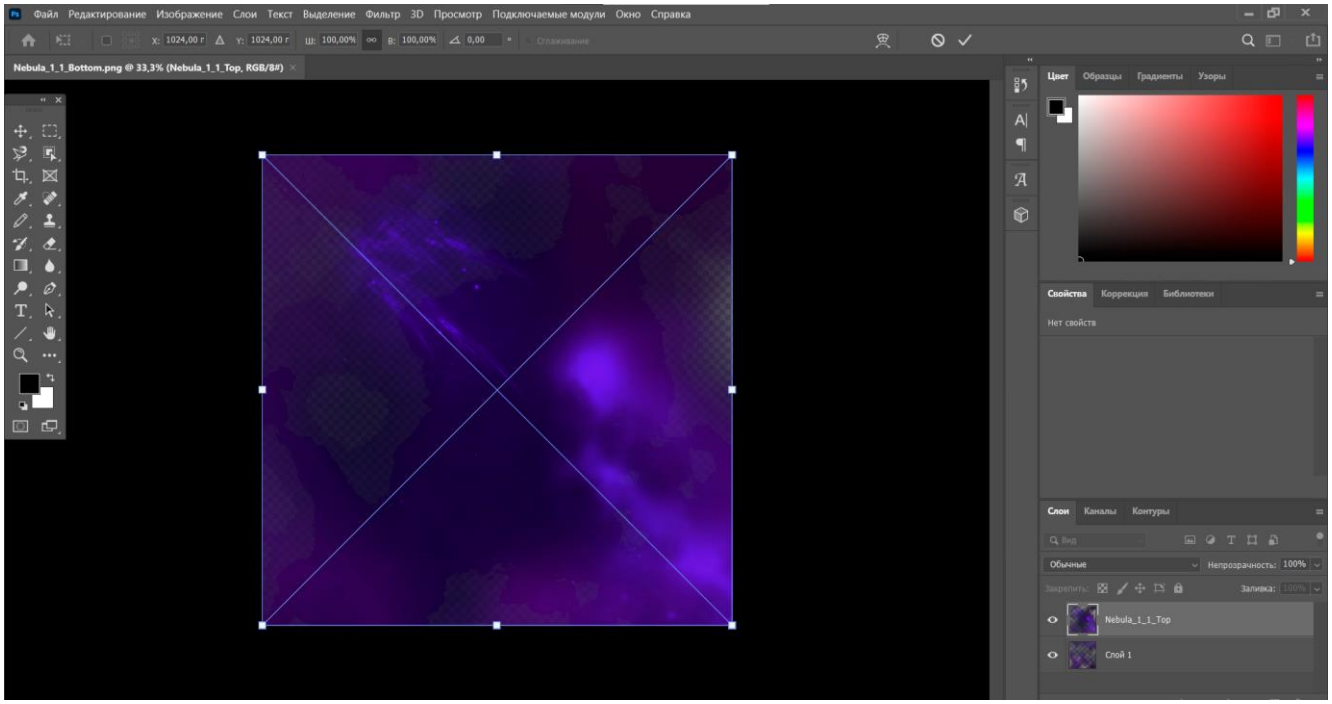

Рисунок 3.5 – Група векторних об’єктів, яка відображує фонове зображення

Якщо є необхідність, то розмір графічного зображеннч можна редагувати в будь-який момент без зайвих втрати якості зображення.

Використовуючи в Unity можливість відтворення простої фігури, для ігрової сцени, було створено пряму як користувацький інтерфейс, елемент, що відображує полосу рівня здоров’я головного космічного крейсера в зеленому кольорі та полосу захисного щита в синьому кольорі.

На рисунку 3.6 наведено вигляд елементу, який відображує рівень здоров’я та захисного поля.

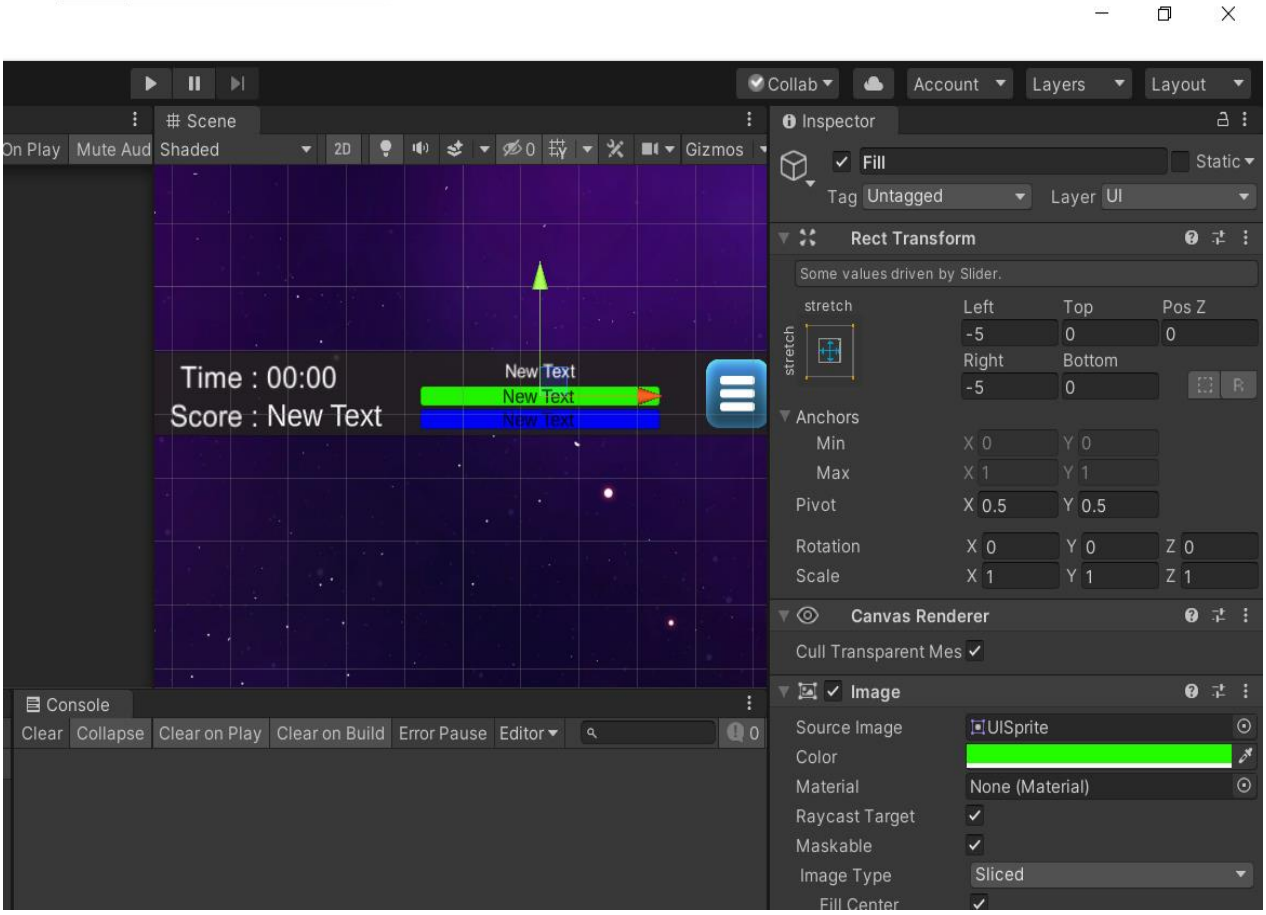

Рисунок 3.6 – Графічний елемент – «Полоса рівня здоров’я та захисного поля ігрового об’єкта».

Ігрові юніти також були розроблені в Adobe Photoshop, такі як ворожі космічні крейсири різних видів та ігрові крейсери, вистріли (як лазери, так і ракети), ілюстративний задній фон планет та туману.

На рисунку 3.7 наведено одного з типів крейсерів створеного об’єкту під назвою «Ворожий крейсер версія 2».

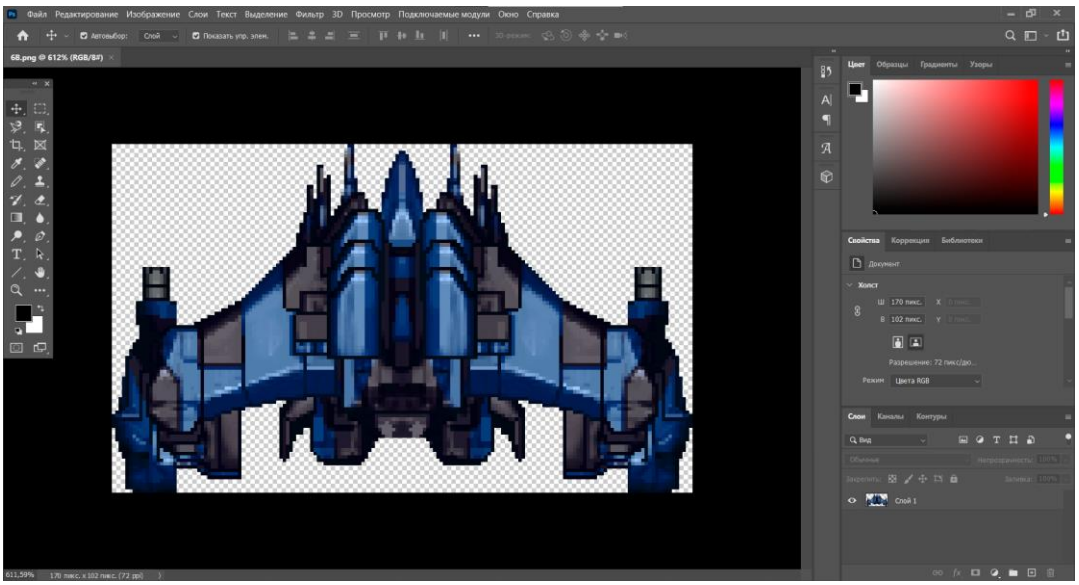

Рисунок 3.7 – Графічний елемент – об’єкт «Ворожий крейсер версія 2»

Після створення всіх растрових об’єктів, було розроблено декілько графічних анімацій за домогою іструментів Unity, яка надає можливість створювати їх за партікал системою. Ефекти мали відображати коли ігровий юніт попадає у ворожого його часткове руйнювання чи повноцінне знищення. Ці ефекти надають візуальне сприйняття гравцю що він попадає по юнітам чи по ньому саому попали На рисунку 3.8 наведено вигляд створеного графічної анімації «Вибух».

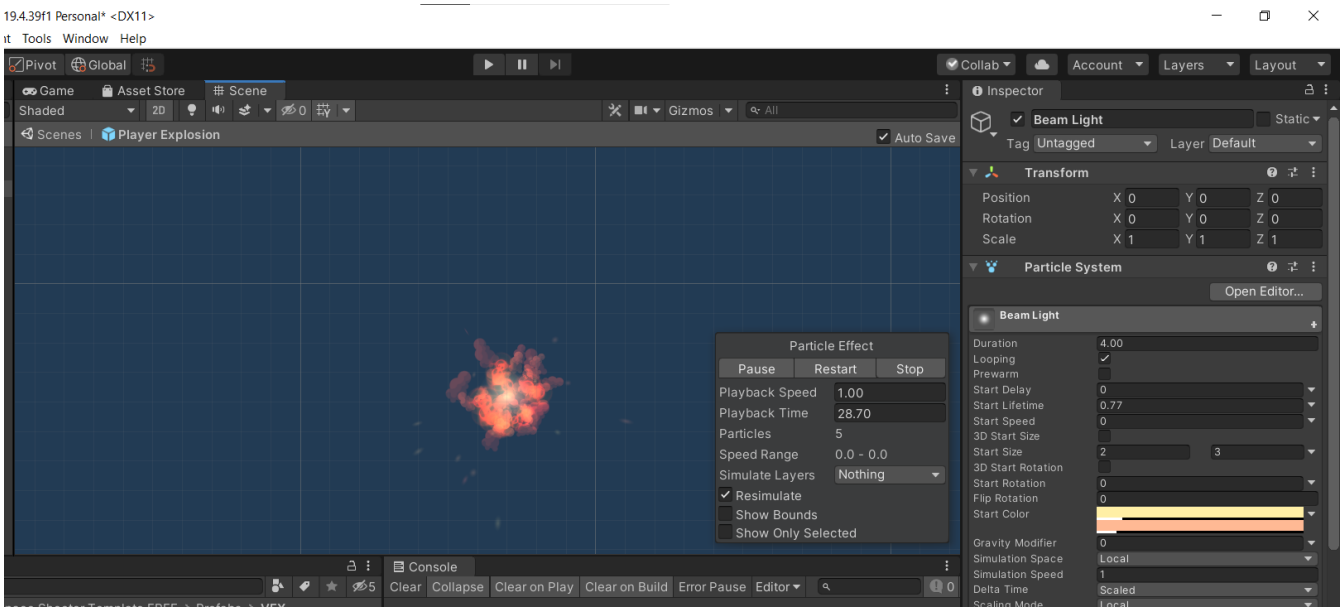

Рисунок 3.8 – Графічна анімація «Вибух»

Після розробки всіх об’єктів потрібно поєднати їх для відображення ігрової сцени. Зображення поєднаних графічних елементів наведено на рисунку 3.9.

Рисунок 3.9 – Поєднання розроблених графічних елементів та формування вигляду ігрової сцени

У кінці для майбутного покращення чи оновлення необхідно зберегти графічні елементи у вигляді, придатному для використання в ігровій системі. Unity підтримує такі графічні формати PNG, BMP, TIF, TGA, JPG, та PSD. Формат PNG дозволяє зберегти прозорість, тому саме він буде використаний у рамках розробки графічних елементів мобільної шутер гри «Space confrontation».

Музичний супровід гри розроблявся множиною кроків у програмному рушії Unity, який містить у собі інструмент «Audio Source», де було його створенно в певних сценах та створенно додатковий скріпт, завдяки якому можна було вносити в ігрову сцену музикальне супроводження. Реалізовний скрип подано в текстовому вигляді.

```
public class MenuGameScene : MonoBehaviour
{
   [Header("Audio Settings")]
   [SerializeField] private Slider slider;
   [SerializeField] private AudioSource soundController;
   [SerializeField] private AudioClip backgroundSound;

private void Awake()
   {
      slider.onValueChanged.AddListener(ChangeSoundVolume);
   }

private void Start()
   {
      soundController.clip = backgroundSound;
      soundController.Play();
   }

private void ChangeSoundVolume(float value)
   {
      soundController.volume = value;
   }
```

В ігровій сцені Unity налаштування музичного супровіду гри зображено на рисунку 3.10.

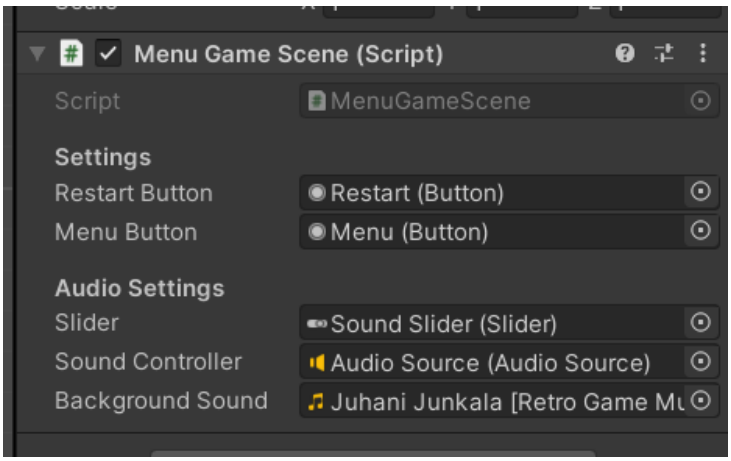

Рисунок 3.10 – Налаштування музичного супровіду в ігровій сцені Unity

3.3 Програмна реалізація методу високорівневого ускладнення гри

Для реалізації високоріневого ускладнення рівнів гри було створенно в програмному рушію Unity сцену під назвою «GameScene», в якій відтворювали задуманий метод в програмному вигляді через програмне середовище Visual Studio 2019, під назвою «LevelControler».

При завантаженні через інтерфейс рівнів важкості «LevelScene» зображений на рисунку 3.11.

Рисунок 3.11 – Інтерфейс рівнів важкості «LevelScene»

До основної ігрової сцени гравець автоматично створює інтерфейс ігрового сцени «GameScene» того рівня що було обрано.

Частина коду завантаження основної ігрової сцени наведена на рисунку 3.12.

```
public class LevelController : MonoBehaviour {

    //Serializable classes implements
    public EnemyWaves[] enemyWaves;

    [SerializeField] private int CurrentLevel;

    public GameObject powerUp;
    public float timeForNewPowerup;
    public GameObject[] planets;
    public float timeBetweenPlanets;
    public float planetsSpeed;
    List<GameObject> planetsList = new List<GameObject>();

    Camera mainCamera;

    Сообщение Unity | Ссылок: 0
    private void Awake()
    {
        if (PlayerPrefs.HasKey(PrefsKey.levelKey))
        {
            CurrentLevel = PlayerPrefs.GetInt(PrefsKey.levelKey);
        }
    }
```

Рисунок 3.12 – Програмний код створення сцени ігрового простору «GameScene»

Процес запуску нової хвилі «Enemy Waves» та після певного проміжку часу відтворює наступну хвилю, а попередню знищує для полегшення навантаження на ігровий пристрій, програмний код наведено на рисунку 3.13.

```
Сообщение Unity | Ссылок: 0
private void Start()
{
    mainCamera = Camera.main;
    //for each element in 'enemyWaves' array creating coroutine which generates the wave
    for (int i = 0; i<enemyWaves.Length; i++)
    {
        StartCoroutine(CreateEnemyWave(enemyWaves[i].timeToStart, enemyWaves[i].wave));
    }
    StartCoroutine(PowerupBonusCreation());
    StartCoroutine(PlanetsCreation());
}

//Create a new wave after a delay
ссылка: 1
IEnumerator CreateEnemyWave(float delay, GameObject Wave)
{
    if (delay != 0)
        yield return new WaitForSeconds(delay);
    if (Player.instance != null)
        Instantiate(Wave);
}
```

Рисунок 3.13 – Програмний код запуску нової хвилі «Enemy Waves»

Частина скрипта генерації ворожих хвиль, зображена на рисунку 3.14. Під генерацією хвиль розуміється, що є певні характеристики, наприклад «Wave» визначає, скільки ворогів буде з'являтися, їх швидкість і інтервал відтворення між ними. Він також визначає їх режим зйомки. Це визначає їх шлях руху.

```
public class Wave : MonoBehaviour {

    FIELDS

    Сообщение Unity | Ссылок: 0
    private void Start()
    {
        StartCoroutine(CreateEnemyWave());
    }

    Ссылок: 2
    IEnumerator CreateEnemyWave() //depending on chosed parameters generating enemies and defining their parameters
    {
        for (int i = 0; i < count; i++)
        {
            GameObject newEnemy;
            newEnemy = Instantiate(enemy, enemy.transform.position, Quaternion.identity);
            FollowThePath followComponent = newEnemy.GetComponent<FollowThePath>();
            followComponent.path = pathPoints;
            followComponent.speed = speed;
            followComponent.rotationByPath = rotationByPath;
            followComponent.loop = Loop;
            followComponent.SetPath();
            Enemy enemyComponent = newEnemy.GetComponent<Enemy>();
            enemyComponent.shotChance = shooting.shotChance;
            enemyComponent.shotTimeMin = shooting.shotTimeMin;
            enemyComponent.shotTimeMax = shooting.shotTimeMax;
            newEnemy.SetActive(true);
            yield return new WaitForSeconds(timeBetween);
        }
        if (testMode)       //if testMode is activated, waiting for 3 sec and re-generating the wave
        {
```

Рисунок 3.14 – Програмний код генерації ворожих хвиль

Для полегшення проходження були створенні допоміжні об’єкти, що мали властивість підняти рівень космічного крейсери, що призводило до апгрейду його озброєння, на рисунку 3.15 подано код, який генерує бонус «levelUp».

```
//endless coroutine generating 'levelUp' bonuses.
ссылка: 1
IEnumerator PowerupBonusCreation()
{
    while (true)
    {
        yield return new WaitForSeconds(timeForNewPowerup);
        Instantiate(
            powerUp,
            //Set the position for the new bonus: for X-axis - random position between the borders of 'Player's' movement; for Y-axis - right above the
            new Vector2(
                Random.Range(PlayerMoving.instance.borders.minX, PlayerMoving.instance.borders.maxX),
                mainCamera.ViewportToWorldPoint(Vector2.up).y + powerUp.GetComponent<Renderer>().bounds.size.y / 2),
            Quaternion.identity
            );
    }
}
```

Рисунок 3.15 – Програмний код генерування бонус «levelUp»

Для візуального простору, де гравець знаходиться в космічному просторі, недостатньо тільки фону космосу та зірок, тому було реалізовано фон, який

змінюється вниз, що дає уявлення, що гравець рухається вгору, так як це локація космосу і об’єктами будуть планети.

На рисунку 3.16 наведено код реалізації руху планет, керування їх кількістю та часом проміжку появи в ігровому просторі.

```
ссылка: 1
IEnumerator PlanetsCreation()
{
    //Create a new list copying the arrey
    for (int i = 0; i < planets.Length; i++)
    {
        planetsList.Add(planets[i]);
    }
    yield return new WaitForSeconds(10);
    while (true)
    {
        ////choose random object from the list, generate and delete it
        int randomIndex = Random.Range(0, planetsList.Count);
        GameObject newPlanet = Instantiate(planetsList[randomIndex]);
        planetsList.RemoveAt(randomIndex);
        //if the list decreased to zero, reinstall it
        if (planetsList.Count == 0)
        {
            for (int i = 0; i < planets.Length; i++)
            {
                planetsList.Add(planets[i]);
            }
        }
        newPlanet.GetComponent<DirectMoving>().speed = planetsSpeed;

        yield return new WaitForSeconds(timeBetweenPlanets);
    }
}
```

Рисунок 3.16 – Програмний код генерування планет

Після демонстрації частин коду та їх опис можливостей буде продемонстровано результат роботи, на рисунку 3.17 відображено ігровий процес сцени «GameScene».

Рисунок 3.17 – Ігровий процес

На рисунку 3.18 продемонстровано процес появи нової ворожої хвилі «Enemy Waves» в ігровій сцені.

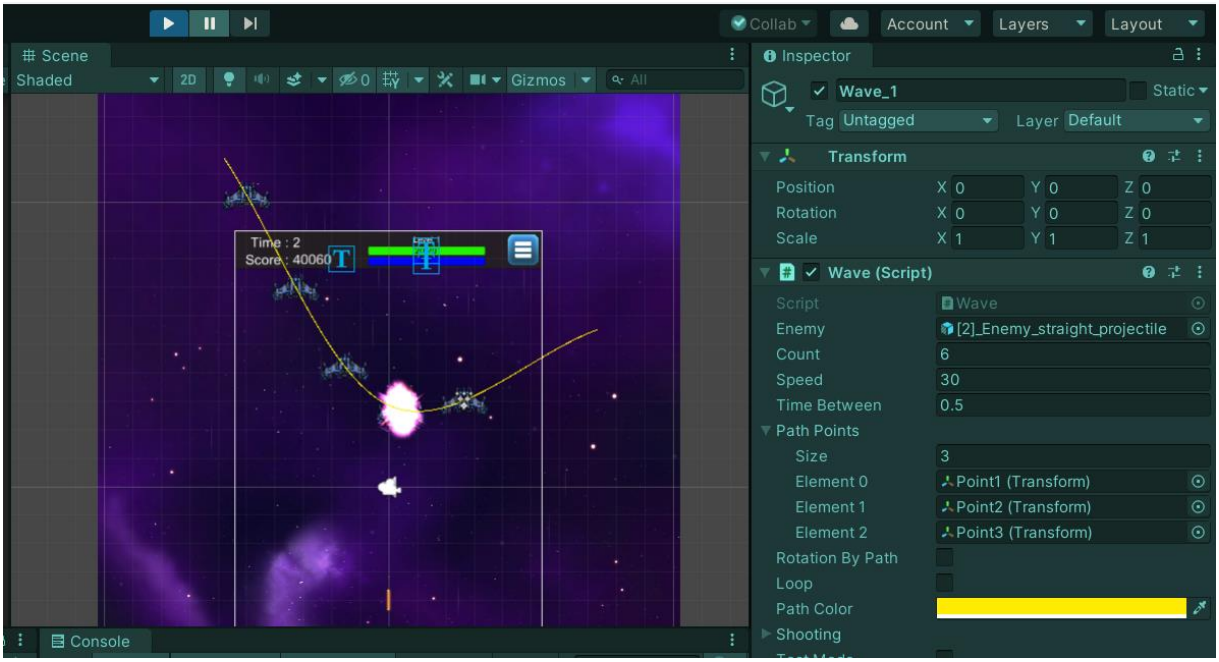

Рисунок 3.18 – Процес появи нової ворожої хвилі «Enemy Waves»

На рисунку 3.19, представлено ігровий об’єкт бонус «levelUp», та його сприйняття на ігрового юніта користувача, як видно кількість випущених лазерних променів є більше чим один, так як на початку гри тільки один промін, це надає гравцю полегшення проходження гри.

Рисунок 3.19 – Представлено ігровий об’єкт - бонус «levelUp»

На рисунку 3.20 продемонстровано дві із чотирьох планет, що відіграють ролі візуального представлення переміщення гравця у гору, так як планети, що появляються рухаючись вниз. Також всі елементи, що виходять за рамки камери, які рухаються за гравцем зникають, для полегшення навантаження на систему пристрою в якому відбуватиметься процес гри.

Рисунок 3.20 – Представлено ігрові об’єкти планети

Реалізований програмний код скрипта під назвою «LevelControler», зображений на рисунку 3.21, який представлений у вигляді програмного рушія Unity в компонентах де можна керувати процесом котролем рівня [20].

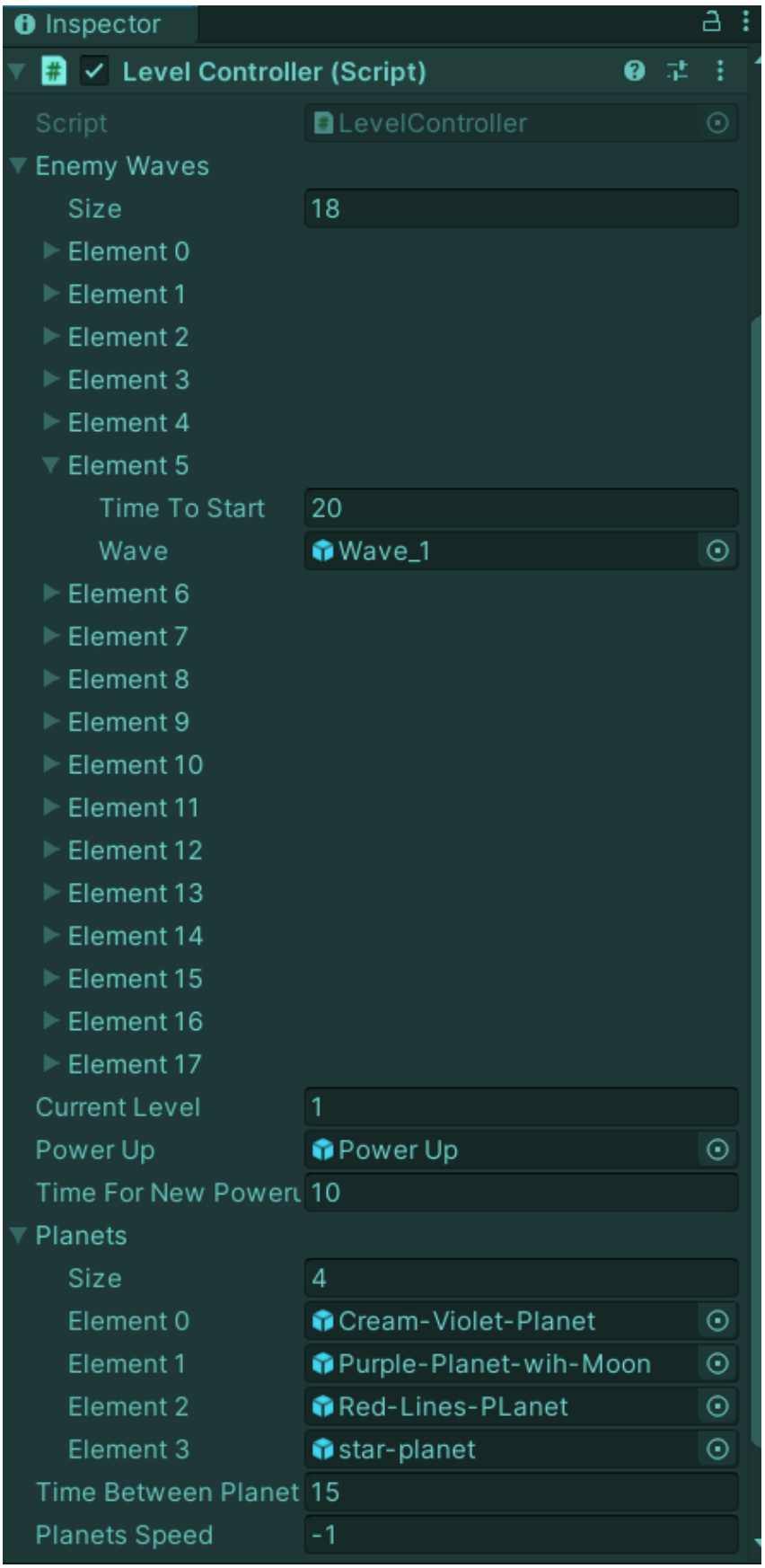

Рисунок 3.21 – Програмний компонент «LevelControler» в ігровій сцені Unity

3.4 Програмна реалізація моделей внутрішнього ігрового магазину

Для цікавішого проходження гри буде реалізовано внутрішній ігровий магазин. У магазин можна потрапити через інтерфейс головного меню. Реалізовуючи дану модель в програмному рушію Unity, було створено ігрову сцену під назвою «ShopScene», у якому розподіллено програмний код на такі складові частини «CoinManager», «DataInfo», «SelectButton», «ShopManager», кожна з яких відповідає своєму функціоналу реалізованого внутриігрового магазину.

«CoinManager» – відповідає за грошову валюту гри, дану валюту можна отримати знищуючи ворожих юнитів, за окремий вид дається певна кількість монет. В ігровій сцені «ShopScene» валюту можна витрати на купівлю більш модифікованих космічні крейсери, для проходження високих рівнів та босів. Даний скрип надає можливість відслідковувати вірну кількість монет гравця при купівлі нових крейсерів, частина коду зображена на рисунку 3.22.

```
using UnityEngine.Events;

[RequireComponent(typeof(TextMeshProUGUI))]
Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 2
public class CoinManager : MonoBehaviour
{
    [SerializeField] private int coins;
    [SerializeField] private UnityEvent dontEnough;
    private TextMeshProUGUI coinText;

    Сообщение Unity | Ссылок: 0
    private void Awake()
    {
        if (coinText == null) coinText = GetComponent<TextMeshProUGUI>();
    }

    Сообщение Unity | Ссылок: 0
    private void Start()
    {
        ResetText();
    }

    Ссылок: 3
    private void ResetText()
    {
        coins = PlayerPrefs.GetInt(PrefsKey.coinKey, 0);
        coinText.text = coins.ToString();
    }

    Ссылок: 0
    public void AddCoins(int count)
    {
        coins += count;
        PlayerPrefs.SetInt(PrefsKey.coinKey, coins);
        ResetText();
```

Рисунок 3.22 – Програмний код відслідковування ігрової валюти

«DataInfo» – головний скрип що відповідає за інформацію, завдяки якому можна буде створити безліч ігрових космічних крейсерів з різними характеристиками, для більш цікавого гемплея для гравця. Характеристики кожного крейсеру матимуть особливості в таких функціоналах як:

– унікальний ID крейсера;
– унікальне зображення крейсера;
– назва крейсера;
– кількість життя;
– кількість захисного слою крейсера;
– рівень атаки;
– рівень швидкості вистрілів;
– ціна крейсера;
– придбаний гравцем на даний момент крейсер чи ні.

Частина коду зображена на рисунку 3.23.

```
22      public ShipInfo GetShipByID(int id)
23      {
24          ShipInfo ship = allShips.Find(x => x.ID == id);
25          if (!ship.isBuyed)
26              ship.isBuyed = PlayerPrefs.HasKey("Byed" + ship.ID);
27
28          return ship;
29      }
30
        ссылка: 1
31      public void IsBuyed(int id)
32      {
33          ShipInfo ship = allShips.Find(x => x.ID == id);
34          PlayerPrefs.SetInt("Byed" + ship.ID, ship.Price);
35      }
36
37      [System.Serializable]
        Ссылок: 6
38      public class ShipInfo
39      {
40          public int ID;
41          public Sprite Image;
42          public string Name;
43          public int Healthy;
44          public int Shield;
45          public int AttackDamage;
46          public int AttackSpeed;
47          public int Price;
48          public bool isBuyed;
```

Рисунок 3.23 – Програмний код інформації унікального ігрового юніта

Реалізований програмний код скрипта під назвою «DataInfo», зображений на рисунку 3.24, який представлений компонента, де можна керувати процесом створення нового космічного крейсеру зі своїми особливостями.

Рисунок 3.24 – Програмний компонент «DataInfo» в ігровій сцені Unity

«SelectButton» – даний скрип відповідає за перевірку та замінну тексту на кнопці придбати, обрати та обрано даний космічний крейсер. Придбати можна при умовах коли гравець має певну кількість ігрової валюти для купівля крейсеру тоді в кнопці текст замінюється на обрати. Обрати дозволено після придбання крейсеру, після він буде керуватися гравцем в ігровій сцені «GameScene», та напис замінна на обрано. Частина програмного коду зображена на рисунку 3.25.

```
44      public void SetSelectedSettigs()
45      {
46          isSelected = PlayerPrefs.GetInt(PrefsKey.selectShipKey) == ID;
47
48          if (isSelected)
49          {
50              text.text = "SELECTED";
51
52              DataInfo.ShipInfo shipInfo = data.GetShipByID(ID);
53
54              PlayerPrefs.SetInt(PrefsKey.shipIDKey, shipInfo.ID);
55              PlayerPrefs.SetInt(PrefsKey.shipShieldKey, shipInfo.Shield);
56              PlayerPrefs.SetInt(PrefsKey.shipHealthyKey, shipInfo.Healthy);
57              PlayerPrefs.SetInt(PrefsKey.shipDamageKey, shipInfo.AttackDamage);
58              PlayerPrefs.SetInt(PrefsKey.shipSpeedKey, shipInfo.AttackSpeed);
59          }
60          else
61          {
62              text.text = "SELECT";
63          }
64
65      }
66
        ссылка: 1
67      private void Chek()
68      {
69          if (!isBuyed)
70          {
71              if (coinManager.Subtract(price))
72              {
73                  SetShipID(ID);
74                  data.IsBuyed(ID);
75              }
76          }
77          else if (!isSelected)
78          {
79              PlayerPrefs.SetInt(PrefsKey.selectShipKey, ID);
80              SetSelectedSettigs();
81          }
82      }
83
        Ссылок: 2
84      private void ResetState()
85      {
86          isBuyed = false;
87          isSelected = false;
88      }
```

Рисунок 3.25 – Програмний код перевірку та замінну текста на кнопці

«ShopManager» – відіграє важливу роль для візуалізації магазину, і пов'язаний з минулими скриптами візуалізації їх функціоналу. За графічною схемою інтерфейсу внутрішнього ігрового магазину побудовано слайдер для зручності перегляду різних крейсерів в такому малому просторі та їх характеристики, та відображення кількісті ігровій валюти на даний момент прогресу отриманим гравцем. Програмний код зображена на рисунку 3.26.

```
public class ShopManager : MonoBehaviour
{
    [Header("Data base Info")]
    [SerializeField] private DataInfo data;
    [SerializeField] private DataInfo.ShipInfo current;
    private int selectShipID = 0;
    private int curentShipID = 0;
    [Space]
    [Header("In Top Panel")]
    [SerializeField] private CoinManager coin;
    [SerializeField] private Button back;
    [Space]
    [Header("In Center Panel")]
    [SerializeField] private Image shipImage;
    [SerializeField] private Button Left;
    [SerializeField] private Button Right;
    [Space]
    [Header("In Buttom Panel")]
    [SerializeField] private TextMeshProUGUI ShipNameText;
    [SerializeField] private TextMeshProUGUI HealthyText;
    [SerializeField] private TextMeshProUGUI ShieldText;
    [SerializeField] private TextMeshProUGUI AttackSpeedText;
    [SerializeField] private TextMeshProUGUI AttackDamageText;
    [SerializeField] private SelectButton selectButton;


    Сообщение Unity | Ссылок: 0
    private void Awake()
    {
        if (Left != null) Left.onClick.AddListener(LeftButton);
        if (Right != null) Right.onClick.AddListener(RightButton);
        if (back != null) back.onClick.AddListener(GoToBack);
    }

    Сообщение Unity | Ссылок: 0
    private void Start()
    {
        if (PlayerPrefs.HasKey(PrefsKey.selectShipKey))
        {
            selectShipID = PlayerPrefs.GetInt(PrefsKey.selectShipKey, 0);
            curentShipID = selectShipID;
        }
        SetShipSetting(curentShipID);
    }

    Ссылок: 3
    private void SetShipSetting(int ID)
    {
        current = data.GetShipByID(ID);
```

Рисунок 3.26 – Програмний код перевірку та замінну тексту на кнопці

Реалізований програмний код скрипта під назвою «ShopManager», зображений на рисунку 3.27, який поданий у вигляді компонента, де можна спостерігати за обраним на даний момент крейсером у магазині та головними налаштуваннями інтерфейсу, як текстового, так і візуального.

Рисунок 3.27 – Програмний компонет «ShopManager» в ігровій сцені Unity

Після демонстрації частин коду та їх опису можливостей буде продемонстровано результат роботи (на рисунку 3.28), де відображено внутрішній ігровий магазин.

Рисунок 3.28 – Представлено інтерфейс внутрішньогого ігрового магазину

3.5 Висновки

У третьому розділі було спроєктовано користувацький інтерфейс та описано, які будуть інтерфейси в мобільному програмному додатку. Для розробки графічних матеріалів було обрано Adobe Photoshop та пошук медіа матеріалів в інтернет середовищі.

Було проведено розробку методу високорівневого ускладнення гри та продемонстрованойого роботу в ігрових сцена Unity.

Реалізовано модель внутрішнього ігрового магазину, описано програмну структуру та продемонстровано кінцевий результат.

4 ТЕСТУВАННЯ ІГРОВОГО ДОДАТКУ

4.1 Вибір методів тестування програмного забезпечення

Тестування мобільних додатків – це процес, за допомогою якого прикладне ПЗ, розроблене для портативних мобільних пристроїв, перевіряється на його функціональність, зручність використання та сумісність [21]. Тестування може бути мануальним або автоматизованим.

Мобільне тестування підпадає до наступних категорій рисунок 4.1:

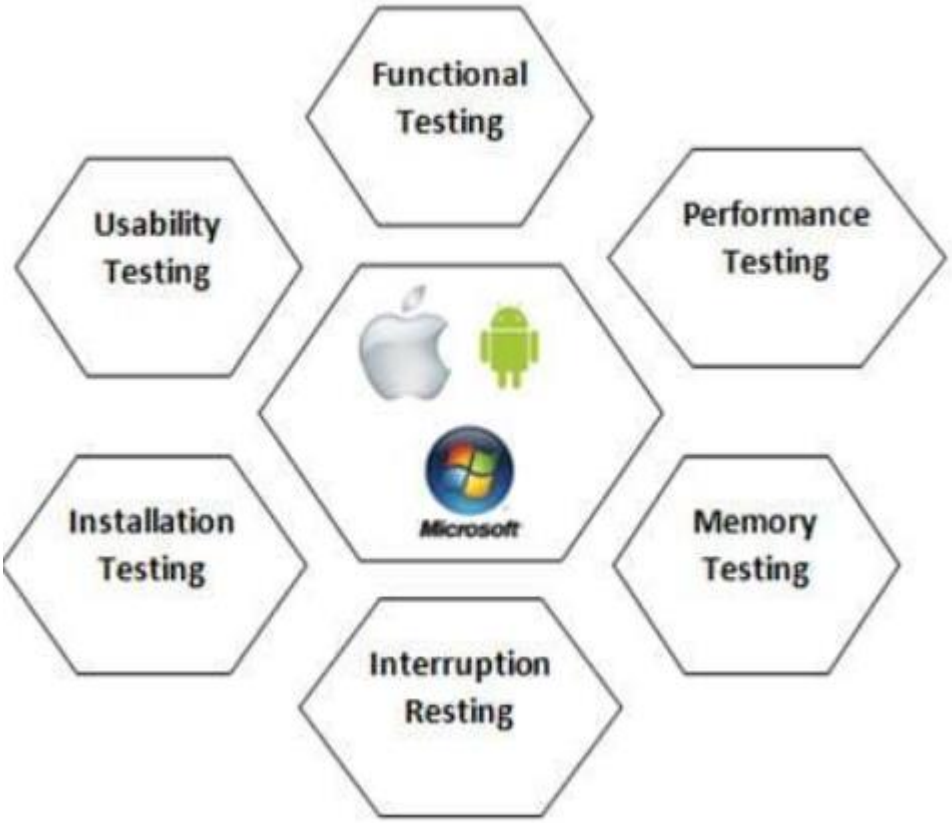

Рисунок 4.1 – Категорії мобільного тестування

Функціональне тестування: Основний тип тестування, що використовується для перевірки функціональних можливостей програми відповідно до специфікації вимог.

Тестування продуктивності: Виконується для тестування продуктивності клієнтських додатків, продуктивності сервера та продуктивності мережі.

Тестування пам'яті: Мобільні пристрої мають обмежену пам’ять порівняно з комп’ютерами, цей тип тестування проводиться для перевірки оптимізованого використання пам’яті додатком.

Тестування на переривання: [22] Використовується для перевірки на перебої через вхідний дзвінок або SMS, попередження про низький обсяг пам’яті, попередження про низький заряд акумулятора тощо під час запуску програми.

Тестування встановлення: Тестування встановлення використовується для перевірки простоти та плавності процесу встановлення, що включає також оновлення та видалення.

Тестування зручності використання: Як завжди, він перевіряв ефективність, результативність та задоволеність від застосування.

Тестування мобільних додатків може бути ручним або автоматизованим. Для цього використовується кілька мобільних засобів автоматизації тестів, не всі, але деякі з них перелічені нижче відповідно до популярності та використання.

Тестування мобільних додатків також є дуже важливим, оскільки клієнти вкладають мільйони у певний продукт, тож додаток з помилками та багами призведе до фінансових втрат, юридичних проблем та навіть може знищити імідж бренду.

Основна різниця між тестуванням мобільних додатків та десктопних програм. Кілька очевидних аспектів, які відрізняють тестування мобільних додатків від тестування програм для настільних ПК:

– десктопна програма тестується на центральному процесорі, мобільна на телефонах (Samsung, Nokia, Apple та HTC);

– розмір екрану мобільного пристрою менший, ніж екрану ПК;

– мобільні пристрої мають менше пам'яті;

– мобільні телефони використовують мережеві підключення, такі як 2G, 3G, 4G або WIFI, десктопні використовують широкосмугові або комутовані з'єднання;

– засоби для автоматизації, що використовується для тестування десктопних програм, можуть не працювати на мобільних додатках.

Типи тестування мобільних додатків. Для вирішення всіх вищезазначених технічних аспектів, проводяться такі типи тестування:

– тестування зручності використання – мобільний додаток повинен бути простий у використанні та забезпечувати задовільний рівень роботи для користувачів;

– тестування сумісності – тестування програми на різних мобільних пристроях, версіях операційних систем, у різних браузерах відповідно до вимог;

– тестування інтерфейсу – тестування опцій меню, кнопок, закладок, історії, налаштувань програми;

– тестування сервісів – тестування програми в режимі онлайн та офлайн;

– тестування ресурсів низького рівня – використання пам'яті, автоматичне видалення тимчасових файлів, проблеми розширення локальної бази даних;

– тестування продуктивності – тестування продуктивності програми шляхом зміни з'єднання з 2G, 3G на WIFI, спільного використання документів, споживання батареї тощо;

– тестування експлуатації – тестування резервних копій та плану відновлення, якщо акумулятор розряджається, або втрата даних під час оновлення програми з App Store/Play Market;

– тестування встановлення – перевірка програми шляхом її встановлення та видалення на пристроях;

– тестування безпеки – тестування програми для перевірки захисту інформаційної системи.

4.2 Тестування розробленого додатку

Початком тестування мобільного ігрового додатку є його запуск. Після запуску додатку очікується побачити завантажене головне меню. Результат запуску додатку наведено на рисунку 4.2.

Рисунок 4.2 – Головне меню гри

Після успішного запуску програми починається тестування функціоналу додатку. Для тестування методу високорівневого ускладнення гри в даному ігровому додатку, першим кроком є запуск першого рівня. Для цього потрібно натиснути відповідну кнопку на головному меню та перейти до екрану «рівня 1». Результат натискання кнопки наведено на рисунку 4.3.

Рисунок 4.3 – Результат натискання кнопки «рівня 1»

Після успішного запуску екрану «рівня 1», відображується процес гри, де гравцю, потрібно знищувати ворожі космічні крейсери для отримання ігрової валюти, щоб в наступних більш важких рівня проходити, за допомого новим придбаним космічним ігровим крейсером. При проходженні ігрового рівня є допоміжний елемент це підняття рівня, який призводить до додаткового вистрілу космічним бластером, для полегшеного проходження рівня. Результат покрокового проходження «рівня 1» зображено на рисунку 4.4.

Рисунок 4.4 – Результат покрокового проходження «рівня 1»

Якщо гравець втратить весь захисний бар’єр, який відображений в горі полосою синього кольору, та полосу життя зеленого кольору, тоді рівень гри буде не пройденим. Після програшу є можливість переграти рівень заново чи повернутися в меню рівнів. Інтерфейс результату програшу гравця зображений на рисунку 4.5.

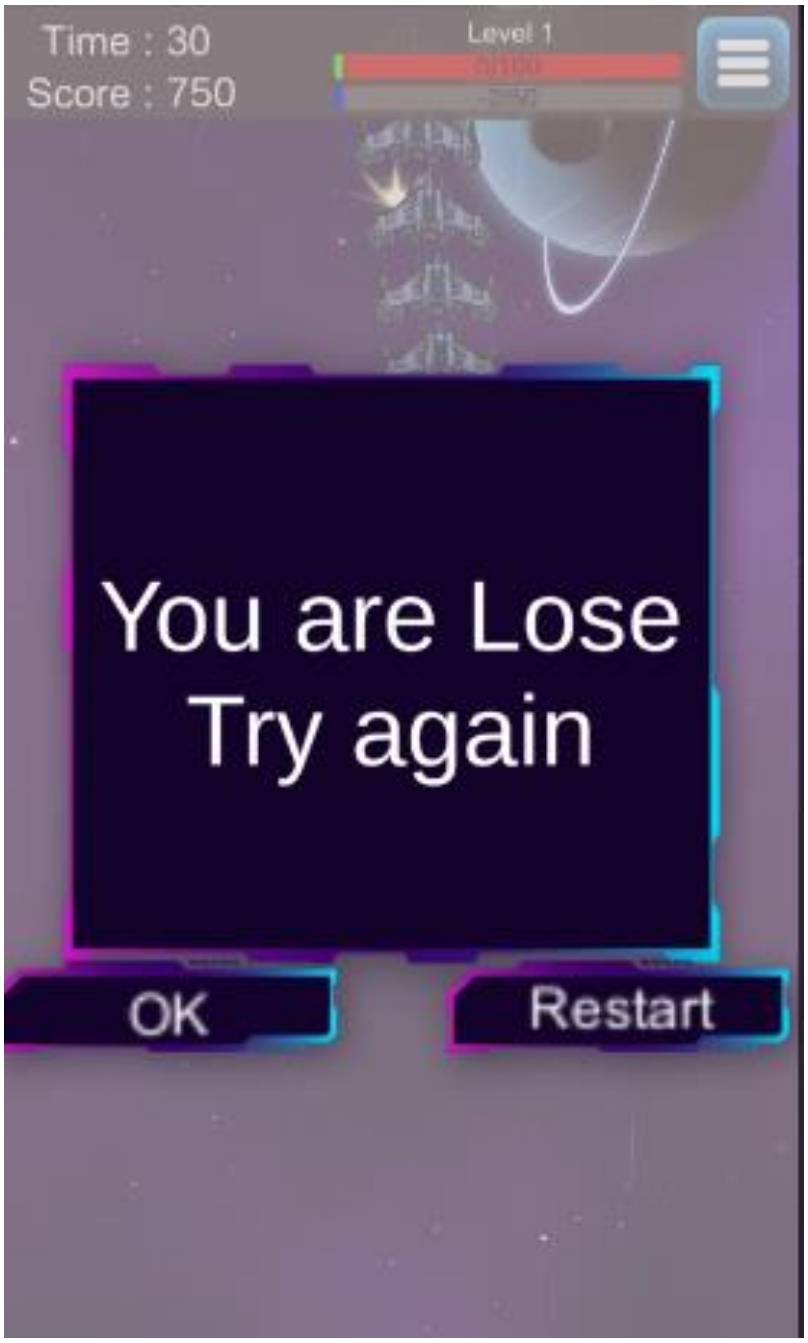

Рисунок 4.5 – Результат програшу гравця

У даній грі є 2 типи перемоги гравця в ігрових рівнях. Перша – це коли гравець проходить весь рівень збереженим космічним крейсером, немає різниці, які пошкодження він отримав на шляху, даний рівень не має босів. Другий спосіб – коли є рівень з босом, тут не достатньо просто вижити, також обов'язковою умовою перемоги є знищення ворожого боса. Інтерфейс результату перемоги гравця зображений на рисунку 4.6.

Рисунок 4.6 – Результат перемоги гравця

Після перемоги в інтерфейсі меню ігрових рівнів важкості можна помітити зірки над рівнем що пройшли. Кількість зірок відображатимуть рівень виконання усіх умов для проходження прогресу гри. Одну зірку можна отримати при проходженні рівня, коли полоса захисного бар'єру на 0, та полоса життя менше 50%. Дві зірки при проходженні, коли тільки захисний бар'єр має значення 0 а полоса життя більше 50%. Три зірки коли рівень життя 100% та захисний бар'єр матиме більше 0. Інтерфейс результату прогресу проходження гравця зображений на рисунку 4.7.

Рисунок 4.7 – Результату прогресу проходження гравця

Ігрова сцена з проходження рівня 2 де є для тесту провірялась робота боса була успішною. Бос мав відміну кількість життя від звичних ворожих юнитів також він не пролітає бойовий простір, а залишаються на місці, повторюючи певні траєкторії руху, перемогти можна тільки знищивши його. Інтерфейс результату проходження гравцем рівень з босом зображений на рисунку 4.8.

Рисунок 4.8 – Результату проходження гравцем рівень з босом

Під час ігрової сцени рівня гри можна поставити на паузу для зручності гравця у випадку його відволікання і щоб прогрес рівня гри не був даремною втратою часу. Така можливість відображує меню, де можна регулювати звук гри, кнопки продовжити гру, чи перезапустити даний рівень та перейти в головне меню. Інтерфейс результату паузи гри зображений на рисунку 4.9.

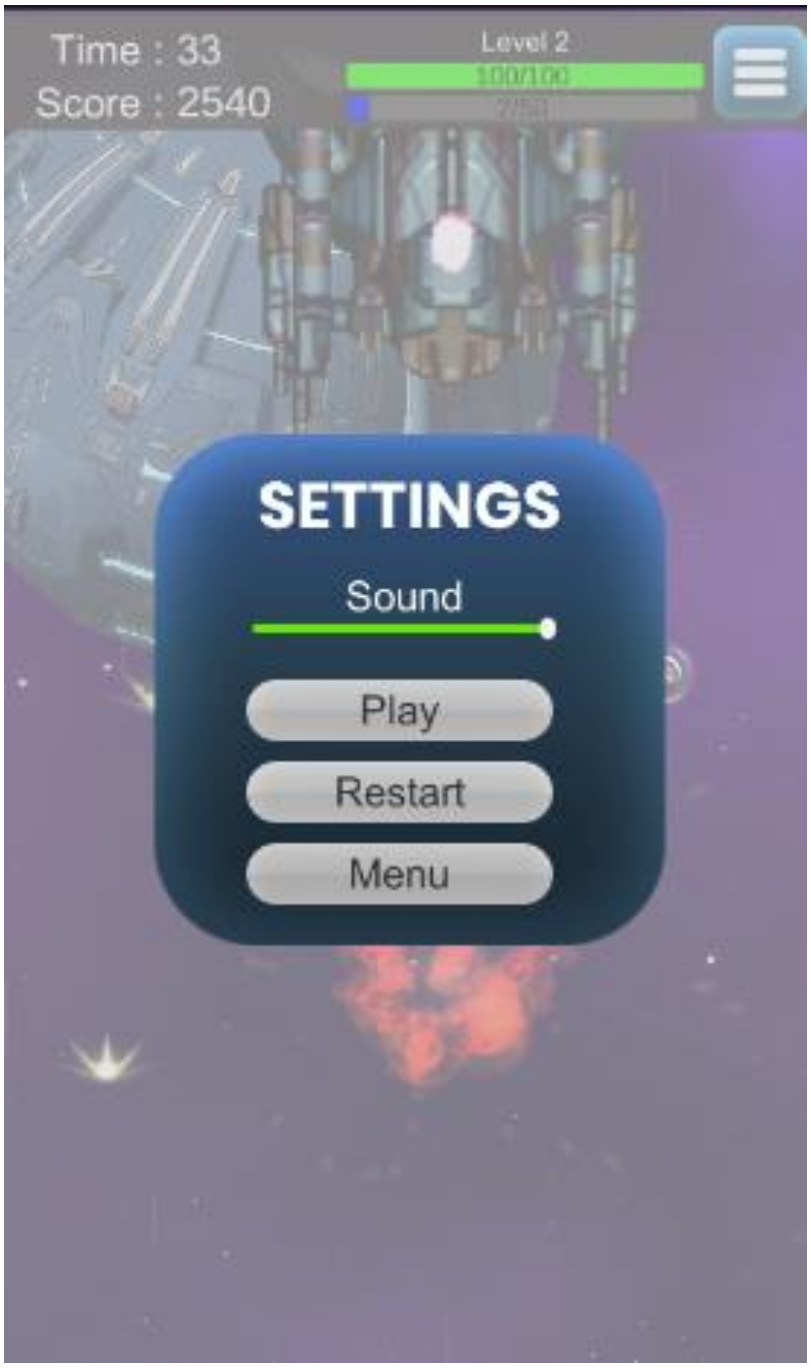

Рисунок 4.9 – Результату паузи гри

Після цікаво проведенного часу та накопиченої ігрової валюти можна придбати собі новий крейсер з новими параметрами, більш модифікованими, ніж у минулої версії крейсера. В залежності від параметрів крейсерів розподіленна їх вартість, яка з кожним разом є більшою, оскільки параметри наступних крейсерів є більшими, ніж у минулих. Для тестування проведено купівлю крейсера за назвою «Riker», у магазині він за послідовністю другий, за ціною – 5000 ігрової валюти. Інтерфейс результату купівлі в магазині гравцем крейсеру зображений на рисунку4.10.

Рисунок 4.10 – Результату купівлі в магазині гравцем крейсеру

Після придбання гравцем крейсеру, його можна випробувати в бойових випробуваннях в ігрових рівнях, для цього потрібно натиснути кнопку «SELECT», після чого буде відображена надпис «SELECTED». Інтерфейс результату вибору гравцем крейсеру в магазині зображений на рисунку 4.11.

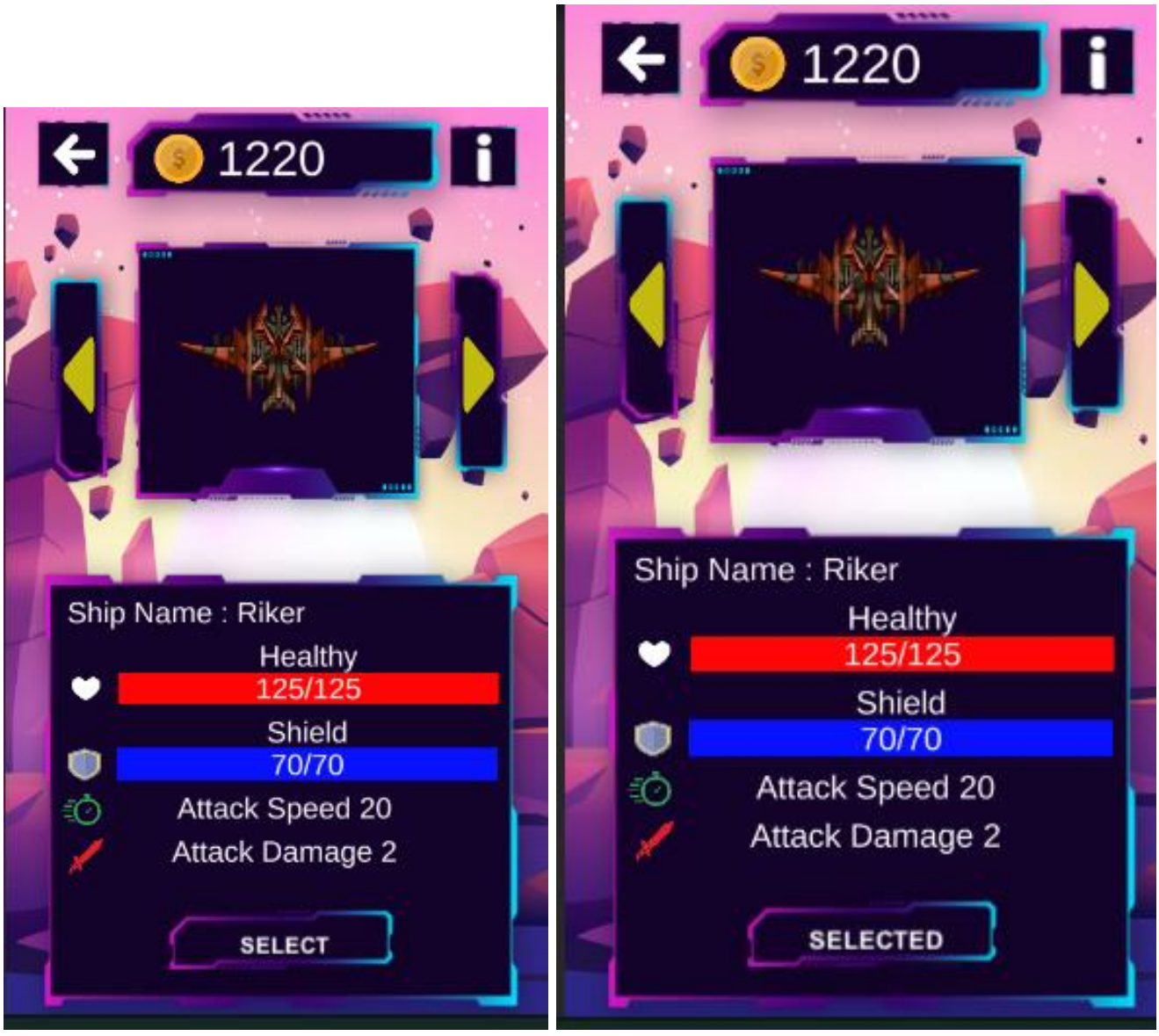

Рисунок 4.11 – Результату вибору гравцем крейсеру в магазині

В ігровій сцені рівня протестовано допоміжний баф підняття рівня, який надає крейсеру гравцю перевагу над супротивниками, це додаткові залпи бластеру. Є декілька рівнів від 1 до 4. Рівні крейсеру діють тільки в межах одного раунда, так як з наступний починає з першого. На першому рівні крейсер має тільки один залп вистрілу бластера, при отримані другого рівня кількість зростає удвічі, на 3 рівні має три залпи та в кінцевому – п’ять залпів. Інтерфейс результату отримання підняття рівня крейсеру гравцем зображено на рисунку 4.12.

Рисунок 4.12 – Результату отримань підняття рівня крейсеру гравцем

Кінцевим елементом тестування було музичний супровід в ігрових сценах, в кожну сцену обрану окрему пісню яка співпадала з ігровим інтерфейсом та функціоналом. Для регулювання гучності звуку в головному меню є можливість зменшення чи збільшеннярівня звуку. Також такі налаштування синхронні з усіма ігровими сценами та полосою в меню паузи ігрової сцени при проходження рівня. Інтерфейс результату регулювання звуку гравцем зображений на рисунку 4.13.

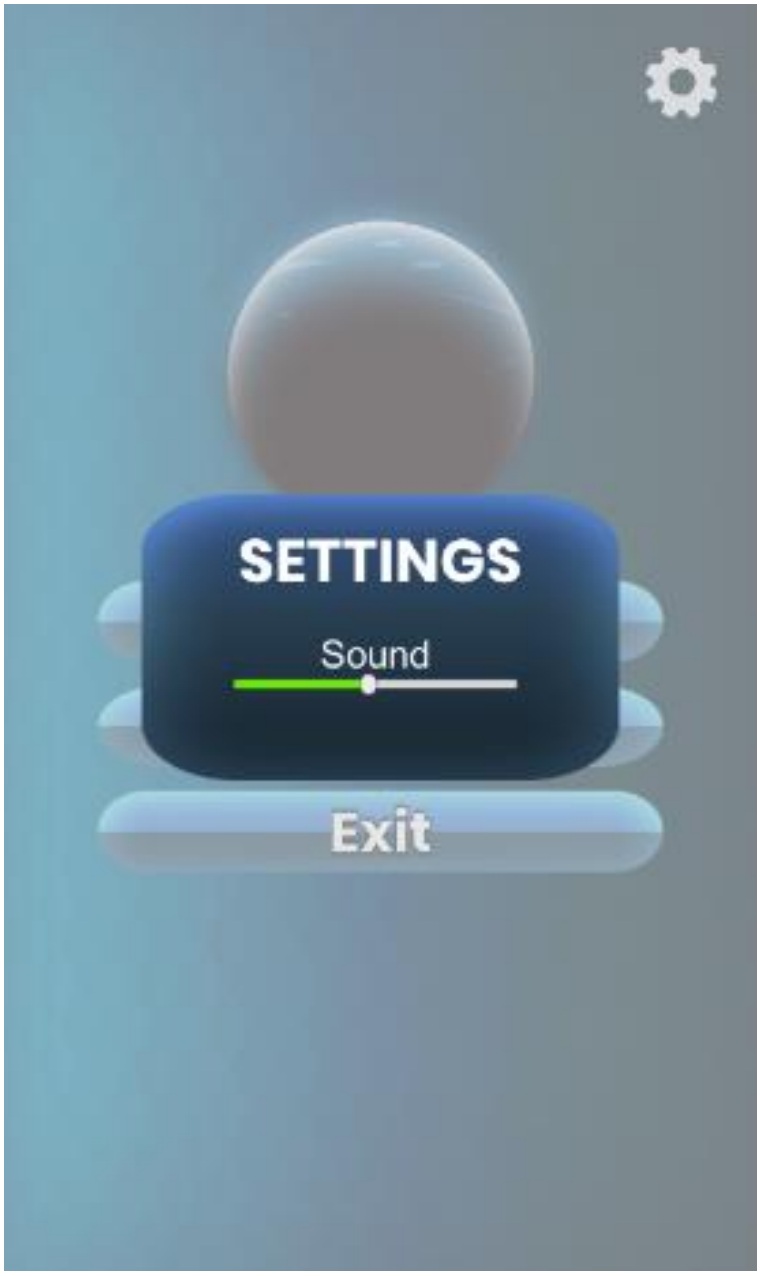

Рисунок 4.13 – Результату регулювання звуку гравцем

Деталі щодо мінімальної та рекомендованої конфігурації серверного комп’ютера можна знайти в таблицях 4.1 та 4.2.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 1.7 ГГц
Об’єм оперативної пам’яті	1 ГБ для 32-розрядної системи і 2 ГБ для 64-разрядної системи
Місце на жорсткому диску	43 166КБ
Операційна система	Від Android 4.4 та вище

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 2.4 ГГц
Об’єм оперативної пам’яті	2 ГБ для 32-розрядної системи і 4 ГБ для 64-разрядної системи
Розмір жорсткого диску	43 166КБ
Операційна система	версія Android 9.0

4.3 Висновки

У четвертому розділі було проведено тестування мобільного ігрового додатку шутер гри «Space confrontation». Проаналізовано методи тестування. Було перевірено швидкість виконання переходу з одного ігрового інтерфейсу в інший, та всі важливі функціональні можливості гри, з описом їх роботи і зображенням в звіті.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено метод високорівневого ускладнення гри та побудовано модель роботи ігрової системи і модель внутрішнього ігрового магазину. Для розробки було використано середовище програмування Visual Studio 2019 та програмний рушій Unity 2D, також для реалізації растрових зображень, які використовуватимуться для візуалізації та сприйняття гравцем ігрового процесу, було використано програму Adobe Photoshop.

Було проаналізовано стан розвитку мобільних ігор. Розглянуто основні аналоги, визначено їх особливості та недоліки і проведено порівняння з власним програмним продуктом.

У результаті аналізу обрано мови програмування C# та технології, які має програмний рушій Unity для графічного інтерфейсу.

У результаті виконання бакалаврської дипломної роботи було зроблено:

– розроблено метод високорівневого ускладення гри та удосконалено його з полегшим створенням наступних рівнів;

– розроблено модель ігрової системи;

– розроблено маніпулятивну систему керування косміним крейсером за допомогою сенсорного екрану мобільного телефону;

– спроєктовано внутрішній ігровий магазин для полегшення проходження гри та його різнобразних можливостей гри;

– розроблено основні модулі ігрового додатку;

– спроєктовано зручний користувацький інтерфейс;

– розроблено та удосконалено графічні ефекти, об’єкти та прифаби для ігрового додатку;

– надано музичне супроводження під час гри чи під час знаходження в ігровому інтерфейсі;

– розроблено мобільниф ігровий додаток «Space confrontation»;

– проведено тестування в програмному ігровому рушії та на мобільних пристроях ігрового додатку.

Перед створенням програмного продукту було розроблено схеми загального алгоритму роботи додатку, обробки вхідних та вихідних даних. Розроблено структуру високорівневого ускладнення гри та алгоритмів роботи мобільного ігрового додатку.

Тестування програми довело повну працездатність даного мобільного програмного додатка та відповідність поставленому технічному завданню.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розробка мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/.

2. Войтко В.В. Особливості розробки мобільної шутер гри «SPACE CONFRONTATION» / В.В. Войтко, А. В. Денисюк, О. В. Гаврилюк, Н. Є. Барчук, Д. С. В. Самарасінгхе // Матеріали Всеукраїнської науково-практичної інтернет-конференції "Молодь в науці: дослідження, проблеми, перспективи - 2022", Секція - Інформаційні технології та комп‘ютерна інженерія. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/viewFile/16192/13632.

3. Технології розробки комп’ютерних ігор [Електронний ресурс] – Режим доступу до ресурсу: http://repository.kpi.kharkov.ua/bitstream/KhPI-Press/37404/1/prohramy_2018_Tekhnolohii_rozrobky.pdf.

4. Mobile game [Електронний ресурс] – Режим доступу до ресурсу: https://www.wikiwand.com/en/Mobile_game.

5. Galaxy Shooter: Air Force War [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.infinite.shooting.galaxy.attack&hl=ru&gl=US.

6. Pirates Of Galaxy: Epic hunter [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.juudoo.pirates.of.galaxy&hl=uk&gl=US.

7. Sky Force 2014 [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=pl.idreams.skyforcehd&hl=ru&gl=US .

8. Level Up!: The Guide to Great Video Game Design 1-е видання: навч. посіб. / John Wiley & Sons – September 14, 2010. – 514 ст..

9. Software design [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Software_design.

10. Level Up!: The Guide to Great Video Game Design 2-е видання: навч. посіб. / John Wiley – April 28, 2014. – 560 с.

11. Epic Games [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Epic_Games.

12. Sample [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Sample.

13. Non-player character [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Non-player_character.

14. Мікротранзакція [Електронний ресурс] – Режим доступу до ресурсу: https://uk.upwiki.one/wiki/Microtransaction.

15. History of video games [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/History_of_video_games .

16. Graphical user interface [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Graphical_user_interface.

17. Raster graphics [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Raster_graphics.

18. Raster graphics editor [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Raster_graphics_editor.

19. Photoshop [Електронний ресурс] – Режим доступу до ресурсу: https://www.adobe.com/ua/products/photoshop.html.

20. Тестування мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/testuvannya-mobilnih-dodatkiv/.

21. Керування ігровими об'єктами (GameObjects) за допомогою компонентів [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/ru/530/Manual/ControllingGameObjectsComponents.html

22. ТОП 15 найкращих мобільних інструментів тестування у 2021 році для Android та iOS [Електронний ресурс] – Режим доступу до ресурсу: https://uk.myservername.com/top-15-best-mobile-testing-tools-2021.

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"_31_" _березня_2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка мобільної шутер гри «Space confrontation»»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доц. Д.І. Катєльніков
31 березня 2022 р.
Виконав:
_____студент гр. 1ПІ-19мс2 Д.С.В. Самарасінгхе
31 березня 2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка мобільної шутер гри «Space confrontation»».

Галузь застосування – ігровий мобільний додаток.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 12 від « 7 » лютого 2022 р.

3. Мета та призначення розробки.

Підвищення можливостей тренування швидкості реакції та адаптивності гравця до рівня ігрової стратегії шляхом розробки та використання мобільної ігрової системи з широким функціоналом посилення юніта і розвиненою системою адаптивного рівневого ускладнення гри, що дозволить реалізувати ігрову систему, орієнтовану на розвиток конкретного користувача.

4. Вихідні дані для проведення БДР

1. Розробка мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/.
2. Технології розробки комп’ютерних ігор [Електронний ресурс] – Режим доступу до ресурсу: http://repository.kpi.kharkov.ua/bitstream/KhPI-Press/37404/1/prohramy_2018_Tekhnolohii_rozrobky.pdf.
3. Mobile game [Електронний ресурс] – Режим доступу до ресурсу: https://www.wikiwand.com/en/Mobile_game.

5. Технічні вимоги

Для створення ігрової системи необхідно:

– розробити метод високорівневого ускладення гри та удосконалити його автоматизацією полегшеного створення наступних рівнів;

– розробити модель ігрової системи;

– розробити маніпулятивну систему керування косміним крейсером за допомогою сенсорного екрану мобільного телефону;

– спроєктувати внутрішній ігровий магазин для остащення ігрових юнітів;

– розробити основні модулі ігрового додатку;

– спроєктувати зручний користувацький інтерфейс;

– розробити та удосконалити графічні ефекти, об’єкти та прифаби для ігрового додатку;

– надати музичне супроводження під час гри чи під час знаходження в ігровому інтерфейсі;

– розробити мобільний ігровий додаток «Space confrontation»;

– провести тестування мобільної гри в програмному ігровому рушії та на мобільних пристроях.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка мобільної шутер гри «Space confrontation»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Катєльніков Д. І.

Оригінальність	**92%**
Схожість	8%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

 Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

 Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Самарасінгхе Д. С. В.

Керівник роботи _____ Катєльніков Д. І.

Додаток В – Лістинг програми

Програмний код скрипта MainMenuController

```
using UnityEngine;

using UnityEngine.UI;
using UnityEngine.SceneManagement;
public class MainMenuController : MonoBehaviour
{
    [Header("Settings")]
    [SerializeField] private Button Settings;
    [Space]
    [Header("Settings for center Buttons")]
    [SerializeField] private Button PlayButton;
    [SerializeField] private Button ShopButton;
    [SerializeField] private Button ExitButton;

    private void Start()
    {
        PlayButton.onClick.AddListener(Play);
        ShopButton.onClick.AddListener(ShowShop);
        ExitButton.onClick.AddListener(Exit);
    }

    private void Play()
    {
        Debug.Log("Play");
        SceneManager.LoadScene(PrefsKey.levelsceneName);
    }

    private void ShowShop()
    {
        SceneManager.LoadScene(PrefsKey.shopSceneName);
    }

    private void Exit()
    {
        Application.Quit();
    }
    public void ClearSave()
    {
        PlayerPrefs.DeleteAll();
    }
}
```

Програмний код скрипта MusicController

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

[RequireComponent(typeof(AudioSource))]
public class MusicController : MonoBehaviour
{
    [SerializeField] private AudioClip audioClip;
    [SerializeField] private Slider slider;

    private AudioSource audio;

    private void OnValidate()
    {
        if(audio == null)
        {
            audio = GetComponent<AudioSource>();
        }
    }

    private void Awake()
    {
        if (audio == null)
        {
            audio = GetComponent<AudioSource>();
        }
```

```
    }

    private void OnEnable()
    {
        audio.clip = audioClip;
        audio.Play();
        audio.volume = PlayerPrefs.GetFloat(PrefsKey.soundKey, 1f);
        if(slider != null)
        {
            slider.value = audio.volume;
        }
    }

    private void OnDisable()
    {
        audio.Stop();
    }

    public void Save(System.Single volume)
    {
        audio.volume = volume;
        PlayerPrefs.SetFloat(PrefsKey.soundKey, volume);
    }


}
```

Програмний код скрипта BackgroundManager

```
using UnityEngine;
using UnityEngine.UI;


public class BackgroundManager : MonoBehaviour
{
    [SerializeField] private SpriteRenderer background;
    [SerializeField] private Text text;
    [SerializeField] private BackgroungVariant[] backgroungVariants;
    [Space]
    [SerializeField] private TopButtonsManager buttonsManager;
    private BackgroungVariant current = new BackgroungVariant();

    private void OnEnable()
    {
        buttonsManager.OnRegionChenged += ChangeBackgroundSprite;
    }

    private void OnDisable()
    {
        buttonsManager.OnRegionChenged -= ChangeBackgroundSprite;
    }

    private void ChangeBackgroundSprite(LevelRegion levelRegion)
    {
        if (current.region == levelRegion) return;
```

```
        foreach(BackgroungVariant backgroungVariant in backgroungVariants)
        {
            if (backgroungVariant.region == levelRegion)
            {
                current = backgroungVariant;
            }
        }

        background.sprite = current.backgroundSprite;
        text.text = current.RegionName;
    }

    [System.Serializable]
    public class BackgroungVariant
    {
        public LevelRegion region;
        public string RegionName;
        public Sprite backgroundSprite;
    }


}
```

Програмний код скрипта TopButtonsManager

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

```
public class TopButtonsManager : MonoBehaviour
{
    public event System.Action<LevelRegion> OnRegionChenged;

    [SerializeField] private LevelRegion current;
    [Space]
    [SerializeField] private Button back;
    [SerializeField] private Button next;

    private void Awake()
    {
        Chek();
        back.onClick.AddListener(MoveBack);
        next.onClick.AddListener(MoveNext);
        SetEvent();
    }

    public void MoveBack()
    {
        int value = (int)current - 1;
        current = (LevelRegion)value;
        Chek();
        SetEvent();
    }

    public void MoveNext()
    {
```

```
    int value = (int)current + 1;
    current = (LevelRegion)value;
    Chek();
    SetEvent();
}

private void SetEvent()
{
    OnRegionChenged?.Invoke(current);
}

private void Chek()
{
    switch (current)
    {
        case LevelRegion.First:
            back.gameObject.SetActive(false);
            next.gameObject.SetActive(true);
            break;
        case LevelRegion.Second:
            back.gameObject.SetActive(true);
            next.gameObject.SetActive(true);
            break;
        case LevelRegion.Third:
            next.gameObject.SetActive(false);
            back.gameObject.SetActive(true);
            break;
    }
```

```
    }
}
```

Програмний код скрипта LevelManager

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

public enum LevelRegion: byte
{
    First = 0,
    Second = 1,
    Third = 2
}

public class LevelManager : MonoBehaviour
{
    [SerializeField] private LevelButton prefab;
    [SerializeField] private Transform perent;
    [Space]
    [SerializeField] private TopButtonsManager buttonsManager;

    private int levelsInRegion = 20;
    private int startIndex;
    private int lastOpenedLevel = 1;

    private List<LevelButton> levels = new List<LevelButton>();

    private void OnEnable()
```

```
{
    buttonsManager.OnRegionChenged += ChangeRegion;
}

private void OnDisable()
{
    buttonsManager.OnRegionChenged -= ChangeRegion;
}

private void Awake()
{
    CreateLevels();
    Activate();
}

private void Activate()
{
    if (PlayerPrefs.HasKey(PrefsKey.nextLevel))
    {
        lastOpenedLevel = PlayerPrefs.GetInt(PrefsKey.nextLevel) + 1;
    }
}

private void CreateLevels()
{
    for (int i = 0; i < levelsInRegion; i++)
    {
        LevelButton levelButton = Instantiate(prefab, perent);
```

```
            levelButton.SetLevelSettings(i, 0);
            levels.Add(levelButton);
        }
    }

    private void ChangeRegion(LevelRegion levelRegion)
    {
        startIndex = ((int)levelRegion * 20) + 1;
        ChangeLevelSettings();
    }

    private void ChangeLevelSettings()
    {
        for (int i = 0; i < levels.Count; i++)
        {
            levels[i].SetLevelSettings(startIndex++, lastOpenedLevel);
            Debug.Log($"Last opened Level {lastOpenedLevel}");
        }
    }

}
```

Програмний код скрипта PlayerShipSettings

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

public class PlayerShipSettings : MonoBehaviour
{
```

```
[SerializeField] private int ID;
[SerializeField] private int Healthy;
[SerializeField] private int Shield;
[SerializeField] private int AttckDamage;
[SerializeField] private int AttckSpeed;

[SerializeField] private List<Sprite> ships;

[SerializeField] private SpriteRenderer shipVizual;
[SerializeField] private Player player;
[SerializeField] private PlayerShooting playerShooting;

[SerializeField] private PolygonCollider2D PolygonCollider2D;
private Rigidbody2D rigidbody2D;

private void Awake()
{
    ID = PlayerPrefs.GetInt(PrefsKey.shipIDKey, 0);
    Shield = PlayerPrefs.GetInt(PrefsKey.shipShieldKey, 50);
    Healthy = PlayerPrefs.GetInt(PrefsKey.shipHealthyKey, 100);
    AttckDamage = PlayerPrefs.GetInt(PrefsKey.shipDamageKey, 1);
    AttckSpeed = PlayerPrefs.GetInt(PrefsKey.shipSpeedKey, 10);
    Init();
}

private void Init()
{
    shipVizual.sprite = SetShipSprite();
```

```
    player.health = Healthy;
    player.shield = Shield;
    playerShooting.AttackSpeed = AttckSpeed;
    playerShooting.AttackDamage= AttckDamage;
}

private void Start()
{
    if (PolygonCollider2D == null) return;

    for (int i = 0; i < PolygonCollider2D.pathCount; i++)
    {
        PolygonCollider2D.SetPath(i, new Vector2[0]);
    }
    PolygonCollider2D.pathCount = shipVizual.sprite.GetPhysicsShapeCount();

    List<Vector2> path = new List<Vector2>();
    for (int i = 0; i < PolygonCollider2D.pathCount; i++)
    {
        path.Clear();
        shipVizual.sprite.GetPhysicsShape(i, path);
        PolygonCollider2D.SetPath(i, path.ToArray());
    }
}

private Sprite SetShipSprite()
{
    return ships[ID];
```

```
    }

}
```

Програмний код скрипта Player

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

using UnityEngine.UI;
using UnityEngine.Events;
using TMPro;

/// <summary>
/// This script defines which sprite the 'Player" uses and its health.
/// </summary>

public class Player : MonoBehaviour
{
    public static Player instance;

    public GameObject destructionFX;
    public UnityEvent OnDestroy;

    public int health;
    private int currenthealth;
    public int shield;
    private int currentshield;
```

```
[SerializeField] private Slider liveSlider;
[SerializeField] private Slider shieldSlider;

[SerializeField] private TextMeshProUGUI healthText;
[SerializeField] private TextMeshProUGUI shieldText;


private void Awake()
{
    if (instance == null)
        instance = this;
}

private void Start()
{
    currenthealth = health;
    currentshield = shield;
    ChangeLiveText();
    ChangeShieldText();
}

public void ChangeLiveText()
{
    healthText.text = currenthealth + "/" + health;
}

public void ChangeShieldText()
{
```

```
    shieldText.text = currentshield + "/" + shield;
}

private void LiveSliderValue(int value)
{
    float newvalue = CanculateLiveSliderValue(value);
    liveSlider.value = newvalue;
}

private void ShieldSliderValue(int value)
{
    float newvalue = CanculateShieldSliderValue(value);
    shieldSlider.value = newvalue;
}

//method for damage proceccing by 'Player'
public void GetDamage(int damage)
{
    Destruction(damage);
}

public void GetTreatment(int treatment)
{
    liveSlider.value += liveSlider.value + CanculateLiveSliderValue(treatment);
}

private float CanculateLiveSliderValue(int value)
{
```

```
    currenthealth -= value;
    return ((currenthealth * 1f) / health);
}

private float CanculateShieldSliderValue(int value)
{
    currentshield -= value;
    return ((currentshield * 1f)/ shield);
}

//'Player's' destruction procedure
void Destruction(int damage)
{
    Instantiate(destructionFX, transform.position, Quaternion.identity);

    if(currentshield == 0)
    {
        LiveSliderValue(damage);
    }
    else if(currentshield < 0)
    {
        LiveSliderValue(Mathf.Abs(currentshield));
        shieldSlider.value = currentshield = 0;
    }
    else
    {
       ShieldSliderValue(damage);
        if (currentshield < 0)
```

```
            {
                LiveSliderValue(Mathf.Abs(currentshield));
                shieldSlider.value = currentshield = 0;
            }
        }

        if (currenthealth < 0)
        {
            liveSlider.value = currenthealth = 0;
            OnDestroy?.Invoke();
            Destroy(gameObject);
        }
    }
}
```

Програмний код скрипта PlayerMoving

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

/// <summary>
/// This script defines the borders of ‘Player’s’ movement. Depending on the chosen handling type, it moves the ‘Player’ together with the pointer.
/// </summary>

[System.Serializable]
public class Borders
{
    [Tooltip("offset from viewport borders for player's movement")]
```

```
    public float minXOffset = 1.5f, maxXOffset = 1.5f, minYOffset = 1.5f, maxYOffset = 1.5f;
    [HideInInspector] public float minX, maxX, minY, maxY;
}

public class PlayerMoving : MonoBehaviour {

    [Tooltip("offset from viewport borders for player's movement")]
    public Borders borders;
    Camera mainCamera;
    bool controlIsActive = true;

    public float speed;
    //anpublic FixedJoystick fixedJoystick;
    public Rigidbody2D rb2D;

    public static PlayerMoving instance; //unique instance of the script for easy access to the script

#if UNITY_EDITOR
    public bool MoveByFingerOrMouse;
#endif

    private void Awake()
    {
        if (instance == null)
            instance = this;
    }
```

```
    private void Start()
    {
        mainCamera = Camera.main;
        ResizeBorders();                    //setting 'Player's' moving borders deending on Viewport's size
    }


    /* public void FixedUpdate()
     {
         Vector3 direction = Vector3.forward * fixedJoystick.Vertical + Vector3.right * fixedJoystick.Horizontal;
         rb2D.AddForce(direction * speed * Time.fixedDeltaTime, ForceMode2D.Impulse);
     }*/


    private void Update()
    {
//#if UNITY_STANDALONE || UNITY_EDITOR      //if the current platform is not mobile, setting mouse handling


        if (Input.GetMouseButton(0)) //if mouse button was pressed
        {
            transform.position = Vector3.MoveTowards(transform.position, mainCamera.ScreenToWorldPoint(Input.mousePosition), speed * Time.deltaTime);
        }
//#endif


#if UNITY_IOS || UNITY_ANDROID //if current platform is mobile,


        if (Input.touchCount == 1) // if there is a touch
```

```
        {
            Touch touch = Input.touches[0];
            Vector3 touchPosition = mainCamera.ScreenToWorldPoint(touch.position); //calculating touch position in the world space
            touchPosition.z = transform.position.z;
            transform.position = Vector3.MoveTowards(transform.position, touchPosition, 30 * Time.deltaTime);
        }
#endif
        transform.position = new Vector3     //if 'Player' crossed the movement borders, returning him back
            (
                Mathf.Clamp(transform.position.x, borders.minX, borders.maxX),
                Mathf.Clamp(transform.position.y, borders.minY, borders.maxY),
                0
            );

    }

    //setting 'Player's' movement borders according to Viewport size and defined offset
    void ResizeBorders()
    {
        borders.minX = mainCamera.ViewportToWorldPoint(Vector2.zero).x + borders.minXOffset;
        borders.minY = mainCamera.ViewportToWorldPoint(Vector2.zero).y + borders.minYOffset;
        borders.maxX = mainCamera.ViewportToWorldPoint(Vector2.right).x - borders.maxXOffset;
        borders.maxY = mainCamera.ViewportToWorldPoint(Vector2.up).y - borders.maxYOffset;
```

```
    }
}
```

Програмний код скрипта PlayerShooting

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

using DG.Tweening;



//guns objects in 'Player's' hierarchy
[System.Serializable]
public class Guns
{
    public GameObject rightGun, leftGun, centralGun;
    [HideInInspector] public ParticleSystem leftGunVFX, rightGunVFX, centralGunVFX;
}

[System.Serializable]
public class Rockets
{
    public GameObject leftRocket, rightRocket;
    [HideInInspector] public ParticleSystem leftRocketVFX, rightRocketVFX;
}

public class PlayerShooting : MonoBehaviour {
```

```
[Tooltip("shooting frequency. the higher the more frequent")]
public float fireRate;

[Tooltip("projectile prefab")]
public GameObject projectileObject;
[Tooltip("current Lazer power")]
[Range(1, 4)]       //change it if you wish
public int weaponPower = 1;
[Range(0.2f, 1f)]
private float resetLazerTime;
public Guns guns;
[Space]
public GameObject projectileObjectRocket;
[Tooltip("current Rocket power")]
[Range(1, 2)]
public int RocketState = 1;
[SerializeField, Range(0.2f, 1f)]
private float resetRocketTime;
private float time;
private bool canRocketShooting = true;
public Rockets rockets;
//time for a new shot
[HideInInspector] public float nextFire;

[HideInInspector] public int AttackSpeed;
[HideInInspector] public int AttackDamage;

[SerializeField] bool AutoShooting;
```

```
[HideInInspector] public int maxweaponPower = 4;
public static PlayerShooting instance;

private void Awake()
{
    if (instance == null)
        instance = this;
}

private void Start()
{
    guns.leftGunVFX = guns.leftGun.GetComponent<ParticleSystem>();
    guns.rightGunVFX = guns.rightGun.GetComponent<ParticleSystem>();
    guns.centralGunVFX = guns.centralGun.GetComponent<ParticleSystem>();

    ChangeRocketSate(RocketState);
}

private void Update()
{

    if (AutoShooting)
    {
        if (Time.time > nextFire)
        {
            MakeAShot();
            nextFire = Time.time + 1 / fireRate;
```

```
            }
        }
        MakeAShotRocket();

       /* if (!canRocketShooting)
        {
            time -= Time.deltaTime;
            canRocketShooting = time < 0.01f;
        }*/
    }

    //method for a shot
    public void MakeAShot()
    {
        switch (weaponPower) // according to weapon power 'pooling' the defined anount of projectiles, on the defined position, in the defined rotation
        {
            case 1:
                CreateLazerShot(projectileObject, guns.centralGun.transform.position, Vector3.zero);
                guns.centralGunVFX.Play();
                break;
            case 2:
                CreateLazerShot(projectileObject, guns.rightGun.transform.position, Vector3.zero);
                guns.leftGunVFX.Play();
                CreateLazerShot(projectileObject, guns.leftGun.transform.position, Vector3.zero);
                guns.rightGunVFX.Play();
```

```
            break;
        case 3:
            CreateLazerShot(projectileObject, guns.centralGun.transform.position, Vector3.zero);
            CreateLazerShot(projectileObject, guns.rightGun.transform.position, new Vector3(0, 0, -5));
            guns.leftGunVFX.Play();
            CreateLazerShot(projectileObject, guns.leftGun.transform.position, new Vector3(0, 0, 5));
            guns.rightGunVFX.Play();
            break;
        case 4:
            CreateLazerShot(projectileObject, guns.centralGun.transform.position, Vector3.zero);
            CreateLazerShot(projectileObject, guns.rightGun.transform.position, new Vector3(0, 0, -5));
            guns.leftGunVFX.Play();
            CreateLazerShot(projectileObject, guns.leftGun.transform.position, new Vector3(0, 0, 5));
            guns.rightGunVFX.Play();
            CreateLazerShot(projectileObject, guns.leftGun.transform.position, new Vector3(0, 0, 15));
            CreateLazerShot(projectileObject, guns.rightGun.transform.position, new Vector3(0, 0, -15));
            break;
    }
}

public void MakeAShotRocket()
{
```

```
        if (!canRocketShooting)
            return;

        switch (RocketState)
        {
            case 1:
                return;
            case 2:
                CreateRocketShot(projectileObjectRocket,
rockets.rightRocket.transform.position, Vector3.zero);
                CreateRocketShot(projectileObjectRocket,
rockets.leftRocket.transform.position, Vector3.zero);
                break;
        }

        time = resetRocketTime;
        canRocketShooting = false;
    }


    public void ChangeRocketSate(int i)
    {
        if(i == 1)
        {
            rockets.leftRocket.SetActive(false);
            rockets.rightRocket.SetActive(false);
            return;
        }
```

```
        if (i > 2)
            i = 2;

        rockets.leftRocket.SetActive(!rockets.leftRocket.activeSelf);
        rockets.rightRocket.SetActive(!rockets.rightRocket.activeSelf);

        RocketState = i;
    }

    void CreateLazerShot(GameObject lazer, Vector3 pos, Vector3 rot) //translating 'pooled' lazer shot to the defined position in the defined rotation
    {
        GameObject Lazer  = Instantiate(lazer, pos, Quaternion.Euler(rot));
        SetParams(Lazer);
    }

    void CreateRocketShot(GameObject rocket, Vector3 pos, Vector3 rot) //translating 'pooled' lazer shot to the defined position in the defined rotation
    {
        GameObject Rocket = Instantiate(rocket, pos, Quaternion.Euler(rot));
        SetParams(Rocket);
    }

    void SetParams(GameObject obj)
    {
        DirectMoving directMoving = obj.GetComponent<DirectMoving>();
        Projectile projectile = obj.GetComponent<Projectile>();
```

```
        if(directMoving != null)
        {
            directMoving.speed = AttackSpeed;
        }
        if(projectile != null)
        {
            projectile.damage = AttackDamage;
        }
    }
}
```

Програмний код скрипта GameSettingsController

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

public class GameSettingsController : MonoBehaviour
{
    [SerializeField] private Canvas settingsCanvas;

    [SerializeField] private Slider soundSlider;
    [SerializeField] private Slider musicSlider;

    [SerializeField] private GameObject finishPanel;
    [SerializeField] private TextMeshProUGUI finaltext;
```

```
[SerializeField] private Button Ok;
[SerializeField] private Button restart;

private void Awake()
{
    if(settingsCanvas.enabled == true)
    {
        HideSetting();
    }

    if (finishPanel.activeSelf)
    {
        finishPanel.SetActive(false);
    }
}

public void ShowSettings()
{
    settingsCanvas.enabled = true;
    Time.timeScale = 0;
}

public void HideSetting()
{
    settingsCanvas.enabled = false;
    Time.timeScale = 1;
}
```

```
    public void Finish(bool win = true)
    {
        Ok.gameObject.SetActive(false);
        restart.gameObject.SetActive(false);

        finishPanel.SetActive(true);
        if (win)
        {
            finaltext.text = "VICTORY";
            Ok.gameObject.SetActive(true);
        }
        else
        {
            finaltext.text = "You are Lose Try again";
            Ok.gameObject.SetActive(true);
            restart.gameObject.SetActive(true);
        }

        Ok.onClick.AddListener(delegate                                    {
SceneManager.LoadScene(PrefsKey.levelsceneName);  });
        restart.onClick.AddListener(delegate                               {
SceneManager.LoadScene(PrefsKey.gameSceneName); });

    }
}
```

Програмний код скрипта ShopManager

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

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using UnityEngine.SceneManagement;

public class ShopManager : MonoBehaviour
{
    [Header("Data base Info")]
    [SerializeField] private DataInfo data;
    [SerializeField] private DataInfo.ShipInfo current;
    private int selectShipID = 0;
    private int curentShipID = 0;
    [Space]
    [Header("In Top Panel")]
    [SerializeField] private CoinManager coin;
    [SerializeField] private Button back;
    [Space]
    [Header("In Center Panel")]
    [SerializeField] private Image shipImage;
    [SerializeField] private Button Left;
    [SerializeField] private Button Right;
    [Space]
    [Header("In Buttom Panel")]
    [SerializeField] private TextMeshProUGUI ShipNameText;
    [SerializeField] private TextMeshProUGUI HealthyText;
    [SerializeField] private TextMeshProUGUI ShieldText;
    [SerializeField] private TextMeshProUGUI AttackSpeedText;
    [SerializeField] private TextMeshProUGUI AttackDamageText;
```

```
[SerializeField] private SelectButton selectButton;



private void Awake()
{
    if (Left != null) Left.onClick.AddListener(LeftButton);
    if (Right != null) Right.onClick.AddListener(RightButton);
    if (back != null) back.onClick.AddListener(GoToBack);
}

private void Start()
{
    if (PlayerPrefs.HasKey(PrefsKey.selectShipKey))
    {
        selectShipID = PlayerPrefs.GetInt(PrefsKey.selectShipKey, 0);
        curentShipID = selectShipID;
    }
    SetShipSetting(curentShipID);
}

private void SetShipSetting(int ID)
{
    current = data.GetShipByID(ID);

    shipImage.sprite = current.Image;
    ShipNameText.text = "Ship Name : " + current.Name;
    HealthyText.text = current.Healthy + "/" + current.Healthy;
    ShieldText.text = current.Shield + "/" + current.Shield;
```

```
    AttackSpeedText.text = "Attack Speed " + current.AttackSpeed;
    AttackDamageText.text = "Attack Damage " + current.AttackDamage;

    selectButton.SetShipID(ID);

    if (current.isBuyed)
    {
        selectButton.SetSelectedSettigs();
    }
    else
    {
        selectButton.SetBuySettings(current.Price);
    }
}

private void LeftButton()
{
    if (curentShipID == 0)
        curentShipID = data.dataCount - 1;
    else
        curentShipID--;

    SetShipSetting(curentShipID);

}

private void RightButton()
{
```

```
        if (curentShipID == data.dataCount - 1)
            curentShipID = 0;
        else
            curentShipID++;

        SetShipSetting(curentShipID);
    }

    private void GoToBack()
    {
        SceneManager.LoadScene(PrefsKey.menuSceneName);
    }
}
```

Програмний код скрипта DataInfo

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

public class DataInfo : MonoBehaviour
{
    [SerializeField] private List<ShipInfo> allShips;
    public int dataCount;

    private void OnValidate()
    {
        for(int i = 0; i < allShips.Count; i++)
        {
            allShips[i].ID = i;
        }
```

```
}

private void Start()
{
    dataCount = allShips.Count;
}

public ShipInfo GetShipByID(int id)
{
    ShipInfo ship = allShips.Find(x => x.ID == id);
    if (!ship.isBuyed)
        ship.isBuyed = PlayerPrefs.HasKey("Byed" + ship.ID);

    return ship;
}

public void IsBuyed(int id)
{
    ShipInfo ship = allShips.Find(x => x.ID == id);
    PlayerPrefs.SetInt("Byed" + ship.ID, ship.Price);
}

[System.Serializable]
public class ShipInfo
{
    public int ID;
    public Sprite Image;
    public string Name;
```

```
        public int Healthy;
        public int Shield;
        public int AttackDamage;
        public int AttackSpeed;
        public int Price;
        public bool isBuyed;
    }
}
```

Програмний код скрипта Wave

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

/// <summary>
/// This script generates an enemy wave. It defines how many enemies will be emerging, their speed and emerging interval.
/// It also defines their shooting mode. It defines their moving path.
/// </summary>
[System.Serializable]
public class Shooting
{
    [Range(0,100)]
    [Tooltip("probability with which the ship of this wave will make a shot")]
    public int shotChance;

    [Tooltip("min and max time from the beginning of the path when the enemy can make a shot")]
```

```
    public float shotTimeMin, shotTimeMax;
}

public class Wave : MonoBehaviour {

    #region FIELDS
    [Tooltip("Enemy's prefab")]
    public GameObject enemy;

    [Tooltip("a number of enemies in the wave")]
    public int count;

    [Tooltip("path passage speed")]
    public float speed;

    [Tooltip("time between emerging of the enemies in the wave")]
    public float timeBetween;

    [Tooltip("points of the path. delete or add elements to the list if you want to change the number of the points")]
    public Transform[] pathPoints;

    [Tooltip("whether 'Enemy' rotates in path passage direction")]
    public bool rotationByPath;

    [Tooltip("if loop is activated, after completing the path 'Enemy' will return to the starting point")]
    public bool Loop;
```

```
    [Tooltip("color of the path in the Editor")]
    public Color pathColor = Color.yellow;
    public Shooting shooting;

    [Tooltip("if testMode is marked the wave will be re-generated after 3 sec")]
    public bool testMode;

    public int OneEnemyCoins = 10;
    #endregion

    private void Start()
    {
        StartCoroutine(CreateEnemyWave());
    }

    IEnumerator CreateEnemyWave() //depending on chosed parameters generating enemies and defining their parameters
    {
        for (int i = 0; i < count; i++)
        {
            GameObject newEnemy;
            newEnemy = Instantiate(enemy, enemy.transform.position, Quaternion.identity);
            FollowThePath followComponent = newEnemy.GetComponent<FollowThePath>();
            followComponent.path = pathPoints;
            followComponent.speed = speed;
            followComponent.rotationByPath = rotationByPath;
            followComponent.loop = Loop;
            followComponent.SetPath();
```

```
            Enemy enemyComponent = newEnemy.GetComponent<Enemy>();
            enemyComponent.shotChance = shooting.shotChance;
            enemyComponent.shotTimeMin = shooting.shotTimeMin;
            enemyComponent.shotTimeMax = shooting.shotTimeMax;
            newEnemy.SetActive(true);
            yield return new WaitForSeconds(timeBetween);
        }
        if (testMode)       //if testMode is activated, waiting for 3 sec and re-generating the wave
        {
            yield return new WaitForSeconds(3);
            StartCoroutine(CreateEnemyWave());
        }
        else if (!Loop)
            Destroy(gameObject);
    }

    void OnDrawGizmos()
    {
        DrawPath(pathPoints);
    }

    void DrawPath(Transform[] path) //drawing the path in the Editor
    {
        Vector3[] pathPositions = new Vector3[path.Length];
        for (int i = 0; i < path.Length; i++)
        {
            pathPositions[i] = path[i].position;
```

```
        }
        Vector3[] newPathPositions = CreatePoints(pathPositions);
        Vector3 previosPositions = Interpolate(newPathPositions, 0);
        Gizmos.color = pathColor;
        int SmoothAmount = path.Length * 20;
        for (int i = 1; i <= SmoothAmount; i++)
        {
            float t = (float)i / SmoothAmount;
            Vector3 currentPositions = Interpolate(newPathPositions, t);
            Gizmos.DrawLine(currentPositions, previosPositions);
            previosPositions = currentPositions;
        }
    }

    Vector3 Interpolate(Vector3[] path, float t)
    {
        int numSections = path.Length - 3;
        int currPt = Mathf.Min(Mathf.FloorToInt(t * numSections), numSections - 1);
        float u = t * numSections - currPt;
        Vector3 a = path[currPt];
        Vector3 b = path[currPt + 1];
        Vector3 c = path[currPt + 2];
        Vector3 d = path[currPt + 3];
        return 0.5f * ((-a + 3f * b - 3f * c + d) * (u * u * u) + (2f * a - 5f * b + 4f * c - d) *
(u * u) + (-a + c) * u + 2f * b);
    }
```

Vector3[] CreatePoints(Vector3[] path) //using interpolation method calculating the path along the path points

```
    {
        Vector3[] pathPositions;
        Vector3[] newPathPos;
        int dist = 2;
        pathPositions = path;
        newPathPos = new Vector3[pathPositions.Length + dist];
        Array.Copy(pathPositions, 0, newPathPos, 1, pathPositions.Length);
        newPathPos[0] = newPathPos[1] + (newPathPos[1] - newPathPos[2]);
        newPathPos[newPathPos.Length - 1] = newPathPos[newPathPos.Length - 2] +
(newPathPos[newPathPos.Length - 2] - newPathPos[newPathPos.Length - 3]);
        if (newPathPos[1] == newPathPos[newPathPos.Length - 2])
        {
            Vector3[] LoopSpline = new Vector3[newPathPos.Length];
            Array.Copy(newPathPos, LoopSpline, newPathPos.Length);
            LoopSpline[0] = LoopSpline[LoopSpline.Length - 3];
            LoopSpline[LoopSpline.Length - 1] = LoopSpline[2];
            newPathPos = new Vector3[LoopSpline.Length];
            Array.Copy(LoopSpline, newPathPos, LoopSpline.Length);
        }
        return (newPathPos);
    }
}
```

Програмний код скрипта LevelBoss

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

```
public class LevelBoss : MonoBehaviour
{
    public static event System.Action<int> OnHitEmemy;

    #region FIELDS
    [Tooltip("Health points in integer")]
    public int health;

    [Tooltip("Enemy's projectile prefab")]
    public GameObject Projectile;

    [Tooltip("VFX prefab generating after destruction")]
    public GameObject destructionVFX;
    public GameObject hitEffect;

    [SerializeField] private float movingSpeed;
    [SerializeField] private int Coins;
    private Vector2 startPoint;
    float currentAngle = 0f;
    [Range(1,3)]
    [SerializeField] private float offsetParam;

  /*  [Range(0,100)]
    [SerializeField] public int shotChance; //probability of 'Enemy's' shooting during tha path
    [SerializeField] public float shotTimeMin, shotTimeMax; //max and min time for shooting from the beginning of the path*/
    #endregion
```

```
    private void Start()
    {
        startPoint = transform.position;
/*
        Invoke("ActivateShooting", Random.Range(shotTimeMin, shotTimeMax));*/
    }

    //coroutine making a shot
   /* void ActivateShooting()
    {
        if (Random.value < (float)shotChance / 100)                    //if random value less than shot probability, making a shot
        {
            Instantiate(Projectile, gameObject.transform.position, Quaternion.identity);
        }
    }
*/
    //method of getting damage for the 'Enemy'
    public void GetDamage(int damage)
    {
        health -= damage;
        //reducing health for damage value, if health is less than 0, starting destruction procedure
        if (health <= 0)
            Destruction();
        else
            Instantiate(hitEffect, transform.position, Quaternion.identity, transform);
    }
```

```
    //if 'Enemy' collides 'Player', 'Player' gets the damage equal to projectile's damage value
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            if (Projectile.GetComponent<Projectile>() != null)
                Player.instance.GetDamage(Projectile.GetComponent<Projectile>().damage);
            else
                Player.instance.GetDamage(1);
        }
    }

    //method of destroying the 'Enemy'
    void Destruction()
    {
        Instantiate(destructionVFX, transform.position, Quaternion.identity);
        OnHitEmemy?.Invoke(Coins);
        Destroy(gameObject);
    }

    private void BossMoving()
    {
        currentAngle += movingSpeed * Time.deltaTime;
        Vector2 offset = new Vector2(Mathf.Sin(currentAngle), Mathf.Cos(currentAngle)) * offsetParam;
        transform.position = startPoint + offset;
    }
```

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

Програмний код скрипта BossShhoting

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

public class BossShhoting : MonoBehaviour
{
    [Tooltip("shooting frequency. the higher the more frequent")]
    public float fireRate;

    [Tooltip("projectile prefab")]
    public GameObject projectileObject;
    [Tooltip("current Lazer power")]
    [Range(1, 4)]       //change it if you wish
    public int weaponPower = 1;
    [Range(0.2f, 1f)]
    private float resetLazerTime;
    public Guns guns;
    [Space]
    public GameObject projectileObjectRocket;
    [Tooltip("current Rocket power")]
```

```
[Range(1, 2)]
public int RocketState = 1;
[SerializeField, Range(0.2f, 1f)]
private float resetRocketTime;
private float time;
private bool canRocketShooting = true;
public Rockets rockets;
//time for a new shot
[HideInInspector] public float nextFire;

[SerializeField] bool AutoShooting;

[HideInInspector] public int maxweaponPower = 4;
public static BossShhoting instance;

private void Awake()
{
    if (instance == null)
        instance = this;
}

private void Start()
{
    guns.leftGunVFX = guns.leftGun.GetComponent<ParticleSystem>();
    guns.rightGunVFX = guns.rightGun.GetComponent<ParticleSystem>();
    guns.centralGunVFX = guns.centralGun.GetComponent<ParticleSystem>();

    ChangeRocketSate(RocketState);
```

```
    }

    private void Update()
    {

        if (AutoShooting)
        {
            if (Time.time > nextFire)
            {
                MakeAShot();
                nextFire = Time.time + 1 / fireRate;
            }
        }
        MakeAShotRocket();
    }

    //method for a shot
    public void MakeAShot()
    {
        switch (weaponPower) // according to weapon power 'pooling' the defined anount of projectiles, on the defined position, in the defined rotation
        {
            case 1:
                CreateLazerShot(projectileObject, guns.centralGun.transform.position, Vector3.zero);
                guns.centralGunVFX.Play();
                break;
            case 2:
```

```
            CreateLazerShot(projectileObject, guns.rightGun.transform.position, Vector3.zero);
            guns.leftGunVFX.Play();
            CreateLazerShot(projectileObject, guns.leftGun.transform.position, Vector3.zero);
            guns.rightGunVFX.Play();
            break;
        case 3:
            CreateLazerShot(projectileObject, guns.centralGun.transform.position, Vector3.zero);
            CreateLazerShot(projectileObject, guns.rightGun.transform.position, new Vector3(0, 0, -5));
            guns.leftGunVFX.Play();
            CreateLazerShot(projectileObject, guns.leftGun.transform.position, new Vector3(0, 0, 5));
            guns.rightGunVFX.Play();
            break;
        case 4:
            CreateLazerShot(projectileObject, guns.centralGun.transform.position, Vector3.zero);
            CreateLazerShot(projectileObject, guns.rightGun.transform.position, new Vector3(0, 0, -5));
            guns.leftGunVFX.Play();
            CreateLazerShot(projectileObject, guns.leftGun.transform.position, new Vector3(0, 0, 5));
            guns.rightGunVFX.Play();
            CreateLazerShot(projectileObject, guns.leftGun.transform.position, new Vector3(0, 0, 15));
            CreateLazerShot(projectileObject, guns.rightGun.transform.position, new Vector3(0, 0, -15));
            break;
```

```
    }
}

public void MakeAShotRocket()
{
    if (!canRocketShooting)
        return;

    switch (RocketState)
    {
        case 1:
            return;
        case 2:
            CreateRocketShot(projectileObjectRocket,
rockets.rightRocket.transform.position, Vector3.zero);
            CreateRocketShot(projectileObjectRocket,
rockets.leftRocket.transform.position, Vector3.zero);
            break;
    }

    time = resetRocketTime;
    canRocketShooting = false;
}

public void ChangeRocketSate(int i)
{
    if (i == 1)
    {
```

```
            rockets.leftRocket.SetActive(false);
            rockets.rightRocket.SetActive(false);
            return;
        }

        if (i > 2)
            i = 2;

        rockets.leftRocket.SetActive(!rockets.leftRocket.activeSelf);
        rockets.rightRocket.SetActive(!rockets.rightRocket.activeSelf);

        RocketState = i;
    }



    void CreateLazerShot(GameObject lazer, Vector3 pos, Vector3 rot) //translating 'pooled' lazer shot to the defined position in the defined rotation
    {
        GameObject Lazer = Instantiate(lazer, pos, Quaternion.Euler(rot));
        //SetParams(Lazer);
    }

    void CreateRocketShot(GameObject rocket, Vector3 pos, Vector3 rot) //translating 'pooled' lazer shot to the defined position in the defined rotation
    {
        GameObject Rocket = Instantiate(rocket, pos, Quaternion.Euler(rot));
        //SetParams(Rocket);
    }
```

Додаток Г – Графічна частина

ГРАФІЧНА ЧАСТИНА
РОЗРОБКА МОБІЛЬНОЇ ШУТЕР ГРИ «SPACE CONFRONTATION»

Рисунок Г.1 – Назва роботи

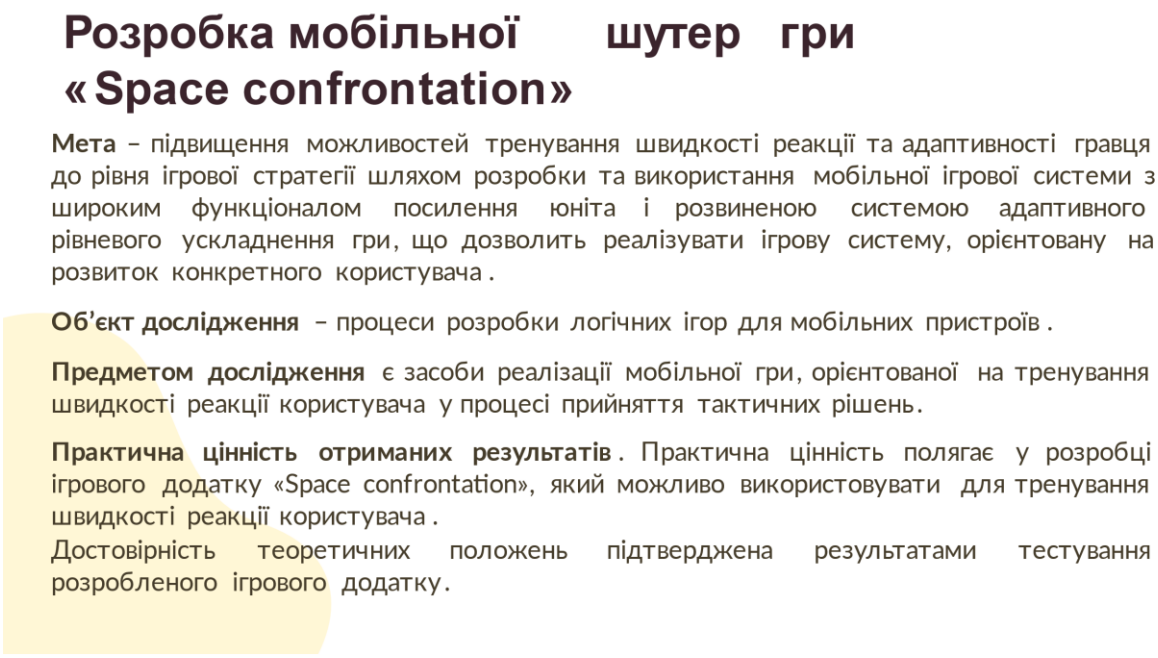

Рисунок Г.2 – Мета, об'єкт і предмет дослідження

Рисунок Г.3 – Наукова новизна

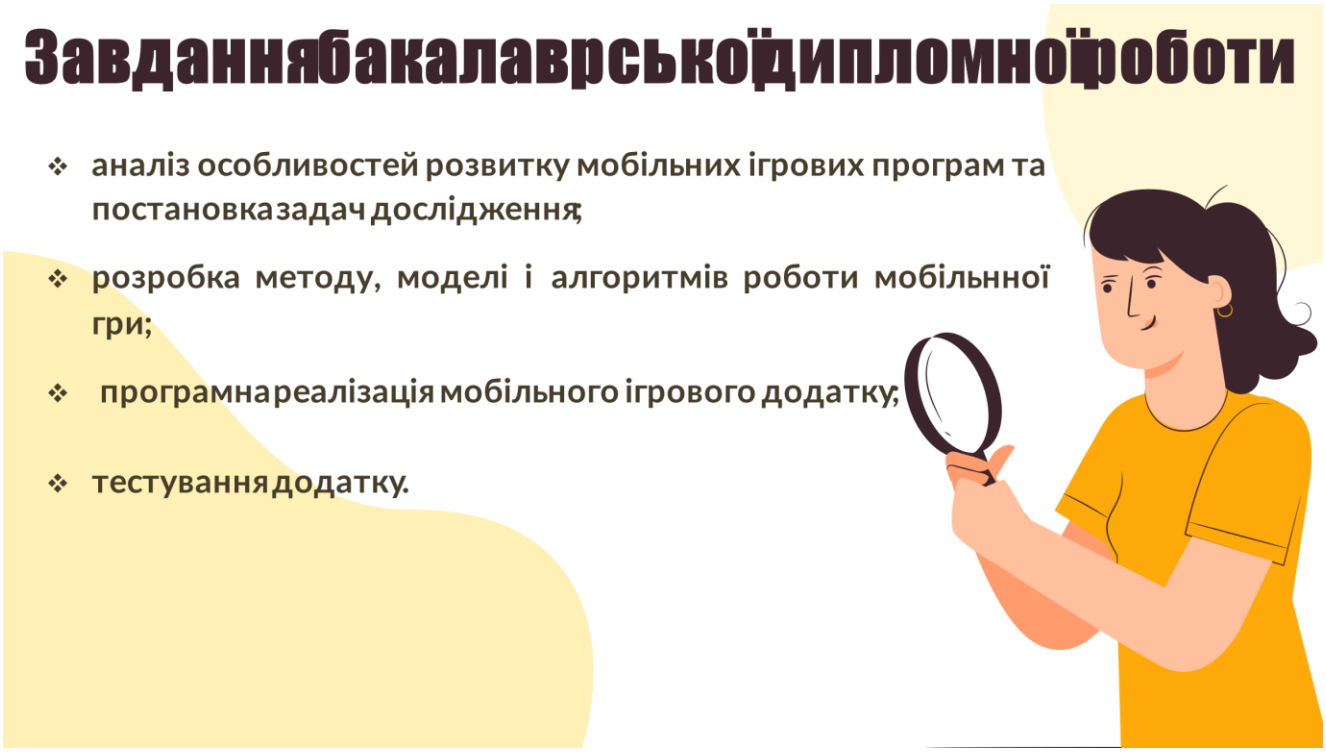

Рисунок Г.4 – Задачі бакалаврської дипломної роботи

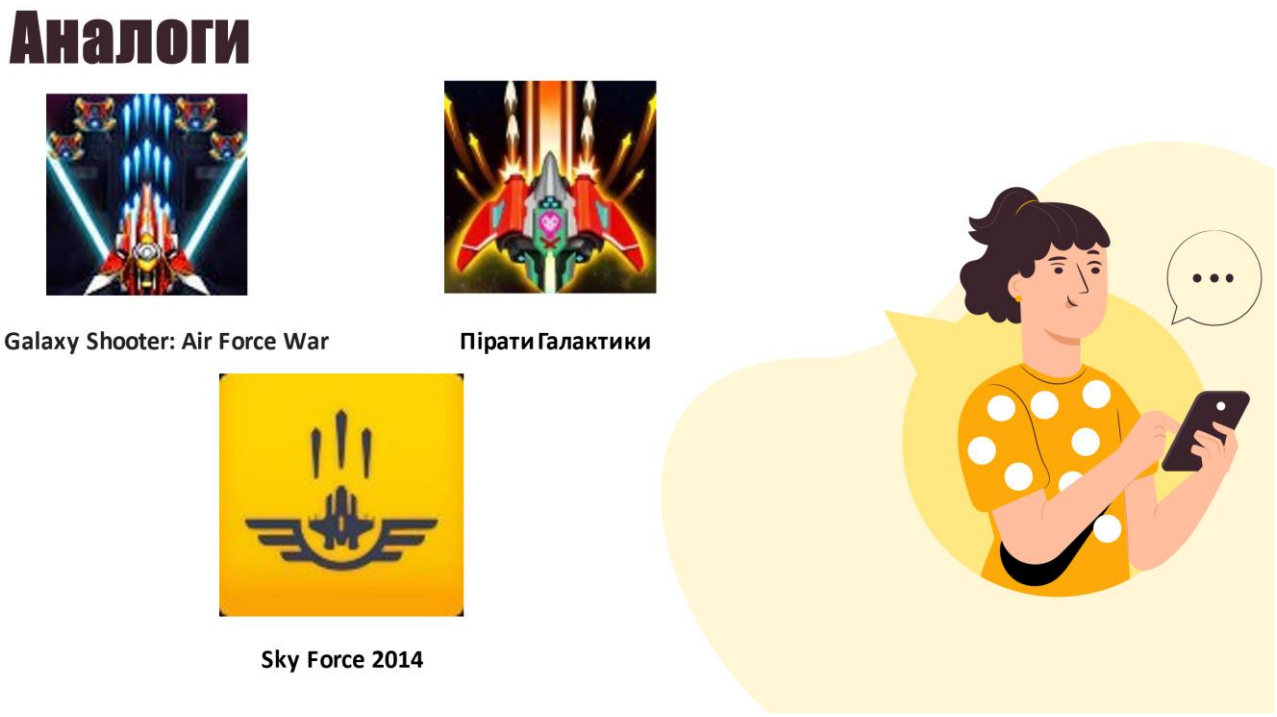

Рисунок Г.5 – Аналоги

Результати порівняльного аналізу аналогів

Функції	Galaxy Shooter: Air Force War	Пірати Галактики	Sky Force 2014	Space confrontation
Система внутриігрового магазину	-	+	-	+
Адаптація та унікальність бойових можливостей	-	+	+	+
Різні види ворожих юнитів та ігрових босів	-	+	+	+
Інтуїтивно зрозумілий інтерфейс користувача	+	-	+	+
Є реклама	+	+	+	-

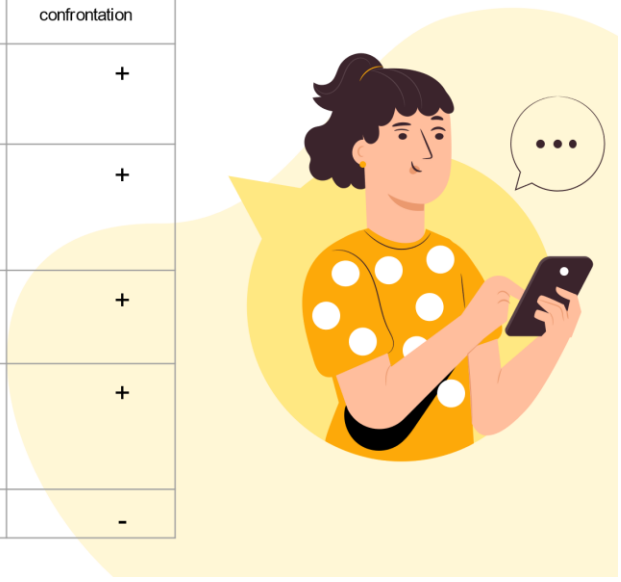

Рисунок Г.6 – Результати порівняльного аналізу аналогів

МЕТОД ВИСОКОРІВНЕВОГО УСКЛАДНЕННЯ ГРИ

1. **Початокігрового процесу**, для ідеальногопроходженнягравцю потрібнопроходитирівні з 100% запасом життя та більше 0% запасу захисного поля, якщо ж на 2 зірки то гравець матиме 100% життя та менше 0% захисного поля.
1. **Ініціалізаціяігрових одиниць середнього типу**
2. **Якщо гравець з легкістю пройшов 5 перших рівнів**, то йому надаються наступні 5 рівнів з важкими юнітами, що матимуть більші характеристики життя, сили пошкодження по гравцю. Якщо ж гравець не зміг пройти всі перші 5 рівнів на відмінно, то йому надаються наступні п'ять легких рівнів для покращення навичок.
3. **Ініціалізаціялегкого типу ігрових рівнів**
4. **Якщо гравець програв**, чи то важкий рівень чи легкий, тоді гравцю надається можливість перейти в ігровий магазин, для купівлі нових крейсерів, що мають більші характеристики, у разі проходженнятаких рівнів гравцю надається рівень з босом.
5. **Гравець можепридбати новіший крейсер**
6. **Ініціалізаціяважкого типу ігрових рівнів**
7. **Гравцю надаєтьсярівень з босом**
8. **Якщо гравець не пройшов рівень з босом**, то йому надається можливістьпридбати в ігровомумагазині новий крейсер, у іншому випадку він слідуєнаступним пригодам.
9. **Якщо гравець пройшов всі додатковіускладнені рівні**, то він закінчує гру із задоволеннямі на відмінно по 3 зірки кожний із рівнів.
10. **Ініціалізаціяігрових юнитів усіх типу рівнів (+50% к характеристикам супротивника)**

Рисунок Г.7 – Метод високорівневого ускладнення гри

БЛОК -СХЕМА АЛГОРИТМУ МЕТОДУ ВИСОКОРІВНЕВОГО УСКЛАДНЕННЯ ГРИ

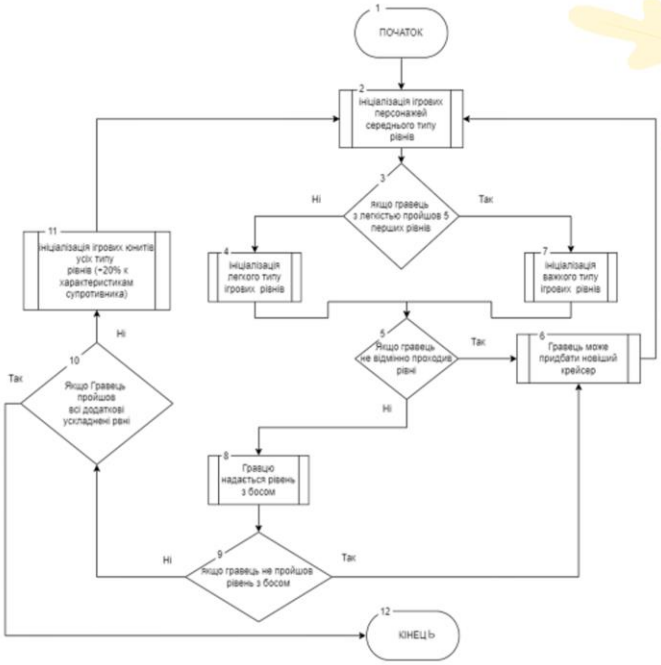

Рисунок Г.8 – Блок-схема алгоритму методу високорівневого ускладнення гри

МОДЕЛ Ь ІГРОВОГО ПРОЦЕСУ

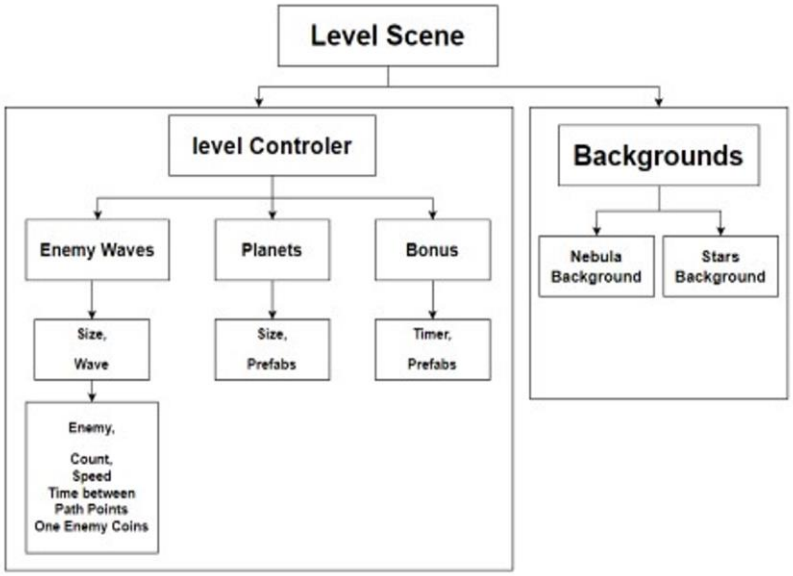

Рисунок Г.9 – Модель ігрового процесу

Розробка моделі внутрішнього ігрового магазину

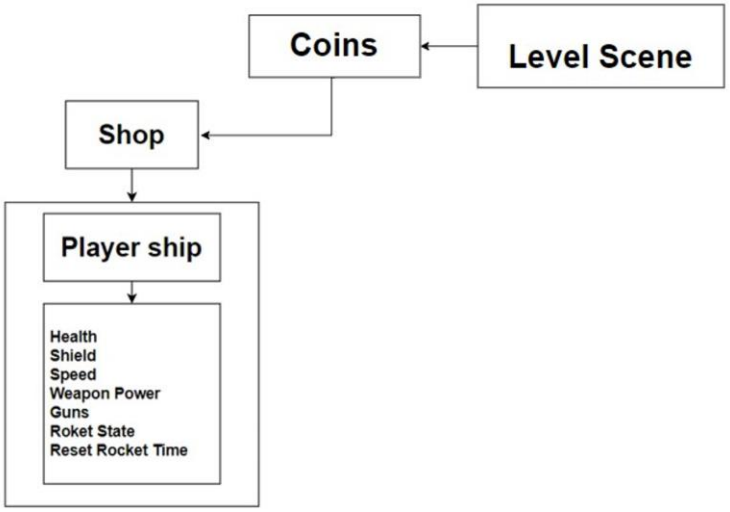

Рисунок Г.10 – Розробка моделі внутрішнього ігрового магазину

Рисунок Г.11 – Алгоритм роботи мобільного ігрового додатку

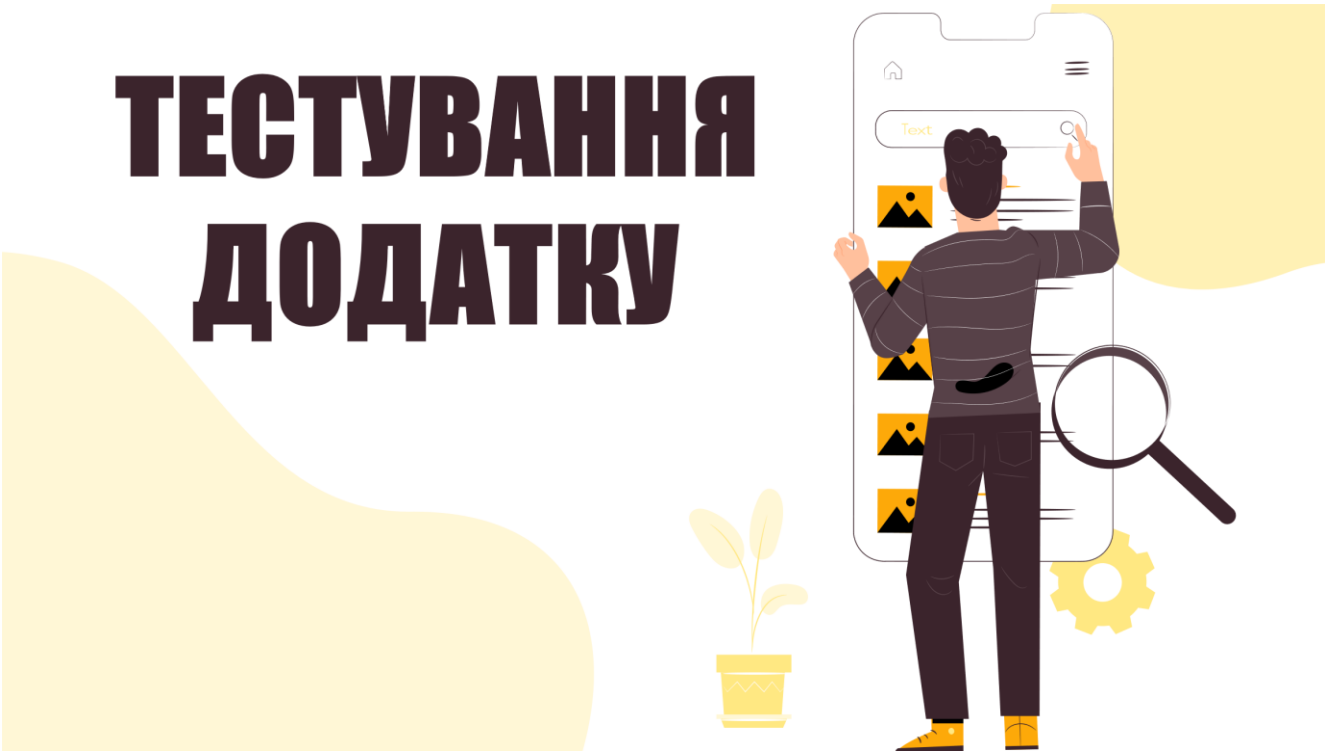

Рисунок Г.12 – Тестування додатку

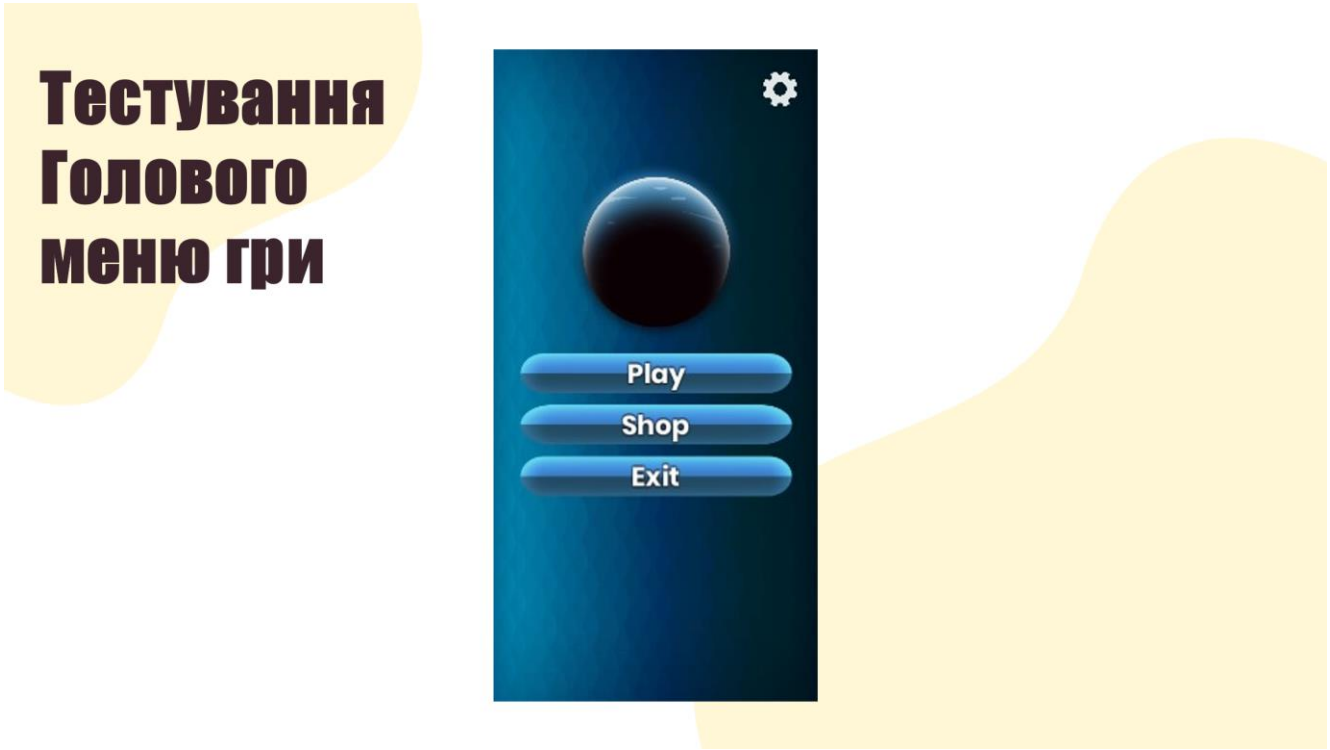

Рисунок Г.13 – Тестування Голового меню гри

Рисунок Г.14 – Тестування результату натискання кнопки «рівня 1»

Тестування результати покрокового проходження «рівня 1»

Рисунок Г.15 – Тестування результати покрокового проходження «рівня 1»

Тестування результат програшу гравця

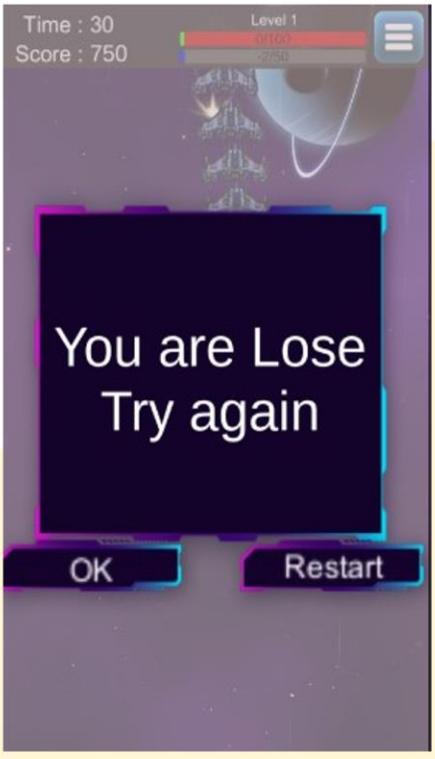

Рисунок Г.16 – Тестування результат програшу гравця

Тестування результат перемоги гравця

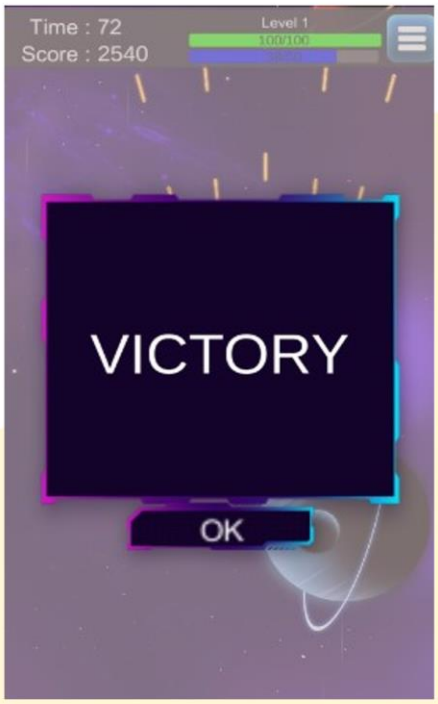

Рисунок Г.17 – Тестування результат перемоги гравця

Тестування результату прогресу проходження гравця

Рисунок Г.18 – Тестування результату прогресу проходження гравця

Тестування результату проходження гравцем рівень з босом

Рисунок Г.19 – Тестування результату проходження гравцем рівень з босом

Тестування результату паузи гри

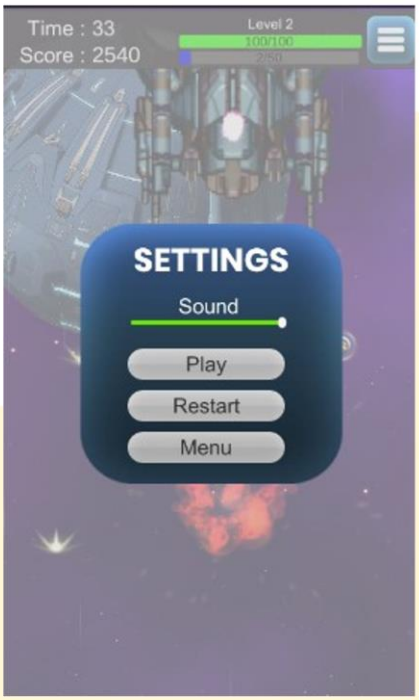

Рисунок Г.20 – Тестування результату паузи гри

Тестування результату купівлі в магазині гравцем крейсеру

Рисунок Г.21 – Тестування результату купівлі в магазині гравцем крейсеру

Тестування результату вибору гравцем крейсеру в магазині

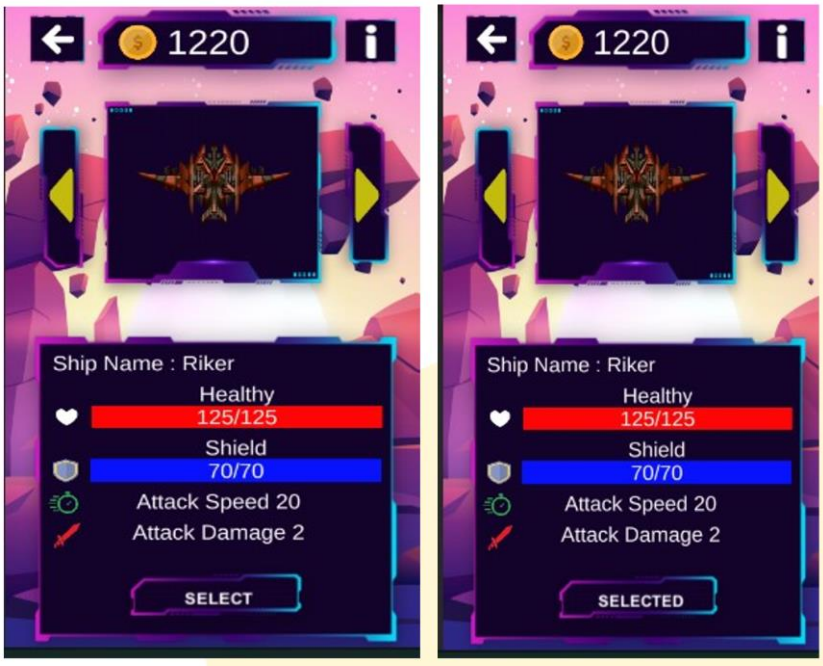

Рисунок Г.22 – Тестування результату вибору гравцем крейсеру в магазині

Тестування результату отримань підняття рівня крейсеру гравцем

Рисунок Г.23 – Тестування результату отримань підняття рівня крейсеру гравцем

Тестування результату регулювання звуку гравцем

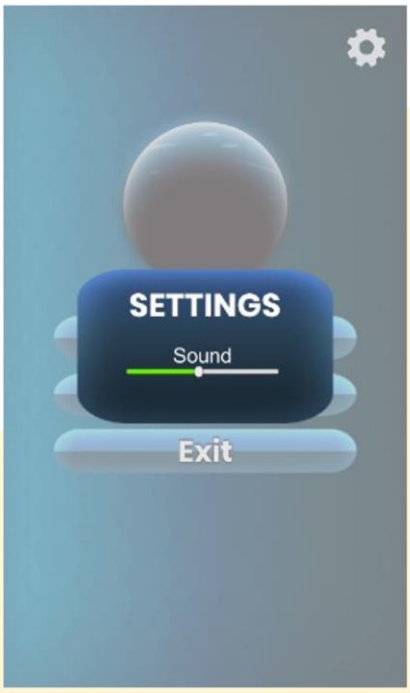

Рисунок Г.24 – Тестування результату регулювання звуку гравцем

Висновки

У бакалаврській дипломній роботі було зроблено:

- розроблено метод високорівневого оускладення гри та удосконалено його з полегшим створенням наступних рівнів;
- розроблено модель ігрової системи;
- розроблено маніпулятивну систему керування космічним крейсером за допомогою сенсорного екрану мобільного телефону;
- спроєктовано внутрішній ігровий магазин для полегшення проходження гри та його різноманітних можливостей гри;
- розроблено основні модулі ігрового додатку;
- спроєктовано зручний користувацький інтерфейс;
- розроблено та удосконалено графічні ефекти, об'єкти та прифаби для ігрового додатку;
- надано музичне супроводження під час гри чи під час знаходження в ігровому інтерфейсі;
- розроблено мобільний ігровий додаток «Space confrontation»;
- проведено тестування в програмному ігровому рушії та на мобільних пристроях ігрового додатку.

Рисунок Г.25 – Висновки

Апробації результату

- Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на Всеукраїнській науково-технічній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2022). Секція Інформаційні технології та комп'ютерна інженерія».

Наукова публікація :

- Войтко В.В. Особливості розробки мобільної шутер гри «SPACE CONFRONTATION» / В.В. Войтко, А. В. Денисюк, О. В. Гаврилюк, Н. Є. Барчук, Д. С. В. Самарасінгхе // Матеріали Всеукраїнської науково-практичної інтернет-конференції "Молодь в науці: дослідження, проблеми, перспективи - 2022", Секція - Інформаційні технології та комп'ютерна інженерія. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/viewFile/16192/13632

Рисунок Г.26 – Апробації та публікації