Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Ігровий програмний застосунок в жанрі Arcade на рушії Unity
(тема)

Виконав:
студент 4 курсу, групи ПЗПІпз-22-1

Сотников В.О.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст.викл. кафедри ПІ Олійник О.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Сотникову В’ячеславу Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Ігровий програмний застосунок в жанрі Arcade на рушії Unity
Затверджена наказом по університету від 17.06. 2024р. № 93 Стз.
2. Термін подання студентом роботи до екзаменаційної комісії 12.07.2024
3. Вихідні дані до роботи Розробити ігровий застосунок в жанрі Arcade на рушії Unity. Гра включатиме класичний геймплей Space Invaders, а також магазин скінів, систему оплати та систему нарахування очок для покращення взаємодії гравця з грою.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	07.05.2024	*виконано*
2	Створення специфікації ПЗ	10.05.2024	*виконано*
3	Проектування ПЗ	20.05.2024	*виконано*
4	Розробка ПЗ	23.05.2024	*виконано*
5	Тестування ПЗ	18.06.2024	*виконано*
6	Оформлення пояснювальної записки	27.06.2024	*виконано*
7	Підготовка презентації та доповіді	01.07.2024	*виконано*
8	Попередній захист	19.07.2024	*виконано*
9	Нормоконтроль, рецензування	19.07.2024	*виконано*
10	Здача роботи у електронний архів	19.07.2024	*виконано*
11	Допуск до захисту у зав. кафедри	19.07.2024	*виконано*

Дата видачі завдання 20 листопада 2023р.

Студент _____ Сотников В.О.
(підпис)

Керівник роботи _____ ст.викл. кафедри ПІ Олійник О.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 92 сторінок, 12 рисунків, 4 таблиць, 15 джерел.

Ключові слова: UNITY, C#, РОЗРОБКА ІГОР, АРКАДА, SPACE INVADERS, ВНУТРІШНІ ПОКУПКИ, МАГАЗИН СКІНІВ.

Об’єкт розробки – ігровий програмний застосунок в жанрі аркада Space Invaders на рушії Unity.

Мета розробки – створення ігрового програмного застосунку, який ефективно реалізує класичний жанр аркади Space Invaders, включаючи магазин скінів, систему оплати та систему нарахування очок.

Метод рішення – використання середовища розробки Unity, мови програмування C#, та інтеграція графічних ресурсів з Unity Store.

У результаті розробки створено ігровий програмний застосунок, що відтворює класичний геймплей Space Invaders, додаючи нові елементи, такі як магазин скінів, система оплати та нарахування очок, що розширює взаємодію гравця з грою та підвищує її привабливість.

Keywords: UNITY, C#, GAME DEVELOPMENT, ARCADE, SPACE INVADERS, IN-APP PURCHASES, SKIN SHOP.

Object of development – a game software application in the arcade genre Space Invaders on the Unity engine.

Purpose of development – to create a game software application that effectively implements the classic arcade genre of Space Invaders, including a skin shop, a payment system, and a scoring system.

Solution method – using the Unity development environment, the C# programming language, and integrating graphic resources from the Unity Store.

As a result of the development, a game software application was created that reproduces the classic Space Invaders gameplay, adding new elements such as a skin shop, payment system, and scoring system, which enhances player interaction with the game and increases its attractiveness.

Я, Сотников В’ячеслав Олександрович, студент гр. ПЗПІпз-22-1, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Ігровий програмний застосунок в жанрі аркада Space Invaders», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ГПЗ - Гральний програмний застосунок

AI - Аркада ігрова

SI - Space Invaders

Unity - Unity3D

МС - Магазин Скінів

СО - Система Оплати

App – Application (застосунок)

AI – Artificial Intelligence (штучний інтелект)

FPS – Frames Per Second (кадри в секунду)

GUI – Graphical User Interface (графічний інтерфейс користувача)

SDK – Software Development Kit (комплект засобів розробки програмного забезпечення)

API – Application Programming Interface (інтерфейс програмування застосунків)

AAB – Android App Bundle (формат файлів для Android)

Unity – середовище розробки ігор Unity

C# – мова програмування C#

UI – User Interface (користувацький інтерфейс)

ВСТУП

У наш час, коли технології швидко змінюються та вдосконалюються, розробка ігор залишається однією з найцікавіших та найдинамічніших галузей програмування. Графіка в комп'ютерних іграх та їхні можливості розширюються крок за кроком, відкриваючи нові можливості для розробників та надаючи користувачам неперевершений досвід.

Space Invaders - це легендарна аркадна відеогра, яка була випущена компанією Taito в 1978 році. Гра є однією з перших ігор, які здобули популярність в ігровій індустрії та стали класикою аркадних ігор.

Актуальність даної роботи полягає в створенні ігрового програмного застосунку в жанрі аркада Space Invaders на рушії Unity [1]. Гра Space Invaders є класичним представником аркадних ігор, що зарекомендувала себе як відмінний спосіб розважання та випробування реакції гравця.

Метою роботи є створення ігрового додатка, який передасть атмосферу оригіналу, та розширить геймплей за допомогою магазину скінів, системи оплати та нарахування очок, забезпечуючи унікальний інтерактивний досвід.

Завдання роботи включають в себе розробку ефективного геймплею, створення інтуїтивного інтерфейсу користувача, інтеграцію магазину скінів, системи оплати та системи нарахування очок. Для досягнення цих цілей використовуватимуться інструменти розробки Unity, мова програмування C#, а також графічні ресурси з Unity Store.

Галузь застосування результатів роботи охоплює галузь ігрової індустрії, та й може бути використана у сферах навчання, розваг та соціальної взаємодії. Ця робота є актуальною та перспективною у контексті постійного розвитку галузі ігор та комп'ютерних технологій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Предметною галуззю для нашої гри є жанр класичних аркадних ігор, а саме 2D-шутери, такі як легендарний "Space Invaders". Аркадні ігри з моменту своєї появи на ринку завоювали широку популярність завдяки простоті управління, динамічному геймплею та можливості змагань за найвищі результати. Основна ідея таких ігор полягає у швидкій реакції та стратегічному мисленні, де гравець повинен знищувати хвилі ворогів, уникаючи їх атак.

Важливими характеристиками жанру аркадних ігор є їх доступність та привабливість для широкої аудиторії [2]. Завдяки зрозумілому геймплею, аркадні ігри залучають як дітей, так і дорослих. Простий контроль, що часто зводиться до кількох кнопок або торкань екрану, робить ці ігри ідеальними для мобільних пристроїв, де інтуїтивний інтерфейс є ключовим фактором.

Історія жанру бере свій початок у 1970-х роках, коли були створені перші аркадні автомати. Гра "Space Invaders", випущена в 1978 році, стала однією з перших великих комерційних успіхів в цій галузі, встановивши стандарти для подальших розробок. Протягом десятиліть аркадні ігри еволюціонували, включаючи нові технології та механіки, але основні принципи динамічного та простого геймплею залишились незмінними.

На сьогоднішній день, мобільні пристрої є найпопулярнішою платформою для аркадних ігор завдяки своїй доступності та портативності. Гравці можуть насолоджуватися улюбленими іграми будь-де та будь-коли, що значно підвищує зручність використання. Крім того, розвиток мобільних технологій дозволив впровадити високоякісну графіку та звук, що робить ігровий досвід ще більш захоплюючим.

Створення гри "Space Invaders 2D" для мобільних телефонів є логічним продовженням традицій жанру аркадних ігор, адаптуючи їх до сучасних вимог ринку. Вона поєднує в собі класичний геймплей з новітніми технологіями, що дозволяє привернути увагу як нових гравців, так і тих, хто ностальгує за

класичними іграми минулих років. Таким чином, наша гра зможе зайняти своє місце серед сучасних мобільних аркадних хітів, зберігаючи дух традицій та впроваджуючи інновації.

1.2 Виявлення та вирішення проблем галузі

Ігрова індустрія, особливо сегмент мобільних аркадних ігор, стикається з рядом викликів, які можуть впливати на успіх нових проектів. Аналіз проблем, з якими зустрічаються розробники та гравці, дозволяє створити гру, що відповідає очікуванням користувачів і задовольняє їх потреби.

Проблеми галузі:

Однією з основних проблем є надмірна насичення ринку. Розробникам важко залучити увагу цільової аудиторії та виділити свою гру серед великої кількості ігор, доступних у магазинах додатків. Щоб отримати шанувальників, гра потребує значних зусиль у маркетингу та просуванні.

Монетизація також є великим викликом. Оскільки більшість гравців очікують, що мобільні ігри будуть безкоштовними, розробники змушені шукати альтернативні способи отримання доходу, такі як реклама або внутрішні покупки в іграх. Успіх гри залежить від балансу між ефективною монетизацією та позитивним користувачим досвідом.

Оскільки мобільні пристрої мають різні технічні характеристики та операційні системи, оптимізація продуктивності є ще однією важливою проблемою. Розробникам потрібно гарантувати, що ігри працюють добре на всіх підтримуваних платформах; це означає, що вони повинні провести ретельну тестування та налаштування.

Розробники повинні приділяти велику увагу кожному компоненту гри через високі вимоги до якості графіки, звуку та інтуїтивного керування, особливо щодо адаптації інтерфейсу до сенсорного керування на мобільних пристроях. Оскільки користувацький досвід має вирішальне значення для залучення та утримання гравців.

Вирішення проблем:

Ефективне просування гри допоможе виділити її серед конкурентів. Використання соціальних мереж, партнерства з впливовими особами (інфлюенсерами) та рекламні кампанії сприятимуть збільшенню впізнаваності гри. Регулярний моніторинг відгуків гравців і оперативне реагування на їхні потреби допоможе покращити гру та зміцнити лояльність користувачів.

Змішана модель монетизації, що поєднує рекламу та внутрішньоігрові покупки, може забезпечити стабільний дохід. Важливо, щоб реклама не була надто нав'язливою, а внутрішньоігрові придбання пропонували реальну цінність, не порушуючи баланс гри.

Проблеми з продуктивністю можна виявити та вирішити за допомогою інструментів профілювання та тестування Unity [3]. Підтримка різноманітних технічних характеристик і розмірів екранів гарантуватиме плавний ігровий процес для кожного користувача.

Ретельне опрацювання графіки, звуку та інтерфейсу сприятиме покращенню користувацького досвіду. Зворотний зв’язок від тестувальників і гравців допоможе вдосконалити керування та загальний ігровий процес, гарантуючи продукт високої якості.

Таким чином, виявлення та вирішення основних проблем мобільної ігрової індустрії дозволить створити конкурентоспроможну гру Space Invaders 2D, яка відповідатиме сучасним стандартам якості та задовольняє очікування гравців. Що приноситиме користь користувачам і надасть комерційний успіх, що є важливим для довгострокової популярності гри [4].

1.3 Постановка задачі

1.3.1 Цільова аудиторія

Цільова аудиторія для мобільної гри Space Invaders 2D включає різноманітні групи користувачів, що дозволяє охопити широке коло гравців завдяки класичному стилю та простому, але захоплюючому ігровому процесу.

Любителі класичних аркадних ігор з ностальгією ставляться до таких ігор, як оригінальний Space Invaders, цінуючи простоту геймплею та ретро-графіку, що нагадує про часи початкової ери відеоігор.

Молоді гравці, які шукають швидкі та захоплюючі ігри для мобільних пристроїв, віддають перевагу яскравій графіці, динамічному темпу та можливості провести час цікаво в короткі проміжки.

Казуальні гравці, які грають для розваги та розслаблення, без складних механік, шукають прості у виконанні, але цікаві ігри, які можна легко грати будь-де і будь-коли.

Шанувальники наукової фантастики зацікавлені в космічних пригодах та битвах з інопланетянами, цінуючи атмосферу гри та її сюжетну складову. Таким чином, гра повинна задовольняти потреби та інтереси різних груп гравців, зберігаючи баланс між складністю та доступністю.

1.3.2 Монетизація

Монетизація гри Space Invaders 2D буде здійснюватися через внутрішньоігрові покупки, зокрема через придбання гемів. Гравці зможуть купувати геми за реальні гроші, що створює основне джерело доходу для гри. Геми, в свою чергу, використовуватимуться для придбання золота, яке є внутрішньоігровою валютою, необхідною для подальших покупок. За золото гравці зможуть купувати різні скіни для своїх космічних кораблів, що додає елемент персоналізації та різноманітності в грі.

Скіни для літаків не тільки змінюватимуть зовнішній вигляд, але й матимуть різні атрибути, що покращують ігрові можливості, такі як посилена зброя, додаткові щити чи швидші двигуни. Це дозволяє гравцям не тільки виділитися серед інших, але й підвищити свої шанси на успіх у грі. Така система монетизації забезпечить стабільний дохід, дозволяючи гравцям насолоджуватися грою без нав'язливої реклами та зберігаючи баланс між безкоштовним і платним контентом.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Концепт-документ гри Space Invaders

Гравець керує літаком, який розташований внизу екрану і може рухатися лише вліво або вправо. Це обмеження надає грі особливий виклик, оскільки гравець повинен точно планувати свої дії для уникнення атак ворогів і снарядів.

Основними супротивниками у грі є інопланетяни, які рухаються з верху екрану донизу. Вони поділяються на різні типи з унікальними атрибутами, такими як кількість життя, швидкість руху та патерни поведінки. Деякі вороги можуть стріляти у відповідь, додаючи елемент небезпеки для гравця. Це вимагає від гравця постійної уваги та швидкої реакції, щоб уникнути їхніх атак і знищити їх до того, як вони досягнуть нижньої частини екрану.

Окрім звичайних ворогів, у грі періодично з'являються боси, які є значно сильнішими супротивниками. Вони з'являються на кожних 5000 очок, що гравець заробляє. Боси мають більше життя і часто мають спеціальні атаки, які ускладнюють їх знищення. Перемога над босами приносить значні нагороди у вигляді очок і золота, що мотивує гравців до подальшого проходження гри.

Гра передбачає три рівні складності. Після завершення кожного рівня складності з'являються нові типи ворогів з різними характеристиками та поведінкою. Це забезпечує постійне зростання викликів, що підтримує інтерес гравця і запобігає одноманітності геймплею.

Система очок у грі базується на кількості та типі знищених ворогів. Кожен тип ворога приносить різну кількість очок, що дозволяє гравцеві накопичувати бали для встановлення нових рекордів. Окрім очок, гравець отримує золото, яке можна використовувати для покращень та покупок всередині гри. Ця система винагород спонукає гравців до активного знищення ворогів та досягнення нових висот.

Гра є безкінечною і триває доти, доки гравець не втратить всі свої три життя. Це додає додатковий рівень напруженості, оскільки гравець постійно намагається продовжити своє виживання, уникаючи атак і знищуючи ворогів.

Така структура геймплею забезпечує високий рівень повторюваності гри, адже гравці постійно намагаються побити свої попередні рекорди і досягти нових успіхів.

Ці механіки разом створюють захоплюючий, напружений та динамічний геймплей, який тримає гравця в постійній напрузі. Унікальна комбінація різних типів ворогів, постійне підвищення складності та система винагород роблять гру привабливою і захоплюючою, що змушує гравців повертатися до неї знову і знову

2.2 Вимоги до графічного дизайну та скінів

Усі зображення в грі повинні бути високодеталізованими та сучасними, відходячи від ретро-стилістики минулих версій. Це означає, що кожен елемент, від корабля гравця до ворогів і босів, повинен бути чітко промальованим і реалістичним, з використанням сучасних технік 3D-графіки. Деталізація включає текстури високої роздільної здатності та складну анімацію, яка додає динамізму та реалістичності.

Візуальні ефекти мають створювати атмосферу гри. Ефекти вибухів, пострілів, знищення ворогів та появи босів повинні бути яскравими і видовищними, з використанням передових технологій освітлення та часток. Ці ефекти підкреслюватимуть динамічність подій, та й сприятимуть більш глибокому зануренню у геймплей. Важливо, щоб ці ефекти були добре збалансованими і не перенасичували екран, зберігаючи зручність для зору гравця.

Скіни для корабля та ворогів повинні бути різноманітними і стилізованими, пропонуючи гравцям можливість персоналізувати свій досвід гри. Кожен скін має бути унікальним і відповідати загальній тематиці гри, одночасно додаючи нові елементи дизайну, що роблять геймплей ще цікавішим. Наприклад, скіни можуть змінювати не лише зовнішній вигляд корабля, але й ефекти пострілів та вибухів, додаючи індивідуальності у гру кожного гравця.

Ергономічність графічного дизайну також має бути на високому рівні. Всі елементи інтерфейсу повинні бути інтуїтивно зрозумілими і легко розпізнаваними, що забезпечує швидке та зручне керування. Кольорова палітра

гри повинна бути добре збалансованою, використовуючи контрастні, але не агресивні кольори, щоб допомогти зосередитися на геймплеї і не напружують зір.

Саме тому, графічний дизайн та скіни у грі повинні відповідати сучасним стандартам якості, забезпечуючи високу деталізацію, яскраві та видовищні візуальні ефекти, можливість персоналізації і комфорт для гравців. Це зробить гру привабливою, захоплюючою та зручною для тривалих ігрових сесій.

2.3 Система оплати та нарахування очок в грі

Систему здобуття очок слід зробити так, щоб вона базувалася на знищенні ворогів. Кожен тип ворога має мати свою вартість в очках, залежно від складності його знищення. Наприклад, базові вороги повинні приносити меншу кількість очок, тоді як більш складні вороги і боси, що з'являються кожні 5000 очок, мають давати значно більше балів. Така система стимулюватиме гравців знищувати більше ворогів і досягати складніших цілей, щоб отримати більше очок.

Крім очок, за кожного знищеного ворога гравець повинен отримувати золото. Золото слід використовувати для внутрішньоігрових покупок, таких як нові скіни, покращення для корабля або спеціальні здібності. Це додає додатковий рівень стратегічного планування, оскільки гравець повинен вирішувати, як найкраще витратити зароблене золото для підвищення своїх шансів на успіх у грі.

Систему оплати слід реалізувати за моделлю free-to-play з можливістю внутрішньоігрових покупок [5]. Гравці повинні мати можливість безкоштовно завантажити та грати у гру, але також мати можливість купувати різні предмети за реальні гроші. Поки можна буде купляти віртуальну ігрову валюту, а саме – геми, які потім можна буде обміняти на золото, а вже за нього купляти скіни з різними характеристиками. В подальшому розвиту, в ці покупки можна включити функціональні поліпшення, що дозволятимуть прокачувати літак гравця та наприклад, збільшити кількість життів або отримати потужнішу зброю.

Важливо, щоб система була забезпечена балансом між можливістю заробити золото та очки в процесі гри і стимулюванням до покупок за реальні гроші [6]. Гра повинна забезпечувати достатньо можливостей для отримання

внутрішньоігрових ресурсів через геймплей, щоб підтримувати інтерес гравців і уникати відчуття необхідності витрачати реальні гроші для досягнення успіху.

В цілому, гра має поєднувати мотивацію досягати вищих результатів через здобуття очок і золота, а також надавати можливості для внутрішньоігрових покупок, що підвищують індивідуальний геймплей та дозволяють гравцям виразити свою унікальність і стиль [7].

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Розглянемо UML-діаграму для проєктування ПЗ гри "Space Invaders", яка ілюструє основні функції гри та взаємозв'язки між ними (див. рис. 3.1) [8].

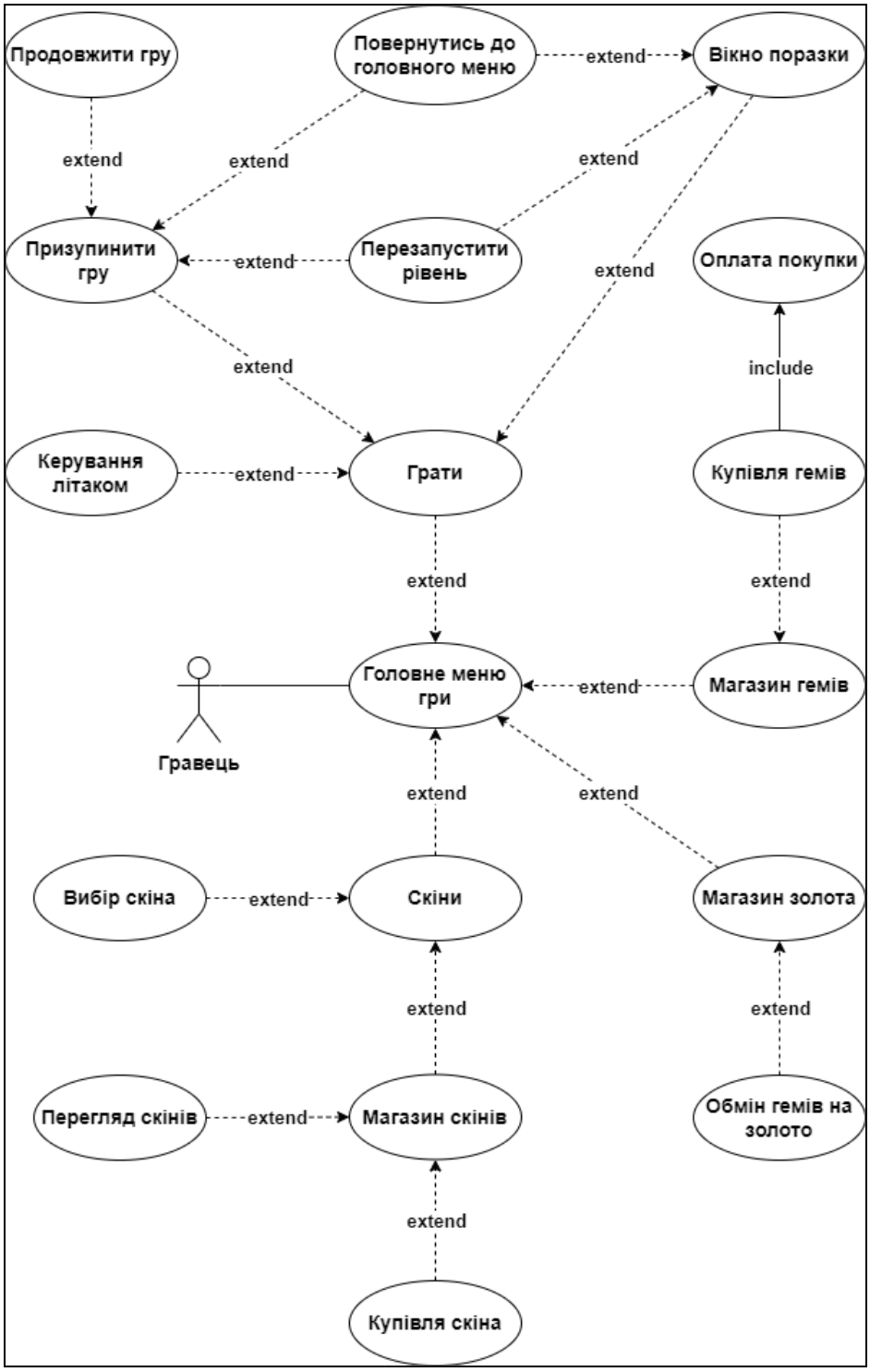


Рисунок 3.1 – UML діаграма варіантів використання

Користувачем даного ігрового застосунку є гравець. Гравець може почати взаємодію з грою через головне меню, де йому доступні кілька основних опцій. У головному меню гравець може обрати "Грати", щоб перейти до основного ігрового процесу, де він керує літаком, знищує ворогів і заробляє очки. Під час гри гравець має можливість призупинити гру, а потім продовжити її з того ж місця. У разі поразки гравець може перезапустити рівень або повернутися до головного меню, де відображається спеціальне "Вікно поразки".

Крім того, у головному меню він може вибрати опцію "Скіни", яка дозволяє переглядати, вибирати та купувати різні скіни для корабля. Гравець може спочатку переглянути доступні скіни, а потім вибрати та придбати бажаний скін у "Магазині скінів". Що забезпечує додаткову персоналізацію гри та робить її більш цікавою для користувача.

Ще однією можливою функцією головного меню є "Магазин гемів", де гравець може купувати геми за реальні гроші. Після вибору цієї опції, гравець проходить процес оплати, завершуючи його придбанням необхідної кількості гемів. Геми є внутрішньоігровою валютою, яку можна використовувати для різних покупок у грі, таких як нові скіни або поліпшення для корабля.

Також у головному меню є "Магазин золота", де гравці можуть обмінювати геми на золото. Золото використовується для внутрішньоігрових витрат і надає гравцю додаткові можливості для розвитку та покращення своїх ігрових можливостей. Процес обміну гемів на золото забезпечує гнучкість у використанні наявних ресурсів, дозволяючи гравцям ефективно планувати свої витрати.

3.2 Проєктування архітектури гри

Архітектура гри буде побудована на основі класів, які відповідатимуть за різні аспекти ігрового процесу. Всі класи можна розділити на кілька категорій: класи механіки гри, класи руху ворогів і їх атрибутів, класи стрільби, класи інтерфейсу, класи для гравця та класи спавну.

Класи механіки гри відповідають за основні правила та логіку ігрового процесу. Вони включають в себе управління ігровими об'єктами, зіткненнями,

підрахунком очок та оновленням стану гри. Це основні класи, які забезпечують безперервний ігровий процес і підтримують інтерактивність гри [9].

Класи руху ворогів та їх атрибутів контролюють поведінку ворогів на ігровому полі. Ці класи визначають патерни руху ворогів, їх швидкість, траєкторії та дії під час зіткнення з іншими об'єктами. Вони також включають атрибути, такі як кількість життів ворога та їхні спеціальні здібності.

Класи стрільби відповідають за механіку стрільби як з боку гравця, так і з боку ворогів. Вони включають в себе логіку запуску снарядів, їх руху та взаємодії з іншими об'єктами гри. Ці класи забезпечують різноманітність бойових дій і підвищують динаміку ігрового процесу.

Класи інтерфейсу відповідають за відображення ігрового меню, статусу гравця, нарахування очок, залишку життів та інших важливих параметрів. Вони забезпечують зручну взаємодію гравця з грою, надаючи необхідну інформацію в реальному часі.

Класи для гравця включають всі аспекти, пов'язані з управлінням ігровим персонажем. Вони відповідають за рух гравця, активацію спеціальних здібностей, взаємодію з об'єктами гри та управління літаком. Ці класи визначають, як гравець може впливати на ігровий процес і взаємодіяти з навколишнім середовищем.

Класи спавну контролюють появу нових ворогів та інших об'єктів на ігровому полі. Вони визначають частоту та місця появи нових елементів, забезпечуючи безперервність ігрового процесу та підтримуючи необхідний рівень складності.

3.3 Проєктування системи зберігання даних

Для гри слід передбачити механізм надійного збереження всіх важливих параметрів і прогресу гравця. Для цього використовуватимемо спеціальний об'єкт, відомий як "плеєр префаб".

Він забезпечуватиме зберігання даних про кількість зароблених монет, досягнуті рекорди та наявні геми. Цей об'єкт буде центральним місцем зберігання

всієї важливої інформації про гравця, що дозволить легко зберігати та завантажувати дані при кожному вході та виході з гри.

Для забезпечення надійності та цілісності даних, плеєр префаб включатиме механізми захисту від втрати прогресу гравця. Це реалізовано через регулярне автозбереження даних та можливість ручного збереження. Використання префабу спрощуватиме управління даними і їх обробку, оскільки всі параметри зберігатимуться в єдиному місці.

Заплановане зберігання даних у плеєр префабі дозволить гравцям продовжувати гру з того ж місця, де вони зупинилися, зберігаючи всі свої досягнення та накопичені ресурси. Це забезпечить безперервність ігрового процесу та зручність користування, роблячи гру більш привабливою та захоплюючою.

3.4 Приклади алгоритмів та методів

Для реалізації економічної системи в грі ми використовуємо клас Game, який управляє всіма аспектами, пов'язаними з монетами гравця [10].

```
using System;
using UnityEngine;

public class Game : MonoBehaviour
{
    public static Game Instance { get; private set; }
    public event Action<int> CoinsUpdated; // Подія оновлення монет

    private int coins;
    public int Coins
    {
        get { return coins; }
        set
        {
            coins = value;
            CoinsUpdated?.Invoke(coins); // Викликаємо подію після
оновлення монет
            SaveCoins(); // Збереження монет
        }
    }
    public bool HasEnoughCoins(int amount)
    {
        return (Coins >= amount);
    }
    public void AddCoins(int amount)
    {
        Coins += amount;
```

```
    }
    public void UseCoins(int amount)
    {
        if (HasEnoughCoins(amount))
        {
          Coins -= amount;
            SaveCoins();
        }
        else
        {
            Debug.LogWarning("Not enough coins!");
        }
    }
    private const string COINS_KEY = "Coins";

    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }
    }

    private void Start()
    {
        LoadCoins();
    }

    private void SaveCoins()
    {
        PlayerPrefs.SetInt(COINS_KEY, Coins);
        PlayerPrefs.Save();
    }

    private void LoadCoins()
    {
        Coins = PlayerPrefs.GetInt(COINS_KEY, 0);
    }
}
```

Центральним елементом цього класу є властивість Coins, яка дозволяє отримувати та встановлювати кількість монет. Коли кількість монет змінюється, викликається подія CoinsUpdated, що дозволяє іншим частинам гри реагувати на ці зміни. Метод SaveCoins відповідає за збереження поточного стану монет у сховищі гравця (PlayerPrefs), що дозволяє зберігати прогрес навіть після виходу з гри. Це досягається за допомогою збереження значення монет під ключем COINS_KEY.

Методи AddCoins і UseCoins забезпечують функціонал додавання та витрати монет відповідно. AddCoins просто збільшує кількість монет на задану величину, тоді як UseCoins спершу перевіряє, чи вистачає у гравця монет для витрати. Якщо монет достатньо, їх кількість зменшується, і знову ж таки викликається метод SaveCoins для збереження змін. У разі недостатньої кількості монет, виводиться попередження, що інформує гравця про відсутність необхідної кількості ресурсів.

Для реалізації руху гравця застосовується клас PlayerMovement, який використовує вхідні дані з джойстика для обчислення вектора руху.

```
public class PlayerMovement : MonoBehaviour
{
    public float speed = 5f;
    public Joystick joystick;

    void Update()
    {
        // Отримати вхід від джойстика
        float horizontalInput = joystick.Horizontal;

        // Рухатися вправо або вліво в залежності від вхідних даних з
джойстика
        Vector2 movement = new Vector2(horizontalInput, 0f);

        // Застосувати рух до позиції гравця
        transform.Translate(movement * speed * Time.deltaTime);
    }
}
```

У методі Update отримуються горизонтальні дані з джойстика, які використовуються для створення вектора руху. Потім цей вектор застосовується до позиції гравця за допомогою методу transform.Translate. Такий підхід дозволяє реалізувати плавний і чутливий рух гравця по екрану, що є критичним для динамічних ігор.

Клас PlayerSpawner відповідає за створення гравця на початку гри.

```
public class PlayerSpawner : MonoBehaviour
{
    public GameObject[] playerPrefabs; // Масив префабів гравців
    private const string SELECTED_SKIN_KEY = "SelectedSkinIndex";

    // Посилання на джойстик
    public Joystick joystick;

    public Image[] livesUI;
    public GameObject explosionPrefab;
    public GameObject explosionPlayerPrefab;
```

```
    public GameObject gameOverPanel;
    public PointManager pointManager;

    private void Start()
    {
        // Отримуємо індекс обраного скіна
        int selectedSkinIndex = PlayerPrefs.GetInt(SELECTED_SKIN_KEY, 0);

        // Перевіряємо індекс і вибираємо потрібний префаб гравця для
спавну
        if (selectedSkinIndex >= 0 && selectedSkinIndex <
playerPrefabs.Length)
        {
            // Спавнимо гравця з обраним префабом
            GameObject playerInstance =
Instantiate(playerPrefabs[selectedSkinIndex], transform.position,
Quaternion.identity);

            // Передаємо джойстик до гравця
            PlayerMovement playerMovement =
playerInstance.GetComponent<PlayerMovement>();
            if (playerMovement != null)
            {
                playerMovement.joystick = joystick;
            }
            else
            {
                Debug.LogWarning("Гравець не має компонента PlayerMovement
або Joystick");
            }
        }
        else
        {
            Debug.LogWarning("Індекс обраного скіна недійсний: " +
selectedSkinIndex);
        }
    }
}
```

У методі Start завантажується індекс обраного скіна з налаштувань гравця (PlayerPrefs). Якщо індекс валідний, вибирається відповідний префаб гравця, який створюється на сцені за допомогою методу Instantiate. Після цього джойстик передається новоствореному гравцю для управління рухом. Що забезпечує індивідуальний підхід до кожного гравця, дозволяючи обирати вигляд свого персонажа і зберігати ці налаштування між сеансами гри.

Для реалізації однієї із всіх наявних поведінок руху боса у хаотичному стилі, реалізовано клас BossChaoticMovement.

```
public class BossChaoticMovement : EnemyMove
{
    public float initialDropDistance = 5f;
    public float boundaryWidth = 10f;
```

```
    public float boundaryHeight = 5f;
   public float changeDirectionInterval = 2f; // Інтервал зміни напрямку
руху
    private bool initialDropComplete = false;
    private Vector2 currentDirection;

    protected override void Start()
    {
        base.Start();
        StartCoroutine(InitialDropRoutine());
    }

    private IEnumerator InitialDropRoutine()
    {
        // Спуск боса вниз на задану відстань
        while (transform.position.y > initialDropDistance)
        {
            transform.Translate(Vector2.down * speed * Time.deltaTime);
            yield return null;
        }
        initialDropComplete = true;
        StartCoroutine(ChangeDirectionRoutine());
    }

    protected override void Move()
    {
        if (initialDropComplete)
        {
            transform.Translate(currentDirection * speed * Time.deltaTime);
            ClampPosition();
        }
    }

    private void ClampPosition()
    {
        Vector3 pos = transform.position;
        pos.x = Mathf.Clamp(pos.x, -boundaryWidth, boundaryWidth);
        pos.y = Mathf.Clamp(pos.y, initialDropDistance - boundaryHeight,
initialDropDistance + boundaryHeight);
        transform.position = pos;
    }

    private IEnumerator ChangeDirectionRoutine()
    {
        while (true)
        {
            currentDirection = new Vector2(Random.Range(-1f, 1f),
Random.Range(-1f, 1f)).normalized;
            yield return new WaitForSeconds(changeDirectionInterval);
        }
    }
}
```

У методі Start викликається InitialDropRoutine, яка відповідає за початковий спуск боса на певну відстань. Вона продовжується виконуватися, поки бос не досягне заданого рівня. Після цього запускається ChangeDirectionRoutine, яка

періодично змінює напрямок руху боса. Кожен новий напрямок обирається випадковим чином, що додає елемент хаосу і непередбачуваності в поведінку боса.

Метод Move забезпечує рух боса в поточному напрямку, а метод ClampPosition обмежує позицію боса в межах визначених кордонів. Це гарантує, що бос не виходитиме за межі ігрового поля, підтримуючи баланс ігрового процесу. Така механіка робить боса складнішим противником, адже гравцю доводиться пристосовуватися до його непередбачуваних рухів.

Клас Purchase відповідає за обробку внутрішньоігрових покупок.

```
public class Purchase : MonoBehaviour
{

    public void OnPurchaseCompleted(Product product)
    {
        switch (product.definition.id)
        {
            case "space.inviders.50gems":
                AddGems(50);
                break;
            case "space.inviders.100gems":
                AddGems(100);
                break;
            case "space.inviders.150gems":
                AddGems(150);
                break;
            case "space.inviders.200gems":
                AddGems(200);
                break;
            case "space.inviders.500gems":
                AddGems(500);
                break;
            case "space.inviders.1000gems":
                AddGems(1000);
                break;
            case "space.inviders.5000gems":
                AddGems(5000);
                break;
            case "space.inviders.9999gems":
                AddGems(9999);
                break;
        }

    }
    private void AddGems(int gems)
    {
        int playerGems = PlayerPrefs.GetInt("gems");
        playerGems += gems;
        PlayerPrefs.SetInt("gems", playerGems);
        PlayerPrefs.Save();
        Debug.Log("Add gems " + gems);
```

```
            GemsUIInfo.instance.UpdateGemsText();
        }
    }
```

Метод OnPurchaseCompleted обробляє успішні транзакції і додає відповідну кількість гемів гравцю через метод AddGems. Цей метод збільшує кількість гемів у PlayerPrefs і зберігає зміни. Крім того, оновлюється інтерфейс, що відображає поточну кількість гемів. Такий підхід дозволяє інтегрувати систему покупок у гру, роблячи її більш зручною і привабливою для гравців.

3.5 Створення UI/UX та інтерфейсу гри

UI/UX розробка є невід'ємною частиною створення гри, що забезпечує зручність користування та приємний досвід для гравців. UI (User Interface) відповідає за візуальні елементи, з якими взаємодіє гравець, такі як кнопки, меню, іконки тощо. UX (User Experience) охоплює загальну взаємодію користувача з грою, включаючи легкість використання інтерфейсу, зручність навігації та задоволеність гравця від гри [11].

У грі основними елементами інтерфейсу є головне меню, ігровий екран, меню паузи та екран поразки. Всі ці елементи повинні бути добре спроектовані, щоб забезпечити інтуїтивне використання та високу естетичну привабливість.

Головне вікно гри (див. рис. 3.2) має космічний фон, що відповідає тематиці гри. На цьому вікні розташовано заголовок гри, який виконано великими, яскравими літерами для привертання уваги гравців. Під заголовком розміщена панель з кнопками, яка містить такі елементи: кнопка «Start», кнопка «Options», кнопка «Store», та кнопка «Exit». Кожна з цих кнопок стилізована відповідно до загального дизайну гри і забезпечує швидкий доступ до основних розділів гри.

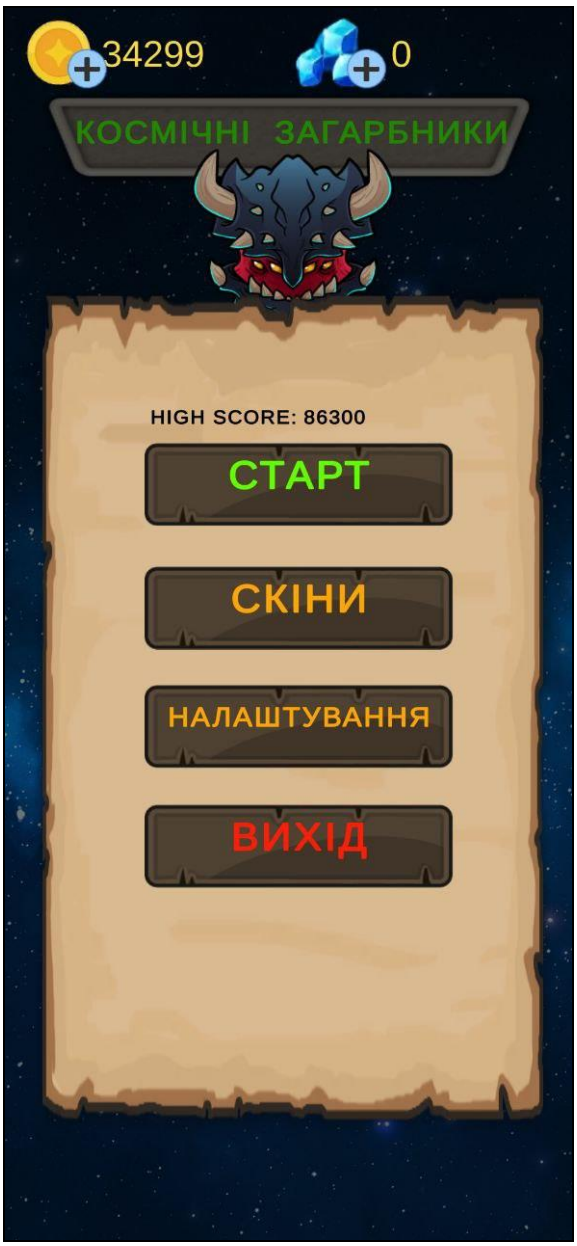


Рисунок 3.2 – Головне меню гри

Ігрове поле (див. рис. 3.3) є основним вікном, де відбувається геймплей. Воно також має космічний фон, на якому розташовані головні елементи гри: гравець, вороги та перешкоди. У верхній частині екрану розміщена інформаційна панель, яка містить лічильники очок, зароблені монети та індикатори життя гравця. Всі ці елементи виконані у стилістиці гри і допомагають гравцю швидко орієнтуватись у ситуації.

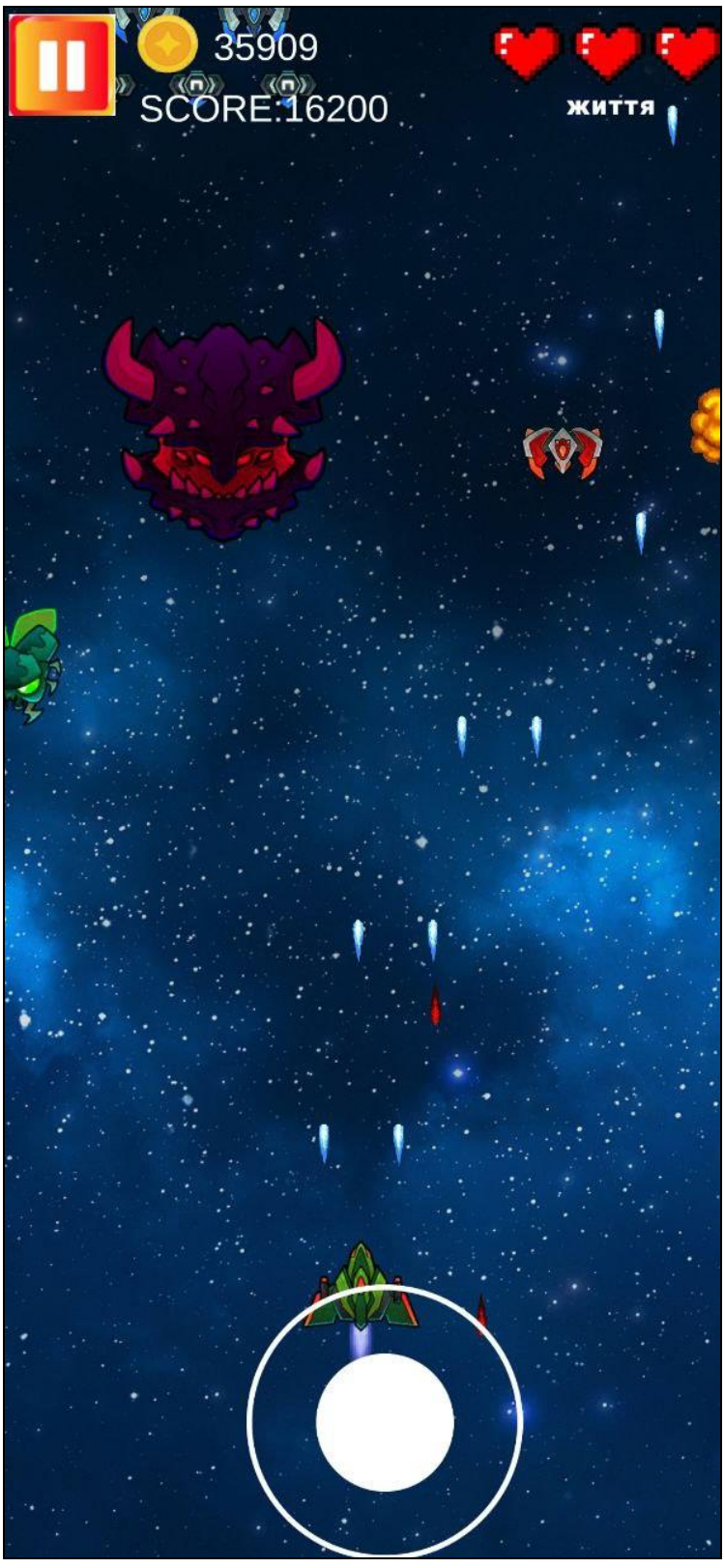


Рисунок 3.3 – Ігровий екран

Під час гри гравець може призупинити гру, натиснувши кнопку паузи, що викликає екран паузи (див. рис. 3.4). Тут відображаються кнопки для продовження гри, виходу в головне меню, та перезапуску рівня. Екран паузи

також оформлений у космічному стилі, з великими та чіткими кнопками для зручності користування.

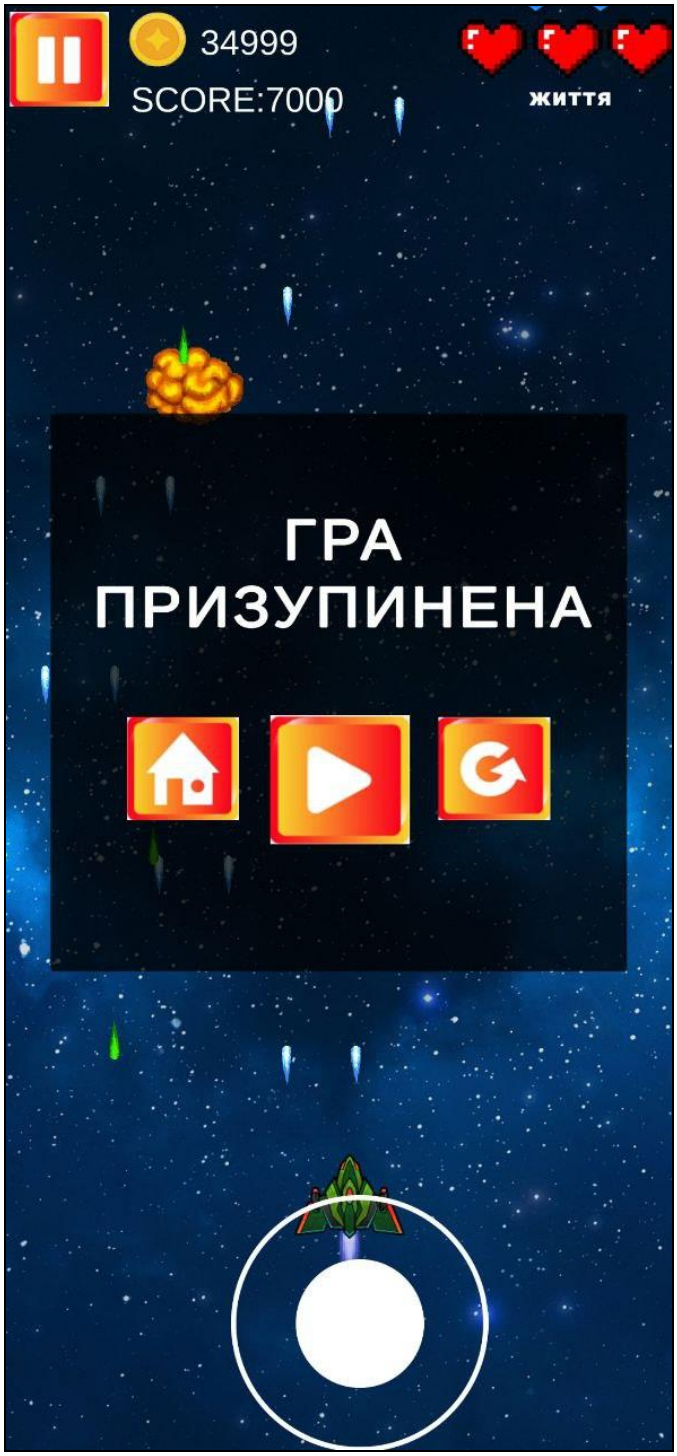


Рисунок 3.4 – Меню паузи

Після завершення рівня або гри в цілому, гравець потрапляє на екран завершення гри (див. рис. 3.5). Тут відображається загальний рахунок гравця, зароблені очки та кнопки для повторного проходження рівня або повернення до

головного меню. Екран завершення гри оформлений у єдиному стилі з іншими елементами інтерфейсу.

Рисунок 3.5 – Меню поразки

Окрім візуальних елементів, значну роль у створенні позитивного користувацького досвіду відіграють звукові ефекти та музика. Вони підсилювать візуальні ефекти, створюючи атмосферу гри. У грі використовується фонова музика, яка грає під час геймплею.

Елементи UX інтерфейсу швидко реагують на дії гравця, забезпечуючи плавні анімації та миттєвий зворотний зв'язок. А саме, це включає відгук на натискання кнопок, плавні переходи між екранами та інші інтерактивні елементи, що роблять взаємодію з грою більш приємною та інтуїтивною.

Всі елементи інтерфейсу, включаючи меню та геймплей, анімовані плавно, щоб забезпечити комфортне користування. Перехід між різними екранами гри відбувається без ривків, що покращує загальне враження від гри.

В грі використовується адаптивний дизайн інтерфейсу, який гарантує, що гра буде виглядати та функціонувати однаково добре на різних пристроях – від смартфонів до планшетів та великих екранів.

Описані елементи інтерфейсу забезпечують гравцям інтуїтивно зрозумілий і приємний досвід взаємодії з грою. Всі компоненти виконані у єдиному стилі, що створює цілісне враження і допомагає гравцям легко орієнтуватися у грі.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Розробка магазину скінів та графічних елементів

Введення системи скінів у гру додасть новий рівень різноманіття та глибини до ігрового процесу. Кожен скін, окрім унікального зовнішнього вигляду, має свої специфічні характеристики, які впливають на геймплей. Ці характеристики включають різну швидкість руху, швидкість стрільби та тип пострілів, що надає гравцям можливість вибору скінів, які найбільше підходять до їх стилю гри та стратегій.

Скіни представлені у вигляді 2D картинок, які відповідають загальній стилістиці гри. Кожен скін має свої унікальні графічні елементи, які додають візуального різноманіття. Обов’язковою технічною характеристикою кожного скіна є наявність 2D ріджед баді та бокс колайдерів, що забезпечують правильну фізику взаємодії з оточуючим середовищем та іншими об'єктами у грі. Що дозволяє скінам органічно вписуватися в ігровий процес, забезпечуючи реалістичні зіткнення та переміщення [12].

Також, скіни оснащені скриптами керування і певними патернами стрільби, що визначають його поведінку у грі. Ці скрипти регулюють швидкість руху, тип та частоту пострілів, надаючи кожному скіну унікальні ігрові характеристики. Скіни зберігаються у вигляді префабів, і це дозволяє легко підвантажувати необхідний скін під час гри. Це все забезпечує високу гнучкість та можливість швидкої зміни зовнішнього вигляду гравця без переривання ігрового процесу.

На рисунку 4.1 представлено меню вибору куплених скінів. Після покупки, користувачі можуть переглядати свої скіни у спеціальному списку, де кожен елемент відображається зображенням та коротким описом характеристик (див. рис. 4.1). Гравець може легко перемикатися між доступними скіном, щоб знайти той, який найбільше відповідає його поточним потребам у грі.

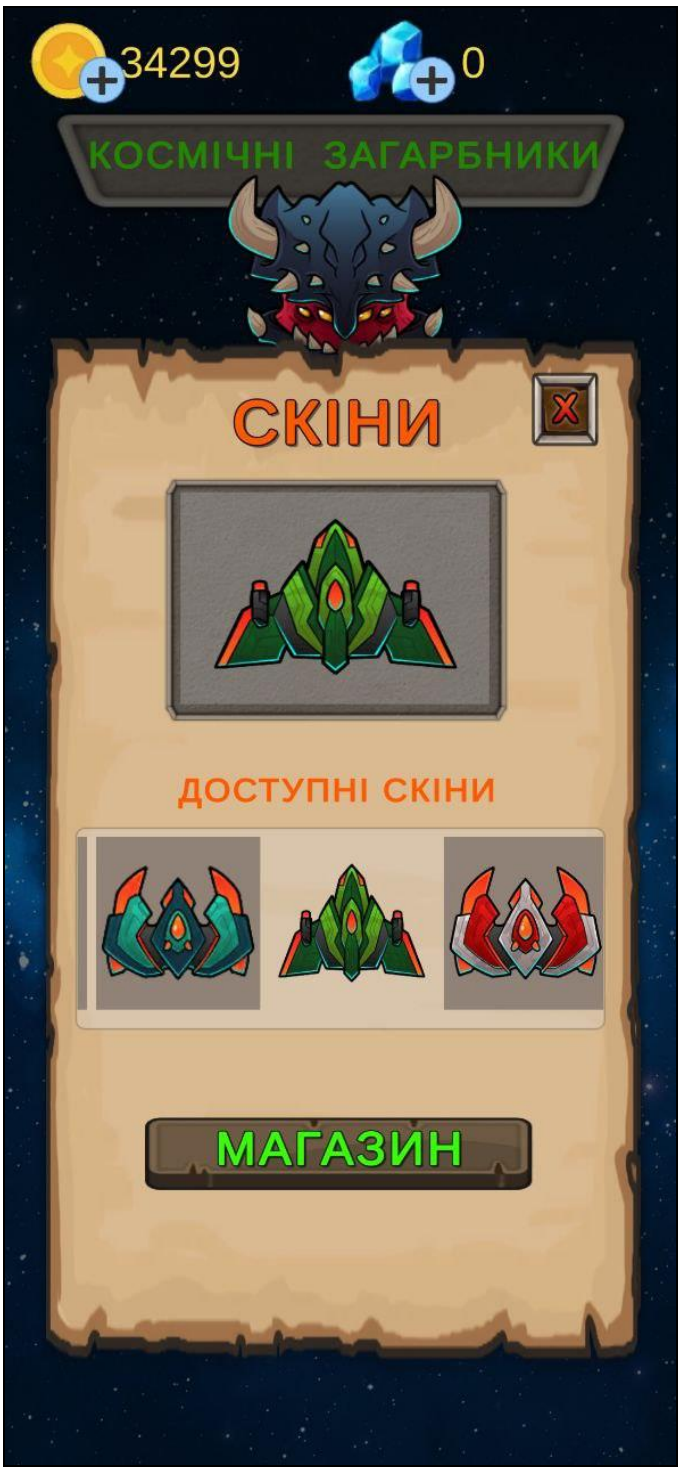


Рисунок 4.1 – Список куплених скінів

Для покупки скінів у грі передбачено спеціальний магазин, як показано на рисунку 4.2. Усі скіни продаються за внутрішню валюту – золоті монети, які гравець може заробити під час гри або придбати через систему внутрішніх покупок. Магазин надає можливість гравцям переглядати доступні скіни, ознайомлюватися з їх цінами. Цей підхід мотивує гравців грати в гру, збираючи

монети для купівлі більш кращого літака та дозволяє монетизувати гру, щоб пришвидчити розвиток свого ігрового прогресу.

Рисунок 4.2 – Магазин скінів

4.2 Реалізація системи оплати

Для монетизації гри Space Invaders було використано підхід покупки ігрової валюти, а саме гемів [13]. Гравці можуть придбати геми за реальні гроші, а потім обмінювати їх на золото. Золото, у свою чергу, використовується для купівлі нових скінів для літачків. Такий підхід дозволяє гравцям швидко й зручно отримувати необхідні ресурси, зберігаючи інтерес до гри та підтримуючи її розвиток.

Монетизація була реалізована з використанням In-App Purchases (IAP). Ця технологія дозволяє інтегрувати платіжні системи безпосередньо в гру, що значно спрощує процес купівлі для користувачів. Спочатку було встановлено відповідну бібліотеку для роботи з IAP. Далі на кнопках покупки було налаштовано спеціальні події кліку, які викликають скрипт для здійснення транзакції. Скрипт перевіряє успішність оплати через API. У разі успішної оплати, користувач отримує відповідну кількість гемів на свій рахунок.

На рисунку 4.3 показано інтерфейс магазину гемів. У цьому магазині користувач може обрати необхідну кількість гемів для покупки. Всі пропозиції гемів відображаються у вигляді зручного списку з відповідними цінами. Кожен елемент списку містить інформацію про кількість гемів та їх вартість у реальних грошах, що дозволяє гравцю легко вибрати оптимальний для себе пакет (див. рис. 4.3).

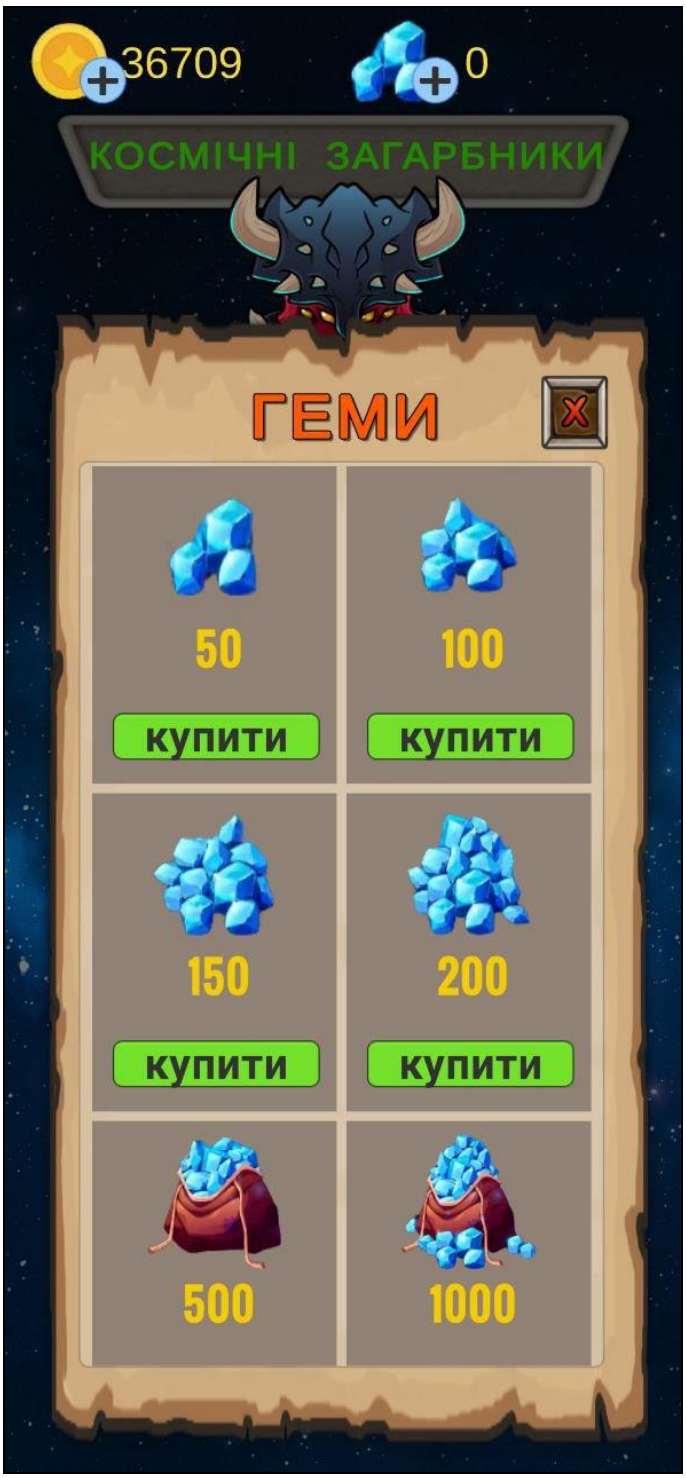


Рисунок 4.3 – Магазин гемів

На рисунку 4.4 показано процес покупки 50 гемів. Після натискання кнопки купівлі відкривається панель Google Play, де гравець може обрати спосіб оплати (див. рис. 4.4). Ця панель дозволяє обрати платіжну карту або інший метод оплати, що є зручним для користувачів, оскільки більшість з них вже мають

налаштовані облікові записи в Google Play [14]. Після вибору методу оплати здійснюється сама транзакція.

Рисунок 4.4 – Купівля 50 гемів, вибір карти

Дальше, як показано на рисунку 4.5, після успішної оплати гравець отримує повідомлення про успішну оплату покупки. Сам процес нарахування гемів

спрацьовує миттєво, що робить покупку швидкою та зручною. Після зарахування гемів гравець може перейти до магазину золота, де відбувається обмін гемів на золото. У магазині золота (див. рис. 4.6), гравець може вибрати необхідну кількість золота відповідно до наявних гемів і здійснити обмін.

Рисунок 4.5 – Купівля 50 гемів, сплачено

Рисунок 4.6 – Магазин золота за геми

Всі транзакції та історію покупок можна переглянути у Google Play Console, де зберігається детальна інформація про всі здійснені покупки іншими гравцями. Це дозволяє розробникам стежити за активністю гравців та контролювати

фінансові операції. Крім того, за допомогою Google Play Console можна вивести зароблені кошти на власну платіжну карту, виконавши всі вимоги платформи. Що забезпечує прозорість фінансових потоків та спрощує управління доходами від гри.

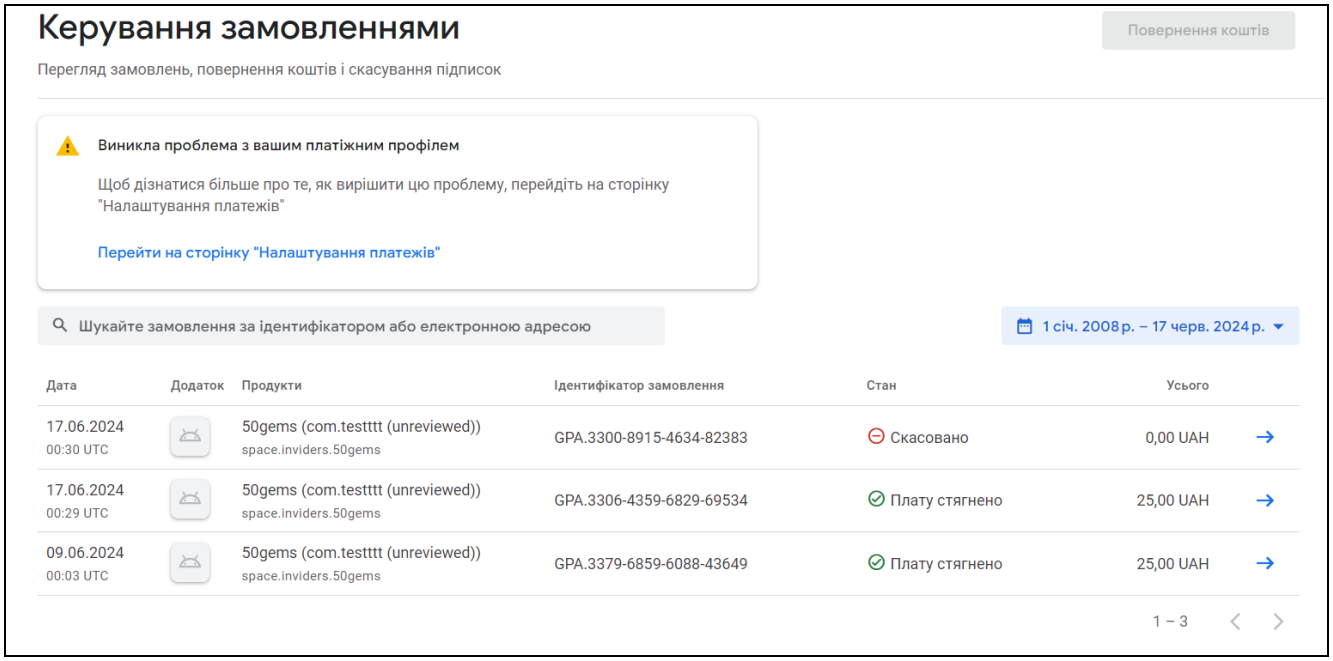


Рисунок 4.7 – Історія покупок в грі

4.3 Логіка нарахування очок в грі

У грі передбачена система нарахування очок, яка підтримує інтерес гравців. Очки нараховуються за знищення ворогів, що додає елемент змагання і мотивує гравців досягати нових рекордів. Гра має безкінечний рівень, де гравець постійно намагається перевершити свій попередній результат, знищуючи дедалі більше ворогів.

Вороги у грі поділяються на декілька типів: від простих до босів. Кожен тип ворога приносить різну кількість очок залежно від його складності та характеристик. За вбивство простого ворога гравець отримує від 50 до 100 очок, тоді як за знищення босів нараховується від 100 до 500 очок, що залежить від складності боса. Така система стимулює гравців атакувати різні типи ворогів для максимізації своїх очок.

Окрім очок, гравці також отримують золоті монети за знищення ворогів. Ці монети можна використовувати для внутрішньоігрових покупок, а саме, купівлі

нових скінів. Прості вороги приносять від 5 до 15 золотих монет, а боси — від 25 до 100 монет. Це додає додатковий рівень стратегічного планування, оскільки гравець вирішує, як найкраще витратити зароблене золото для підвищення своїх шансів на успіх у грі.

Таким чином, система нарахування очок та монет не лише робить гру більш захоплюючою, але й стимулює гравців досягати нових висот, знищуючи більше ворогів та постійно покращуючи свої результати. Очки стають своєрідним рекордом, до якого прагнуть усі гравці, що сприяє тривалому інтересу до гри.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тестування ігрового застосунку

Тестування ігрового застосунку проводилось у динамічному режимі, що передбачало ручну перевірку всіх функцій гри. Це забезпечило ретельне тестування кожного аспекту гри, зокрема механіки гри, логіки нарахування очок, поведінки ворогів, а також інтерфейсу користувача. Даний підхід дозволив виявити й усунути баги, які могли б залишитися непоміченими під час автоматичного тестування [15].

Результати тестування подані у вигляді тест-кейсів, які детально описують перевірку окремих функцій гри. Кожен тест-кейс включає інформацію про ідентифікатор тесту, опис функції, власника тесту, дату створення та мету тесту. Вони також містять передумови, опис конкретних випадків тестування, очікувані результати та висновки щодо їх проходження.

Таблиця 5.1 – Тест-кейс №1

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Опис функції:	Логіка нарахування очок за знищення ворогів		
Власник тесту:	Сотников		
Дата створення:	18.06.2024		
Мета тесту:	Перевірити коректність нарахування очок за знищення різних типів ворогів		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити гру	Гра успішно завантажується і готова до гри	Пройдено
2	Розпочати нову гру	Гра починається з початковими умовами	Пройдено
Нарахування очок			
№	Опис випадку	Очікуваний результат	Висновок
1	Знищити простого ворога	Гравець отримує від 50 до 100 очок	Пройдено
2	Знищити декілька видів босів	Гравець отримує від 100 до 500 очок залежно від складності	Пройдено

Таблиця 5.2 – Тест-кейс №2

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Опис функції:	Здійснення покупки гемів		
Власник тесту:	Сотников		
Дата створення:	18.06.2024		
Мета тесту:	Перевірити коректність процесу купівлі 50 гемів у грі		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити гру	Гра успішно завантажується і готова до гри	Пройдено
2	Перейти до магазину гемів	Гравцю відкривається магазин гемів з різними пропозиціями	Пройдено
Здійснення покупки гемів			
№	Опис випадку	Очікуваний результат	Висновок
1	Натиснути на плюсик з іконкою гемів	Відкривається меню покупки гемів	Пройдено
2	Вибрати опцію покупки 50 гемів	Гравець вибирає покупку 50 гемів	Пройдено
3	Вибрати карту з панелі Google Play	Відкривається інтерфейс вибору карти оплати	Пройдено
4	Підтвердити оплату	Відбувається підтвердження оплати	Пройдено
5	Завершити покупку	50 гемів успішно нараховано на аккаунт гравця	Пройдено

5.2 Приклади критичних помилок

У процесі тестування програмного забезпечення гри було виявлено декілька критичних помилок, які значно впливають на ігровий процес і досвід користувача. Нижче наведено приклади таких помилок з детальним описом.

Помилка 1: Некоректне нарахування гемів при покупці

Опис помилки: При здійсненні покупки 50 гемів користувачеві нараховується 150 гемів, тобто процес нарахування виконується тричі.

Вплив: Це порушує баланс гри і може призвести до значних втрат доходу.

Кроки для відтворення:

1. Відкрити гру.
2. Натиснути на плюсик з іконкою гемів.

3. Вибрати опцію покупки 50 гемів.
4. Вибрати карту з панелі Google Play.
5. Підтвердити оплату.
6. Після завершення покупки нараховується 150 гемів замість 50.

Помилка 2: Втрата прогресу після оновлення гри

Опис помилки: Після оновлення гри гравці втрачають весь свій прогрес, включаючи рівні, досягнення та внутрішньоігрові покупки.

Вплив: Це призводить до негативного досвіду користувачів та може спричинити втрату гравців.

Кроки для відтворення:

1. Гравець досягає певного прогресу в грі.
2. Гра оновлюється до нової версії.
3. Після оновлення весь прогрес гравця втрачається.

Помилка 3: Некоректне відображення інтерфейсу на деяких пристроях

Опис помилки: Інтерфейс гри відображається некоректно на деяких моделях смартфонів, що ускладнює або робить неможливим доступ до певних функцій.

Вплив: Це ускладнює взаємодію користувачів з грою і може призвести до відмови від гри.

Кроки для відтворення:

1. Відкрити гру на моделі смартфона з проблемами відображення.
2. Інтерфейс відображається некоректно (наприклад, кнопки перекриваються або виходять за межі екрану).

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Визначення плану впровадження

Впровадження програмного забезпечення для ігрового мобільного застосунку вимагає чітко визначеного плану, який забезпечить успішний запуск та стабільну роботу гри. Сформуємо підходи для впровадження гри на платформі Google Play Market, а також детально опишемо етапи цього процесу.

Таблиця 6.1 – План впровадження ігрового мобільного застосунку

Назва етапу	Дата початку	Дата завершення
Внутрішнє тестування	01.07.2024	07.07.2024
Підготовка до закритого тестування	08.07.2024	10.07.2024
Закрите тестування	11.07.2024	13.07.2024
Підготовка до виходу в «Production»	14.07.2024	14.07.2024
Вихід в «Production»	15.07.2024	16.07.2024

Першим етапом є внутрішнє тестування, яке передбачає перевірку всіх функціональних можливостей гри командою розробників та тестувальників. В ньому важливо виявити та усунути всі критичні помилки до переходу на наступні етапи. Внутрішнє тестування забезпечить початковий рівень якості продукту, який відповідає вимогам специфікацій і готовий до закритого тестування.

Наступним етапом є підготовка до закритого тестування, що включає налаштування серверів, створення спеціальних облікових записів для тестувальників та підготовку необхідної документації. Під час цього етапу також проводиться навчання тестувальників щодо використання інструментів тестування та методик виявлення помилок. Після завершення підготовчих робіт можна переходити до закритого тестування.

Закрите тестування передбачає участь обмеженого кола користувачів, які тестують гру в умовах, максимально наближених до реальних. Це дозволяє виявити потенційні проблеми, що можуть виникнути під час експлуатації гри

кінцевими користувачами. На цьому етапі збирається зворотний зв'язок від тестувальників, який аналізується для подальшого вдосконалення продукту.

Після закритого тестування відбувається підготовка до виходу в "Production". Цей етап включає остаточне тестування всіх компонентів гри, створення резервних копій та підготовку релізних нотаток. Проводиться перевірка відповідності гри вимогам Google Play Market, включаючи політики безпеки та конфіденційності.

Заключним етапом є вихід в "Production", що передбачає публікацію гри в Google Play Market. Після цього гра стає доступною для завантаження та використання всіма користувачами. Важливо забезпечити оперативну підтримку користувачів та моніторинг гри для виявлення та швидкого усунення можливих проблем.

6.2 Впровадження в Google Play

Впровадження гри відбувалось поетапно. Спершу було зареєстровано обліковий запис у Google Play Console. Під час реєстрації було введено всю необхідну інформацію та сплачено реєстраційний збір у розмірі 25 доларів США. Після цього розробник отримав доступ до всіх інструментів та сервісів, необхідних для публікації та керування грою.

Наступним кроком було створення нового проекту в Google Play Console. Увійшовши в консоль, було вибрано опцію "Створити новий застосунок". Введено основну інформацію про гру, зокрема її назву, мову та короткий опис. Дана інформація є важливою, оскільки вона буде відображатися в магазині Google Play та допомагати потенційним користувачам знайти і зацікавитись грою.

Після створення проекту було підготовлено гру до компіляції у форматі Android App Bundle (AAB). Цей формат забезпечує оптимізацію розміру завантажуваного файлу та покращує продуктивність гри. Після компіляції AAB-файлу його було ретельно протестовано, щоб переконатися у відсутності помилок та стабільній роботі гри.

Далі було завантажено AAB-файл через розділ "Релізи" в Google Play Console. Для цього було вибрано опцію "Керувати продуктивними релізами" та завантажено підготовлений файл. У цьому ж розділі було зазначено додаткову інформацію про версію гри, включаючи номер версії та нотатки про зміни.

Одночасно з цим було підготовлено маркетингові матеріали для сторінки гри в Google Play. Сюди входили іконка застосунку, скріншоти, промо-відео та банер. Усі ці елементи створювалися з урахуванням вимог Google щодо розмірів та якості, щоб забезпечити їх привабливість для користувачів та надання повної інформації про гру.

Після чого було налаштування категорій та рейтингів гри. Для цього було обрано відповідну категорію, що полегшує користувачам пошук гри в магазині. Також було встановлено віковий рейтинг за допомогою інструмента для оцінки контенту, що допомагає визначити відповідність змісту гри певній віковій групі.

Після завершення всіх підготовчих етапів гра була готова до перевірки та модерації. Google перевіряє кожен застосунок перед його публікацією, щоб переконатися у відповідності всім вимогам та політикам Google Play. Цей процес може зайняти кілька днів. Після успішного проходження модерації гра стала доступною для завантаження користувачами.

Після публікації гри в Google Play важливо постійно моніторити її роботу та відгуки користувачів. Використовуючи інструменти Google Play Console, розробники можуть аналізувати продуктивність гри, виявляти помилки та отримувати зворотний зв'язок від користувачів. Щоб оперативно вносити необхідні зміни та покращувати гру на основі відгуків.

ВИСНОВКИ

В процесі реалізації гри було проведено детальний аналіз предметної галузі, розроблено концепт-документ, спроєктовано архітектуру програмного забезпечення, а також здійснено тестування ігрового застосунку. Метою роботи було створення аркадної гри у жанрі Space Invaders на платформі Unity, включаючи додаткові функціональні можливості, такі як магазин скінів, система нарахування очок та система оплати.

Розробка гри почалася з аналізу предметної галузі, що дозволило визначити основні вимоги до гри та цільову аудиторію. Було виявлено, що основними користувачами гри будуть шанувальники класичних аркадних ігор, тому основний акцент робився на збереження традиційного ігрового процесу Space Invaders з додаванням сучасних елементів. Постановка задачі включала розробку механіки гри, графічного дизайну, а також системи нарахування очок, що стимулює гравців до досягнення нових рекордів.

Концепт-документ гри включав детальний опис геймплею, вимог до графічного дизайну та функціональних можливостей. Було розроблено кілька видів ворогів, включаючи босів, які надають більше очок та монет за їх знищення. Це додало стратегічного елемента в гру, оскільки гравці повинні обдумано витрачати зароблені ресурси на покращення своїх можливостей.

Проєктування архітектури програмного забезпечення виконувалося з використанням UML-діаграм, що дозволило створити чітку структуру коду та забезпечити легкість у підтримці та масштабуванні гри. Основна увага приділялася створенню інтуїтивно зрозумілого інтерфейсу користувача, що дозволило зробити гру доступною для широкої аудиторії.

Опис прийнятих програмних рішень включав розробку магазину скінів, систему оплати та логіку нарахування очок. Магазин скінів дозволяє гравцям купувати нові графічні елементи за внутрішньоігрову валюту, що заробляється під час гри. Система оплати була інтегрована з Google Play, що забезпечило зручність для користувачів у здійсненні покупок.

Тестування програмного забезпечення проводилося у динамічному режимі, включаючи ручну перевірку всіх функцій гри. Що дозволило виявити та усунути баги, що могли б залишитися непоміченими під час автоматичного тестування. Результати тестування були оформлені у вигляді тест-кейсів, що описують перевірку окремих функцій гри.

Після завершення всіх підготовчих етапів гра була готова до перевірки та модерації у Google Play. Процес публікації включав перевірку на відповідність вимогам та політикам Google, що забезпечило високу якість кінцевого продукту. Після успішної модерації гра стала доступною для завантаження користувачами.

Одже, реалізація гри Space Invaders на платформі Unity дозволила створити високоякісний продукт, що відповідає сучасним вимогам ігрової індустрії. Впровадження системи нарахування очок та магазину скінів додало додаткового інтересу до гри та стимулювало гравців до активної взаємодії з ігровим світом. Постійний моніторинг відгуків користувачів та оперативне внесення змін дозволяють покращувати гру та підтримувати її популярність серед гравців.

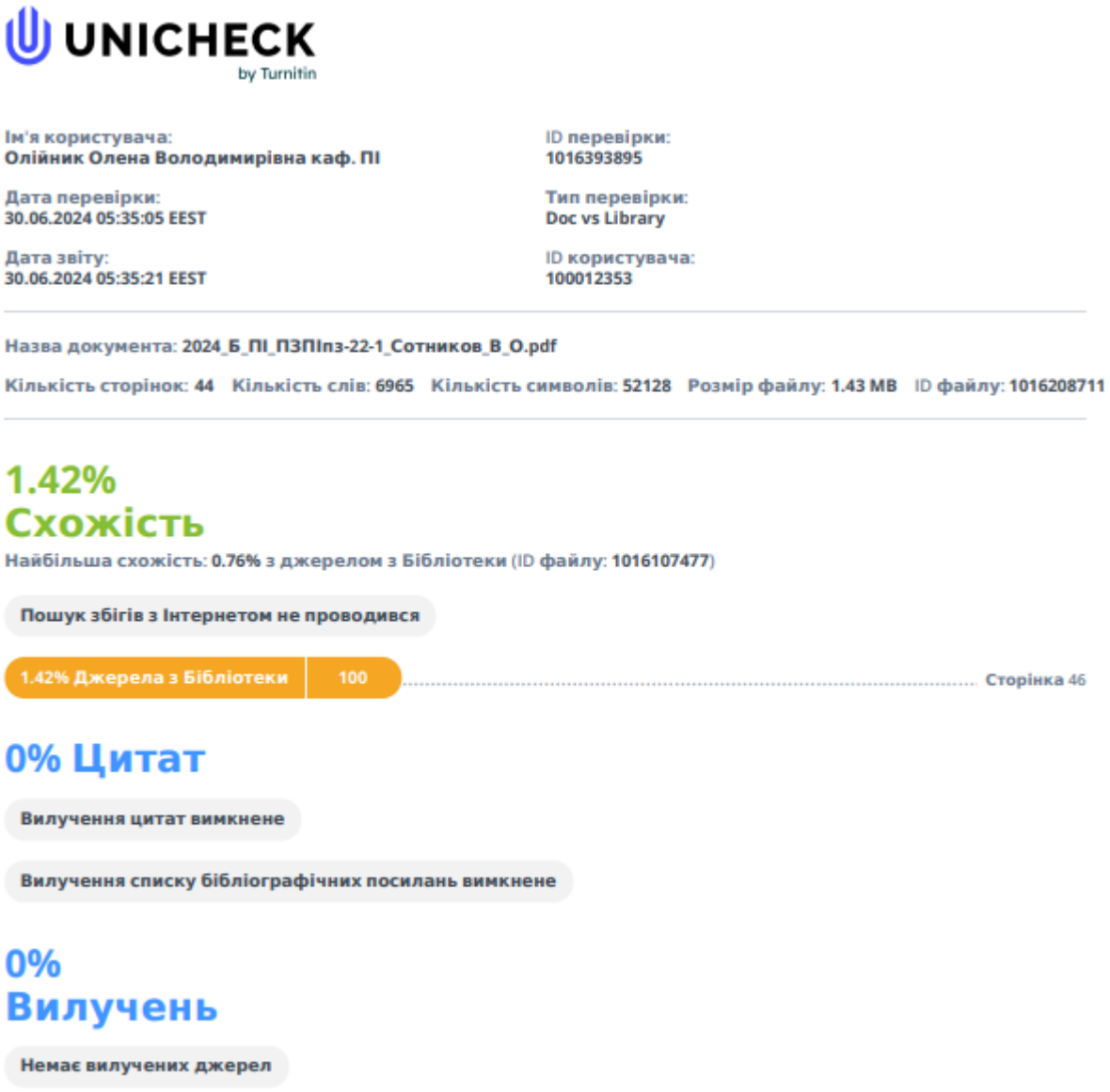
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Коваленко І. П. Розробка комп'ютерних ігор на платформі Unity: навч. посіб. – Київ: Вид-во Київського університету, 2018. – 350 с.
2. Андрющенко О. М. Основи геймдизайну: теорія та практика. – Харків: ХНУ імені В. Н. Каразіна, 2019. – 275 с.
3. Ляшенко В. О., Чернов А. В. Програмування на C# для Unity: навч. посіб. – Одеса: ОНПУ, 2017. – 312 с.
4. Савченко Н. К. Моніторинг та аналіз відгуків користувачів мобільних додатків. – Ужгород: УжНУ, 2021. – 190 с.
5. Горбачук П. І. Мобільні ігри та монетизація: сучасні підходи та стратегії. – Дніпро: Вид-во Дніпропетровського університету, 2018. – 198 с.
6. Шаповалов І. С. Основи маркетингу мобільних додатків. – Київ: Вид-во КНТЕУ, 2021. – 220 с.
7. Іванченко В. Г. Технології розробки інтерактивних ігор. – Чернівці: ЧНУ імені Ю. Федьковича, 2019. – 290 с.
8. Дубровський М. О. UML для проєктування програмних систем: навч. посіб. – Полтава: Полтавський технічний університет, 2020. – 260 с.
9. Ковальчук М. І., Стеценко Л. П. Основи розробки комп'ютерних ігор на Unity. – Вінниця: ВНТУ, 2018. – 310 с.
10. Петренко В. С. Алгоритми та методи у розробці ігор. – Запоріжжя: ЗНУ, 2020. – 275 с.
11. Мельник А. П., Кравець І. В. Графічний дизайн в ігровій індустрії. – Рівне: РДГУ, 2018. – 230 с.
12. Пилипчук О. Д. Інтерфейси користувача: дизайн та оцінка. – Тернопіль: ТНТУ, 2017. – 200 с.
13. Ткаченко О. Р. Системи оплати в мобільних іграх. – Житомир: ЖДУ імені Івана Франка, 2019. – 210 с.
14. Яковенко О. В. Використання платформи Google Play для розповсюдження мобільних додатків. – Суми: СумДУ, 2019. – 185 с.

15. Зінченко Т. В. Тестування програмного забезпечення: методології та інструменти. – Львів: Львівська політехніка, 2020. – 240 с.

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016393895

Дата перевірки:
30.06.2024 05:35:05 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
30.06.2024 05:35:21 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІпз-22-1_Сотников_В_О.pdf

Кількість сторінок: 44 Кількість слів: 6965 Кількість символів: 52128 Розмір файлу: 1.43 MB ID файлу: 1016208711

1.42%
Схожість

Найбільша схожість: 0.76% з джерелом з Бібліотеки (ID файлу: 1016107477)

Пошук збігів з Інтернетом не проводився

1.42% Джерела з Бібліотеки 100 Сторінка 46

0% Цитат

Вилучення цитат вимкнене

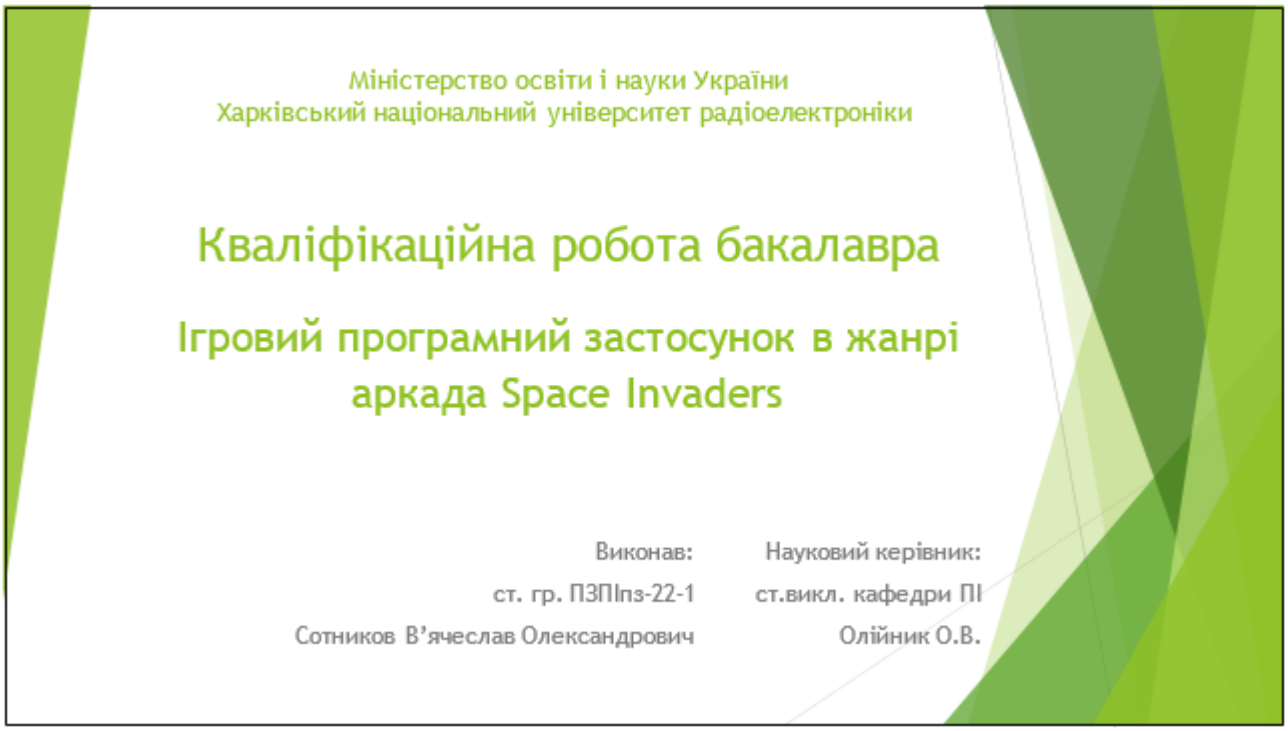
Вилучення списку бібліографічних посилань вимкнене

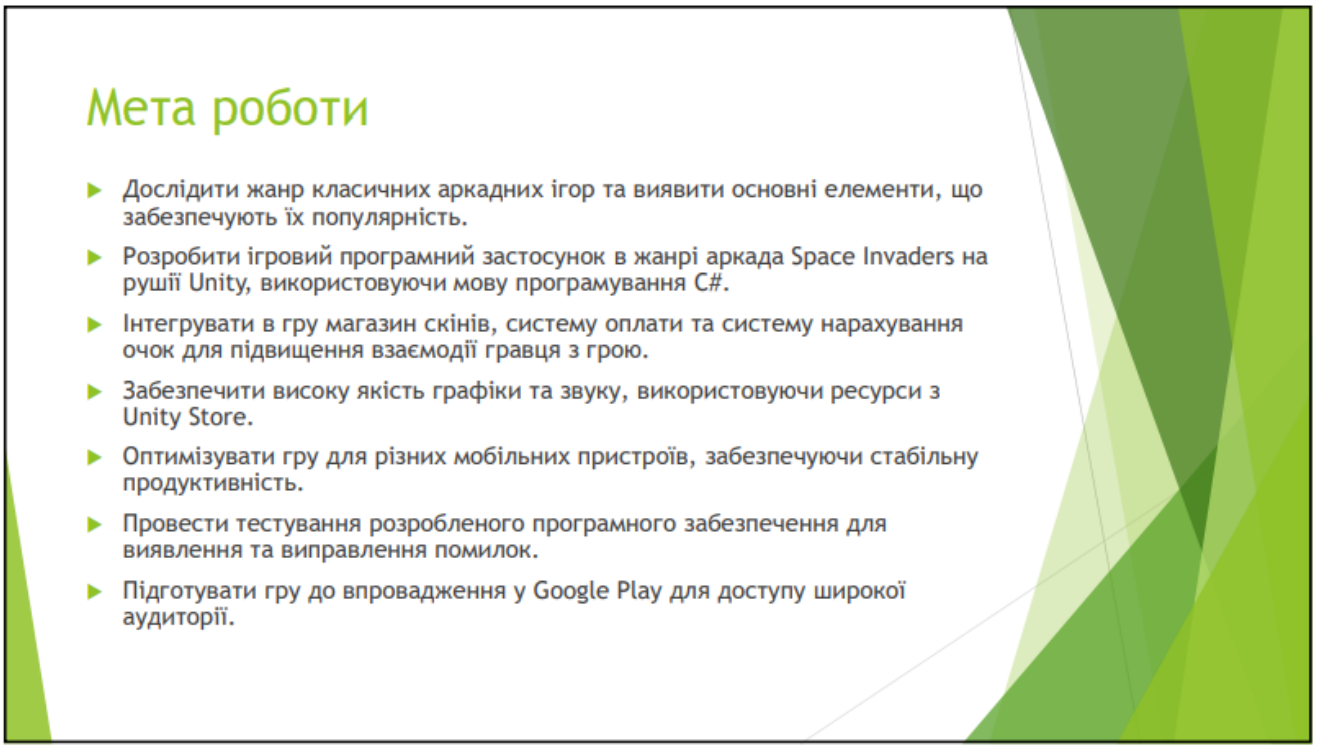
0%
Вилучень

Немає вилучених джерел

ДОДАТОК Б

Слайди презентації





Аналіз існуючих рішень

Гра	Переваги	Недоліки
Space Invaders (оригінал) 	Класика жанру, що визначила стандарти аркадних ігор. Проста механіка гри, яка легко зрозуміла.	Старі графічні та звукові елементи. Відсутність сучасних функцій, таких як внутрішні покупки чи мультиплеєр.
Galaga 	Покращена графіка у порівнянні з оригінальним Space Invaders. Динамічніший геймплей.	Відсутність сучасних елементів монетизації. Може здатися занадто простою для сучасних гравців.
Geometry Wars: Retro Evolved 	Сучасна графіка та візуальні ефекти. Висока динаміка та реіграбельність.	Може бути складною для новачків. Відсутність внутрішніх покупок або кастомізації персонажів.

Вибір інструментів

- Unity - використовується як основний рушій для розробки гри завдяки його потужним можливостям у створенні 2D та 3D ігор, а також великій кількості доступних ресурсів та підтримки.
- C# - мова програмування, яка використовується для написання скриптів у Unity, забезпечуючи управління ігровою логікою та інтеракціями.
- Середовище написання коду Visual Studio.

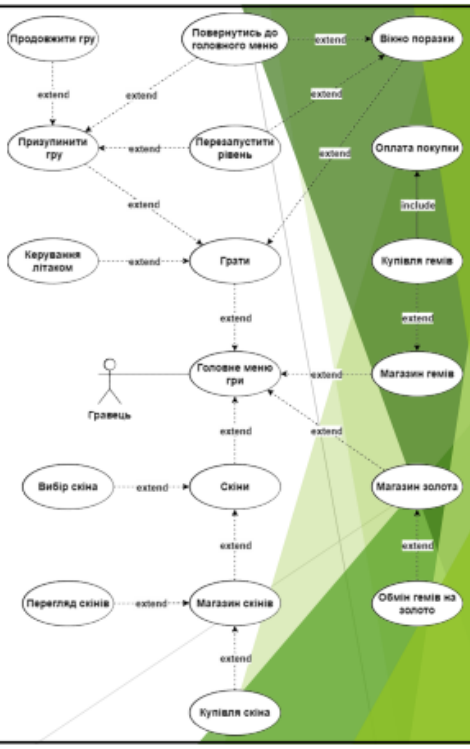
Опис системи

Можливості головного меню:

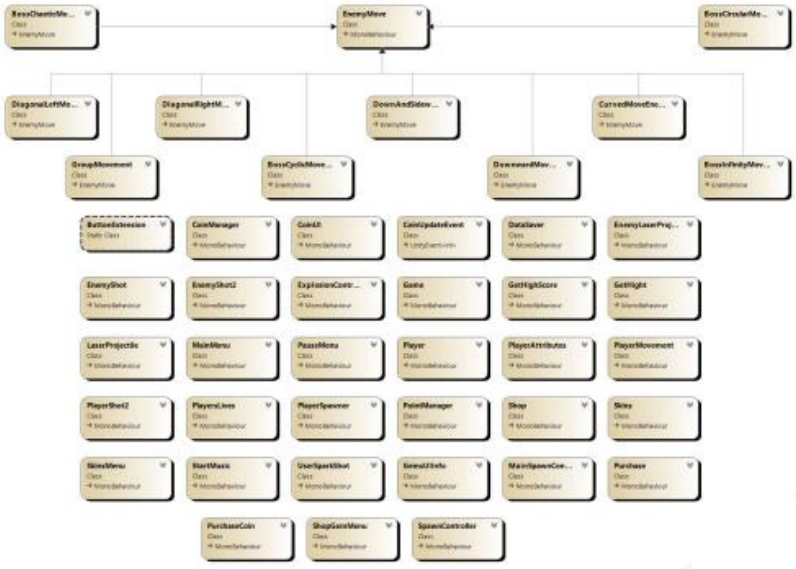
- Вибір скіна
- Перегляд скінів
- Магазин скінів
- Купівля скіна
- Магазин гемів
- Оплата покупки
- Обмін гемів на золото
- Магазин золота

Можливості в грі:

- Керування літаком
- Призупинити гру
- Продовжити гру
- Перезапустити рівень



Структура класів гри



Приклад реалізації хаотичного руху боса

```
public class BossChaoticMovement : EnemyMove
{
    // Початкова відстань спуску
    public float initialDropDistance = 5f;
    // Ширина межі для обмеження руху боса
    public float boundaryWidth = 10f;
    // Висота межі для обмеження руху боса
    public float boundaryHeight = 5f;
    // Інтервал зміни напрямку руху
    public float changeDirectionInterval = 2f;
    private bool initialDropComplete = false;
private Vector2 currentDirection; // Поточний напрямок руху
    protected override void Start()
    {
        base.Start();
        StartCoroutine(InitialDropRoutine()); }
    // Корутина для початкового спуску боса
    private IEnumerator InitialDropRoutine()
    {
        while (transform.position.y > initialDropDistance)
        {
            // Рух вниз до досягнення початкової відстані
            transform.Translate(Vector2.down * speed *
Time.deltaTime);
            yield return null;
        }
        initialDropComplete = true; // Позначаємо, що спуск завершено
        // Починаємо корутину зміни напрямку руху
        StartCoroutine(ChangeDirectionRoutine()); }
```

```
// Перевизначена функція для здійснення руху
    protected override void Move()
    {
        if (initialDropComplete)
        {
            // Здійснення руху в поточному напрямку
            transform.Translate(currentDirection * speed * Time.deltaTime);
            ClampPosition();
        }
    }

    // Функція для обмеження позиції боса
    private void ClampPosition()
    {
        Vector3 pos = transform.position;
        // Обмежуємо координати x та y в межах встановлених значень
        pos.x = Mathf.Clamp(pos.x, -boundaryWidth, boundaryWidth);
        pos.y = Mathf.Clamp(pos.y, initialDropDistance - boundaryHeight,
initialDropDistance + boundaryHeight);
        transform.position = pos; // Встановлюємо нову позицію боса
    }

    // Корутина для зміни напрямку руху
    private IEnumerator ChangeDirectionRoutine()
    {
        while (true)
        {
            // Генеруємо випадковий напрямок і нормалізуємо його
            currentDirection = new Vector2(Random.Range(-1f, 1f),
Random.Range(-1f, 1f)).normalized;
            yield return new WaitForSeconds(changeDirectionInterval); }
    }
}
```

Приклад реалізації контролера спавну

```
IEnumerator SpawnEnemies()
{
    while (true)
    {
        // Отримуємо список ворогів в залежності від рівня гравця
        List<GameObject> currentEnemies = GetEnemiesForLevel();

        // Перевірка, чи список ворогів не порожній
        if (currentEnemies.Count > 0)
        {
            // Вибір випадкового ворога зі списку
            int randomIndex = Random.Range(0, currentEnemies.Count);
            GameObject enemyPrefab = currentEnemies[randomIndex];

            // Спавнимо ворога в межах зони спавна
            Vector3 randomPosition = GetRandomPositionInSpawnZone();
            GameObject enemy = Instantiate(enemyPrefab, randomPosition, Quaternion.identity);

            // Встановлюємо швидкість ворога (якщо вороги мають метод встановлення швидкості)
            // enemy.GetComponent<Enemy>().SetSpeed(enemySpeed);
        }

        // Затримка перед наступним спавном
        yield return new WaitForSeconds(spawnInterval);
    }
}
// Метод для обчислення випадкової позиції в межах зони спавна
Vector3 GetRandomPositionInSpawnZone()
{
    // Отримуємо 2D-колайдер зони спавна
    Collider2D spawnArea = GetComponent<Collider2D>();

    // Отримуємо межі колайдера зони спавна
    Bounds bounds = spawnArea.bounds;

    // Обчислюємо випадкові координати в межах bounds
    float randomX = Random.Range(bounds.min.x, bounds.max.x);
    float randomY = Random.Range(bounds.min.y, bounds.max.y);

    // Повертаємо випадкову позицію в межах зони спавна
    return new Vector3(randomX, randomY, 0); // 0 для z-координати, оскільки ми працюємо в 2D
}
```

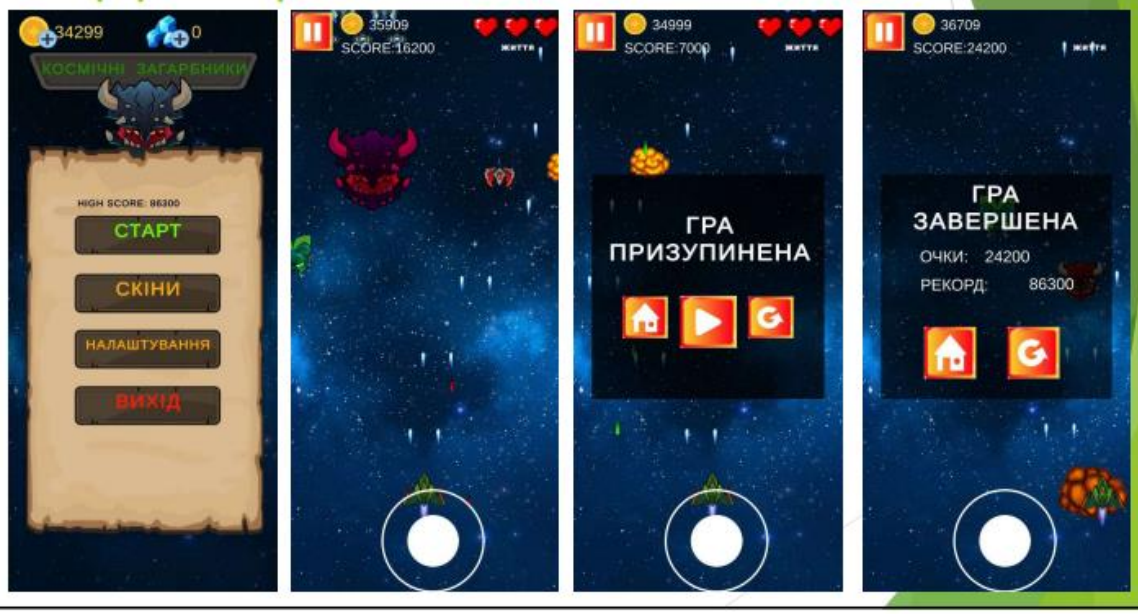
```
// Метод для обрання списку ворогів для відповідного рівня
List<GameObject> GetEnemiesForLevel()
{
    int playerScore = FindObjectOfType<PointManager>().score; // Отримуємо очки гравця з
PointManager

    if (playerScore < level1Threshold)
    {
        return level1Enemies;
    }
    else if (playerScore < level2Threshold)
    {
        return level2Enemies;
    }
    else
    {
        return level3Enemies;
    }
}
// Метод для встановлення швидкості ворогів
public void SetEnemySpeed(float speed)
{
    enemySpeed = speed;
}
// Метод для спавна боса
public void SpawnBoss()
{
    // Перевірка, чи є доступні префаби босів
    if (bossPrefabs.Count > 0)
    {
        // Вибір випадкового префабу боса зі списку
        int randomIndex = Random.Range(0, bossPrefabs.Count);
        GameObject bossPrefab = bossPrefabs[randomIndex];

        // Вибір випадкової позиції в межах зони спавна
        Vector3 randomPosition = GetRandomPositionInSpawnZone();

        // Спавнимо боса
        Instantiate(bossPrefab, randomPosition, Quaternion.identity);
    }
}
```

Інтерфейс гри



Інтерфейс гри

Монетизація

Монетизація гри здійснюється через внутрішньоігрові покупки, а саме через придбання гемів. Гравці можуть купувати геми за реальні гроші, що створює основне джерело доходу для гри. Геми використовуються для придбання золота, яке використовується для подальших покупок скінів.

Висновки

У процесі розробки аркадної гри "Space Invaders" на платформі Unity було проведено детальний аналіз предметної галузі, розроблено концепт-документ, спроектовано архітектуру програмного забезпечення та здійснено успішне тестування ігрового застосунку. Основною метою проекту було створення конкурентоспроможного ігрового продукту з можливістю привертання широкої аудиторії та ефективної монетизації через магазин скінів та систему оплати.

Додаток В

Фінальний код працюючої програми

```
using UnityEngine;
using TMPro;

public class CoinManager : MonoBehaviour
{
    public static CoinManager Instance;

    public int coins;
    public TMP_Text coinsText;

    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject); // Не видаляти при зміні сцени
        }
        else
        {
            Destroy(gameObject);
        }
    }

    void Start()
    {
        UpdateCoinsUI();
    }

    public void AddCoins(int amount)
    {
        coins += amount;
        UpdateCoinsUI();
    }

    void UpdateCoinsUI()
    {
        coinsText.text = "COINS: " + coins.ToString();
    }
}
using TMPro;
using UnityEngine;

public class CoinUI : MonoBehaviour
{
    [SerializeField] private TMP_Text coinsText;

    private void OnEnable()
    {
        Game.Instance.CoinsUpdated += UpdateCoinText; // Підписуємося на
подію оновлення монет
        UpdateCoinText(Game.Instance.Coins); // Оновлюємо монети при
активації об'єкта
    }
```

```
    private void OnDisable()
    {
        Game.Instance.CoinsUpdated -= UpdateCoinText; // Відписуємося від
події при вимкненні об'єкта
    }

    private void UpdateCoinText(int coins)
    {
        if(coinsText!=null)
            coinsText.text = coins.ToString(); // Оновлення тексту монет
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using static UnityEngine.RuleTile.TilingRuleOutput;

public class CurvedMoveEnemy : EnemyMove
{
    public float amplitude = 1f;
    public float frequency = 1f;

    protected override void Move()
    {
        // Рух по дуговій траєкторії
        float x = amplitude * Mathf.Sin(frequency * Time.time);
        float y = -speed * Time.deltaTime; // Рух вниз

        transform.Translate(new Vector2(x, y));
    }
}
using UnityEngine;
using UnityEngine.UI;

public class DataSaver : MonoBehaviour
{
    public string objectName;
    public int id;

    private static DataSaver activeDataSaver; // Посилання на активний
екземпляр DataSaver

    private void Start()
    {
        // Зчитуємо id з PlayerPrefs при ініціалізації
        int savedId = PlayerPrefs.GetInt(objectName, -1);

        // Перевіряємо, чи айді був встановлений, якщо ні, то встановлюємо
його на 0
        if (savedId == -1)
        {
            id = 0;
            PlayerPrefs.SetInt(objectName, id);
        }
        else
        {
            id = savedId;
```

```
        }

        // Перевіряємо, чи цей екземпляр є активним
        if (activeDataSaver == this)
        {
            // Якщо так, оновлюємо текст кнопки
            UpdateButtonText("Selected");
        }
    }

    public void saveInt()
    {
        // Перевіряємо, чи нове значення id відрізняється від збереженого
        if (id != PlayerPrefs.GetInt(objectName, -1))
        {
            // Зберігаємо нове значення id в PlayerPrefs
            PlayerPrefs.SetInt(objectName, id);

            // Позначаємо поточний екземпляр як активний
            activeDataSaver = this;

            // Оновлюємо текст кнопки
            UpdateButtonText("Обрано");
        }
    }

    private void UpdateButtonText(string text)
    {
        // Отримуємо компонент тексту кнопки і змінюємо текст
        Text buttonText = GetComponentInChildren<Text>();
        if (buttonText != null)
        {
            buttonText.text = text;
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DiagonalLeftMovement : EnemyMove
{
    protected override void Move()
    {
        // Рух по діагоналі справа наліво
        if (allowHorizontalMovement && allowVerticalMovement)
        {
            Vector2 movement = (horizontalDirection + -verticalDirection) *
speed * Time.deltaTime;
            transform.Translate(movement);
        }
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        // Якщо зіткнення з об'єктом з тегом Boundary
        if (other.CompareTag("Boundary"))
        {
```

```
            // Змінити напрямок та зміститися вниз
            if (allowHorizontalMovement)
            {
                ChangeHorizontalDirection();
            }
        }
        else if (other.gameObject.tag == "BoundaryDestroy")
        {
            // Знищення при зіткненні з BoundaryDestroy
            Destroy(gameObject);
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class DiagonalRightMovement : EnemyMove
{
    protected override void Move()
    {
        // Рух по діагоналі зліва направо
        if (allowHorizontalMovement && allowVerticalMovement)
        {
            Vector2 movement = (Vector2.right + verticalDirection) * speed
* Time.deltaTime;
            transform.Translate(movement);
        }
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        // Якщо зіткнення з об'єктом з тегом Boundary
        if (other.CompareTag("Boundary"))
        {
            // Змінити напрямок та зміститися вниз
            if (allowHorizontalMovement)
            {
                ChangeHorizontalDirection();
            }
        }
        else if (other.gameObject.tag == "BoundaryDestroy")
        {
            // Знищення при зіткненні з BoundaryDestroy
            Destroy(gameObject);
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DownAndSidewaysMovement : EnemyMove
{
    // Величина для зміни напрямку руху
    public float boundaryWidth = 10f;

    // Логіка для руху
    protected override void Move()
```

```
    {
        // Рух по осі Y (вниз)
        if (allowVerticalMovement)
        {
            transform.Translate(verticalDirection * speed *
Time.deltaTime);
        }

        // Рух по осі X в межах 10
        if (allowHorizontalMovement)
        {
            float horizontalMovement = Mathf.PingPong(Time.time * speed,
boundaryWidth * 2) - boundaryWidth;
            transform.Translate(horizontalDirection * horizontalMovement *
Time.deltaTime);
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using static UnityEngine.RuleTile.TilingRuleOutput;

public class DownwardMovement : EnemyMove
{
    public bool moveRight = true;

    protected override void Move()
    {
        // Рух по осі X в межах 10
        float movementX = moveRight ? speed * Time.deltaTime : -speed *
Time.deltaTime;
        transform.Translate(new Vector2(movementX, -speed *
Time.deltaTime));

        // Зміна напрямку при досягненні межі
        if (transform.position.x >= 10 || transform.position.x <= -10)
        {
            moveRight = !moveRight;
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag == "Boundary")
        {
            speed *= -1;
        }
        else if (collision.gameObject.tag == "BoundaryDestroy")
        {
            // Знищення при зіткненні з BoundaryDestroy
            Destroy(gameObject);
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class EnemyLaserProjectile : MonoBehaviour
{
    public float speed = 1f; // Швидкість польоту

    private void Start()
    {

    }
    void Update()
    {
        transform.Translate(Vector2.right * speed * Time.deltaTime);
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        // Перевірка зіткнення з об'єктом Boundary
        if (other.gameObject.tag == "Boundary")
        {
            // Знищення лазерного знаряду
            Destroy(gameObject);
        }
    }

}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyMove : MonoBehaviour
{
    public float speed = 1f;
    public int health = 1; // Кількість життя ворога

    // Змінні для визначення напрямку руху
    public Vector2 horizontalDirection = Vector2.left; // Напрямок руху по
горизонталі
    public Vector2 verticalDirection = -Vector2.down; // Напрямок руху по
вертикалі

    // Змінні для вимкнення руху в певному напрямку
    public bool allowHorizontalMovement = true;
    public bool allowVerticalMovement = true;

    // Подія для обробки зіткнень
    public event System.Action<Collider2D> OnTriggerEnter2DEvent;

    // Параметри для анімації вибуху
    public GameObject explosionPrefab; // Префаб вибуху
    public float explosionScale = 0.5f; // Масштаб вибуху

    // Змінна для кількості очок, які гравець отримує за вбивство ворога
    public int scoreValue = 50;

    // Змінна для кількості монет, які гравець отримує за вбивство ворога
    public int coinValue = 10;

    private PointManager pointManager;
```

```
    protected virtual void Start()
    {
        try
        {
            // Знаходимо PointManager у сцені
            pointManager = FindObjectOfType<PointManager>();
        }
        catch (System.Exception ex)
        {
            Debug.LogError("Error initializing PointManager: " +
ex.Message);
        }
    }

    protected virtual void Update()
    {
        // Рух ворога
        Move();
    }

    protected virtual void Move()
    {
        // Рух по заданому напрямку тільки, якщо дозволено
        Vector2 movement = new Vector2(
            allowHorizontalMovement ? horizontalDirection.x : 0f,
            allowVerticalMovement ? verticalDirection.y : 0f
        );

        transform.Translate(movement * speed * Time.deltaTime);
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        try
        {
            // Виклик події для подальшої обробки в похідних класах
            OnTriggerEnter2DEvent?.Invoke(other);

            // Обробка зіткнення з гравцем або його пострілом
            if (other.gameObject.CompareTag("PlayerProjectile"))
            {
                // Зменшуємо життя ворога
                health--;

                // Перевірка, чи інший об'єкт не є null, перед його
знищенням
                if (other.gameObject != null)
                {
                    Destroy(other.gameObject);
                }

                // Якщо життя ворога досягає нуля або нижче, знищуємо його
                if (health <= 0)
                {
                    // Створюємо анімацію вибуху
                    GameObject explosion = Instantiate(explosionPrefab,
transform.position, Quaternion.identity);
```

```
                    explosion.transform.localScale = new
Vector3(explosionScale, explosionScale, 1f);

                    // Видаляємо вибух після завершення анімації,
перевіряючи, чи вибух не є null
                    if (explosion != null)
                    {
                        Destroy(explosion,
explosion.GetComponent<Animator>().GetCurrentAnimatorStateInfo(0).length);
                    }

                    // Додаємо очки гравцеві, перевіряючи, чи pointManager
не є null
                    pointManager?.UpdateScore(scoreValue);

                    // Додаємо монети гравцеві через CoinManager,
перевіряючи, чи Instance не є null
                    if (CoinManager.Instance != null)
                    {
                        CoinManager.Instance.AddCoins(coinValue);
                    }

                    // Знищуємо ворога
                    if (gameObject != null)
                    {
                        Destroy(gameObject);
                    }
                }
            }

            // Якщо зіткнулися з межами екрану, змінюємо напрямок руху
            if (other.gameObject.CompareTag("Boundary"))
            {
                ChangeHorizontalDirection();
            }
            else if (other.gameObject.CompareTag("BoundaryDestroy"))
            {
                // Перевірка, чи інший об'єкт не є null перед його
знищенням
                if (gameObject != null)
                {
                    Destroy(gameObject);
                }
            }
        }
        catch (System.Exception ex)
        {
            Debug.LogError("Error in OnTriggerEnter2D: " + ex.Message);
        }
    }

    public void TakeDamage(int damage)
    {
        try
        {
            // Зменшуємо здоров'я ворога
            health -= damage;
```

```
            // Якщо здоров'я ворога досягає нуля або нижче, знищуємо його
            if (health <= 0)
            {
                // Створюємо анімацію вибуху
                GameObject explosion = Instantiate(explosionPrefab,
transform.position, Quaternion.identity);
                explosion.transform.localScale = new
Vector3(explosionScale, explosionScale, 1f);

                // Видаляємо вибух після завершення анімації, перевіряючи,
чи вибух не є null
                if (explosion != null)
                {
                    Destroy(explosion,
explosion.GetComponent<Animator>().GetCurrentAnimatorStateInfo(0).length);
                }

                // Додаємо очки гравцеві, перевіряючи, чи pointManager не є
null
                pointManager?.UpdateScore(scoreValue);

                // Додаємо монети гравцеві через CoinManager, перевіряючи,
чи Instance не є null
                if (CoinManager.Instance != null)
                {
                    CoinManager.Instance.AddCoins(coinValue);
                }

                // Перевірка, чи об'єкт не є null перед його знищенням
                if (gameObject != null)
                {
                    Destroy(gameObject);
                }
            }
        }
        catch (System.Exception ex)
        {
            Debug.LogError("Error in TakeDamage: " + ex.Message);
        }
    }

    // Метод для зміни напрямку руху по горизонталі
    protected virtual void ChangeHorizontalDirection()
    {
        // Змінюємо напрямок по горизонталі на протилежний
        horizontalDirection *= -1;
    }

    // Метод для зміни напрямку руху по вертикалі
    protected virtual void ChangeVerticalDirection()
    {
        // Змінюємо напрямок по вертикалі на протилежний
        verticalDirection *= -1;
    }

    // Метод для вимкнення руху по горизонталі
    public void DisableHorizontalMovement()
    {
```

```
        allowHorizontalMovement = false;
    }

    // Метод для вимкнення руху по вертикалі
    public void DisableVerticalMovement()
    {
        allowVerticalMovement = false;
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyShot : MonoBehaviour
{
    public float spawnTimer = 3f; // Інтервал між пострілами
    public GameObject enemyProjectilePrefab; // Префаб LaserProjectile
    public float spawnMax = 10; // Швидкість знаряду
    public float spawnMin = 4; // Швидкість знаряду

    // Start is called before the first frame update
    void Start()
    {
        spawnTimer = Random.Range(spawnMin, spawnMax);
    }

    private void Update()
    {
        spawnTimer -=Time.deltaTime;
        if (spawnTimer <= 0)
        {
            Instantiate(enemyProjectilePrefab, transform.position,
Quaternion.Euler(0f, 0f, -90f));
            spawnTimer = Random.Range(spawnMin, spawnMax);
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ExplosionController : MonoBehaviour
{
    // Публічний метод для видалення об'єкта вибуху
    public void DestroyExplosion()
    {
        Destroy(gameObject);
    }
}
using System;
using UnityEngine;

public class Game : MonoBehaviour
{
    public static Game Instance { get; private set; }
    public event Action<int> CoinsUpdated; // Подія оновлення монет

    private int coins;
```

```
    public int Coins
    {
        get { return coins; }
        set
        {
            coins = value;
            CoinsUpdated?.Invoke(coins); // Викликаємо подію після
оновлення монет
            SaveCoins(); // Збереження монет
        }
    }
    public bool HasEnoughCoins(int amount)
    {
        return (Coins >= amount);
    }
    public void AddCoins(int amount)
    {
        Coins += amount;
    }
    public void UseCoins(int amount)
    {
        if (HasEnoughCoins(amount))
        {
            Coins -= amount;
            SaveCoins();
        }
        else
        {
            Debug.LogWarning("Not enough coins!");
        }
    }
    private const string COINS_KEY = "Coins";

    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }
    }

    private void Start()
    {
        LoadCoins();
    }

    private void SaveCoins()
    {
        PlayerPrefs.SetInt(COINS_KEY, Coins);
        PlayerPrefs.Save();
    }

    private void LoadCoins()
```

```
    {
        Coins = PlayerPrefs.GetInt(COINS_KEY, 0);
    }
}
using TMPro;
using UnityEngine;

public class GetHighScore : MonoBehaviour
{
    [SerializeField] public TMP_Text highScoreText;

    private void Start()
    {
        if (highScoreText == null)
        {
            Debug.LogError("TextMeshProUGUI component is not assigned!");
            return;
        }

        // Перевіряємо, чи існує значення для збереження найвищого рахунку
        if (PlayerPrefs.HasKey("SavedHightScore"))
        {
            // Отримуємо значення найвищого рахунку з PlayerPrefs
            int highScore = PlayerPrefs.GetInt("SavedHightScore");

            // Ініціалізуємо текст об'єкта з цим скриптом значенням
найвищого рахунку
            highScoreText.text = highScore.ToString();
        }
        else
        {
            Debug.LogWarning("SavedHightScore not found in PlayerPrefs!");
        }
    }
}
using UnityEngine;
using UnityEngine.UI;

public class GetHight : MonoBehaviour
{
    private void Start()
    {
        // Перевіряємо, чи існує значення для збереження найвищого рахунку
        if (PlayerPrefs.HasKey("SavedHightScore"))
        {
            // Отримуємо значення найвищого рахунку з префаба
            int highScore = PlayerPrefs.GetInt("SavedHightScore");

            // Ініціалізуємо текст об'єкта з цим скриптом значенням
найвищого рахунку
            GetComponent<Text>().text = highScore.ToString();
        }
        else
        {
            Debug.LogWarning("SavedHightScore not found in PlayerPrefs!");
        }
    }
}
```

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

public class LaserProjectile : MonoBehaviour
{
    public float speed = 1f; // Швидкість польоту
    public Vector2 direction = new Vector2(0, 1); // Напрямок руху

    private void Start()
    {

    }

    void Update()
    {
        // Рух лазерного знаряду
        Move();
    }

    void Move()
    {
        // Рух враховуючи напрямок та швидкість
        transform.Translate(direction * speed * Time.deltaTime);
    }
    void OnTriggerEnter2D(Collider2D other)
    {
        // Перевірка зіткнення з об'єктом ворога
        if (other.gameObject.CompareTag("Enemy"))
        {
            // Отримуємо компонент ворога
            EnemyMove enemy = other.gameObject.GetComponent<EnemyMove>();

            // Якщо у ворога є компонент EnemyMove
            if (enemy != null)
            {
                // Зменшуємо життя ворога
                enemy.TakeDamage(1);

                // Видалення лазерного снаряду
                Destroy(gameObject);
            }
        }

        // Перевірка зіткнення з об'єктом Boundary
        if (other.gameObject.CompareTag("Boundary"))
        {
            // Видалення лазерного снаряду
            Destroy(gameObject);
        }
    }

}
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
```

```
    public void StartGame()
    {
        SceneManager.LoadScene("Game");
        Time.timeScale = 1.0f;
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    private bool isPaused;
    public GameObject pausePanel;
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetButtonDown("Cancel"))
        {
            if (isPaused)
            {
                ResumeGame();
            }
            else { PauseGame();}
        }
    }

    public void PauseGame()
    {
        Time.timeScale = 0f;
        isPaused = true;
        pausePanel.SetActive(true);
    }

    public void ResumeGame()
    {
        Time.timeScale = 1.0f;
        isPaused = false;
        pausePanel.SetActive(false);
    }

    public void RestartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name); //
Перезавантаження поточного рівня
        Time.timeScale = 1.0f;
    }

    public void GoHome()
    {
        SceneManager.LoadScene("Menu");
    }
```

```
}

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

public class Player : MonoBehaviour
{
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.collider.gameObject.tag == "Enemy")
        {
            Destroy(collision.collider.gameObject);
            Instantiate(PlayersLives.Instance.explosionPrefab,
transform.position, Quaternion.identity);
            PlayersLives.Instance.TakeDamage(1);
        }
    }

    private void OnTriggerEnter(Collider collision)
    {
        if (collision.gameObject.tag == "EnemyShot")
        {
            Destroy(collision.gameObject);
            Instantiate(PlayersLives.Instance.explosionPrefab,
transform.position, Quaternion.identity);
            PlayersLives.Instance.TakeDamage(1);
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    public float speed = 5f;
    public Joystick joystick;

    void Update()
    {
        // Отримати вхід від джойстика
        float horizontalInput = joystick.Horizontal;

        // Рухатися вправо або вліво в залежності від вхідних даних з
джойстика
        Vector2 movement = new Vector2(horizontalInput, 0f);

        // Застосувати рух до позиції гравця
        transform.Translate(movement * speed * Time.deltaTime);
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class PlayersLives : MonoBehaviour
```

```
{
    public static PlayersLives Instance; // Сінглтон

    public int lives = 3;
    public Image[] livesUI;
    public GameObject explosionPrefab;
    public GameObject explosionPlayerPrefab;
    public GameObject gameOverPanel;
    public PointManager pointManager;

    private void Awake()
    {
        if (Instance == null)
            Instance = this;
        else
            Destroy(gameObject);
    }

    public void TakeDamage(int damage)
    {
        lives -= damage;

        // Оновити відображення життів
        UpdateLivesUI();

        if (lives <= 0)
        {
            GameOver();
        }
    }

    void UpdateLivesUI()
    {
        for (int i = 0; i < livesUI.Length; i++)
        {
            if (i < lives)
                livesUI[i].enabled = true;
            else
                livesUI[i].enabled = false;
        }
    }

    void GameOver()
    {
        gameOverPanel.SetActive(true);
        Time.timeScale = 0f;
        pointManager.HighScoreUpdate();
        Instantiate(explosionPlayerPrefab, transform.position,
Quaternion.identity);
        Destroy(gameObject);
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class PlayerSpawner : MonoBehaviour
```

```
{
    public GameObject[] playerPrefabs; // Масив префабів гравців
    private const string SELECTED_SKIN_KEY = "SelectedSkinIndex";

    // Посилання на джойстик
    public Joystick joystick;

    public Image[] livesUI;
    public GameObject explosionPrefab;
    public GameObject explosionPlayerPrefab;
    public GameObject gameOverPanel;
    public PointManager pointManager;

    private void Start()
    {
        // Отримуємо індекс обраного скіна
        int selectedSkinIndex = PlayerPrefs.GetInt(SELECTED_SKIN_KEY, 0);

        // Перевіряємо індекс і вибираємо потрібний префаб гравця для
спавну
        if (selectedSkinIndex >= 0 && selectedSkinIndex <
playerPrefabs.Length)
        {
            // Спавнимо гравця з обраним префабом
            GameObject playerInstance =
Instantiate(playerPrefabs[selectedSkinIndex], transform.position,
Quaternion.identity);

            // Передаємо джойстик до гравця
            PlayerMovement playerMovement =
playerInstance.GetComponent<PlayerMovement>();
            if (playerMovement != null)
            {
                playerMovement.joystick = joystick;
            }
            else
            {
                Debug.LogWarning("Гравець не має компонента PlayerMovement
або Joystick");
            }
        }
        else
        {
            Debug.LogWarning("Індекс обраного скіна недійсний: " +
selectedSkinIndex);
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class PointManager : MonoBehaviour
{
    public int score;
    public TMP_Text scoreText;
    public TMP_Text finaleScoreText;
```

```
    public TMP_Text highScoreText;
    void Start()
    {

    }
    public void UpdateScore(int points)
    {
        score += points;
        scoreText.text = "SCORE:" + score;
        Game.Instance.AddCoins(5);
    }

    public void HighScoreUpdate()
    {
        if (PlayerPrefs.HasKey("SavedHightScore"))
        {
            if(score > PlayerPrefs.GetInt("SavedHightScore")) {
                PlayerPrefs.SetInt("SavedHightScore", score);
            }

        }
        else
        {
            PlayerPrefs.SetInt("SavedHightScore", score);
        }

        finaleScoreText.text = score.ToString();
        highScoreText.text =
PlayerPrefs.GetInt("SavedHightScore").ToString();
    }

}
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class Shop : MonoBehaviour
{

    #region Singlton:Shop

    public static Shop Instance;
    private void Awake()
    {
        if (Instance == null)
            Instance = this;
        else
            Destroy(gameObject);
    }
    #endregion
    [System.Serializable] public class ShopItem
    {
        public Sprite Image;
        public int Price;
        public bool IsPurchased = false;
    }
```

```
    public List<ShopItem> ShopItemsList;

    [SerializeField] GameObject ItemTemplate;
    GameObject g;

    [SerializeField] Transform ShopScrollView;
    [SerializeField] Animator NoCoinAnim;
    [SerializeField] GameObject ShopPanel;

    Button buyButton;
    void Start()
    {
        int len = ShopItemsList.Count;
        for (int i = 0; i < len; i++)
        {
            // Отримання статусу купленості з PlayerPrefs при старті гри
            int isPurchased = PlayerPrefs.GetInt("Skin_" + i, 0); // 0 -
означає, що скін не куплений
            ShopItemsList[i].IsPurchased = isPurchased == 1 ? true : false;

            g = Instantiate(ItemTemplate, ShopScrollView);
            g.transform.GetChild(0).GetComponent<Image>().sprite =
ShopItemsList[i].Image;

g.transform.GetChild(1).GetChild(0).GetComponent<TMP_Text>().text =
ShopItemsList[i].Price.ToString();
            buyButton = g.transform.GetChild(2).GetComponent<Button>();
            if (ShopItemsList[i].IsPurchased)
            {
                DisableBuyButton();
            }

            buyButton.AddEventListener(i, OnShopItemBtnclicked);
        }
    }

    void OnShopItemBtnclicked(int itemIndex)
    {
        if (Game.Instance.HasEnoughCoins(ShopItemsList[itemIndex].Price))
        {
            Game.Instance.UseCoins(ShopItemsList[itemIndex].Price);
            ShopItemsList[itemIndex].IsPurchased = true;
            buyButton =
ShopScrollView.GetChild(itemIndex).GetChild(2).GetComponent<Button>();
            DisableBuyButton();

            Skins.Instance.AddSkin(ShopItemsList[itemIndex].Image);

            // Оновлення та збереження статусу купленості в PlayerPrefs
            PlayerPrefs.SetInt("Skin_" + itemIndex, 1); // 1 - означає, що
скін куплений
            PlayerPrefs.Save(); // Зберегти зміни
        }
        else
        {
            NoCoinAnim.SetTrigger("NoCoins");
            Debug.Log("No Gold");
```

```
        }
    }

    void DisableBuyButton()
    {
        buyButton.interactable = false;
        buyButton.transform.GetChild(0).GetComponent<TMP_Text>().text =
"Purchased";
    }

    public void OpenShop()
    {
        ShopPanel.SetActive(true);
    }

    public void CloseShop()
    {
        ShopPanel.SetActive(false);
    }

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

public class Skins : MonoBehaviour
{
    #region Singlton:Skins

    public static Skins Instance;
    private void Awake()
    {
        if (Instance == null)
            Instance = this;
        else
            Destroy(gameObject);
    }
    #endregion

    public class Skin
    {
        public Sprite Image;
    }

    public List<Skin> SkinsList;

    GameObject g;
    int newSelectedIndex, prevSelectedIndex;
    [SerializeField] GameObject SkinUITemplate;
    [SerializeField] Transform SkinsScrollView;
    [SerializeField] Color ActiveSkinColor;
    [SerializeField] Color DefaultSkinColor;

    [SerializeField] Image CurrentSkin;

    private const string SELECTED_SKIN_KEY = "SelectedSkinIndex";
```

```
    private void Start()
    {
        GetAvailableSkins();
        // Зчитуємо обраний скін при старті гри
        int selectedSkinIndex = PlayerPrefs.GetInt(SELECTED_SKIN_KEY, 0);
        SelectedSkin(selectedSkinIndex);
    }

    void GetAvailableSkins()
    {
        for (int i = 0; i < Shop.Instance.ShopItemsList.Count; i++)
        {
            if (Shop.Instance.ShopItemsList[i].IsPurchased)
            {
                AddSkin(Shop.Instance.ShopItemsList[i].Image);
            }
        }
    }

    public void AddSkin(Sprite image)
    {
        if (SkinsList == null)
        {
            SkinsList = new List<Skin>();
        }
        Skin skin = new Skin()
        {
            Image = image
        };

        SkinsList.Add(skin);
        g = Instantiate(SkinUITemplate, SkinsScrollView);
        g.transform.GetChild(0).GetComponent<Image>().sprite = skin.Image;

        g.transform.GetComponent<Button>().AddEventListener(SkinsList.Count
- 1, OnSkinClick);
    }

    void OnSkinClick(int SkinIndex)
    {
        SelectedSkin(SkinIndex);
        // Зберігаємо обраний скін у PlayerPrefs при зміні
        PlayerPrefs.SetInt(SELECTED_SKIN_KEY, SkinIndex);
        PlayerPrefs.Save();
    }

    void SelectedSkin(int SkinIndex)
    {
        prevSelectedIndex = newSelectedIndex;
        newSelectedIndex = SkinIndex;

SkinsScrollView.GetChild(prevSelectedIndex).GetComponent<Image>().color =
DefaultSkinColor;

SkinsScrollView.GetChild(newSelectedIndex).GetComponent<Image>().color =
ActiveSkinColor;
```

```
        CurrentSkin.sprite = SkinsList[SkinIndex].Image;
    }
}

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

public class SkinsMenu : MonoBehaviour
{
    [SerializeField] GameObject SkinPanel;
    public void OpenSkinsProfile()
    {
        SkinPanel.SetActive(true);
    }

    public void CloseSkinsProfile()
    {
        SkinPanel.SetActive(false);
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class StartMusic : MonoBehaviour
{
    public AudioClip musicClip; // Поле для збереження аудіо кліпу
    private AudioSource musicSource; // Поле для збереження аудіо джерела

    // Метод Start викликається перед першим оновленням кадру
    void Start()
    {
        musicSource = gameObject.AddComponent<AudioSource>(); // Додаємо
аудіо джерело до поточного об'єкта
        if (musicClip != null) // Перевіряємо, чи переданий аудіо кліп
        {
            musicSource.clip = musicClip; // Призначаємо аудіо кліп аудіо
джерелу
            musicSource.loop = true; // Встановлюємо параметр loop в true,
щоб музика відтворювалася циклічно
            musicSource.Play(); // Запускаємо музику
        }
        else
        {
            Debug.LogWarning("No music clip assigned!"); // Виводимо
попередження, якщо аудіо кліп не переданий
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

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

```
public class UserSparkShot : MonoBehaviour
{
    public float spawnInterval = 3f; // Інтервал між пострілами
    public GameObject laserProjectilePrefab; // Префаб LaserProjectile
    public float projectileSpeed = 10f; // Швидкість знаряду
    public Vector2 projectileDirection = new Vector2(0, 1); // Напрямок
знаряду (за замовчуванням - вгору)

    // Start is called before the first frame update
    void Start()
    {
        // Розпочинаємо інвок повторюваного виклику методу SpawnProjectile
        InvokeRepeating("SpawnProjectile", 0f, spawnInterval);
    }

    // Метод для створення LaserProjectile
    void SpawnProjectile()
    {
        // Створюємо новий знаряд за допомогою префабу
        GameObject laserProjectile = Instantiate(laserProjectilePrefab,
transform.position, Quaternion.identity);

        // Додаємо швидкість та напрямок знаряду
        LaserProjectile laserProjectileScript =
laserProjectile.GetComponent<LaserProjectile>();
        laserProjectileScript.speed = projectileSpeed; // Встановлюємо
швидкість
        laserProjectileScript.direction = projectileDirection; //
Встановлюємо напрямок

        // Повертаємо снаряд на 90 градусів по осі Z
        laserProjectile.transform.Rotate(new Vector3(0, 0, 90));
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BossChaoticMovement : EnemyMove
{
    // Початкова відстань спуску
    public float initialDropDistance = 5f;

    // Ширина межі для обмеження руху боса
    public float boundaryWidth = 10f;

    // Висота межі для обмеження руху боса
    public float boundaryHeight = 5f;

    // Інтервал зміни напрямку руху
    public float changeDirectionInterval = 2f;

    private bool initialDropComplete = false; // Прапорець, що позначає
завершення початкового спуску
    private Vector2 currentDirection; // Поточний напрямок руху

    protected override void Start()
    {
```

```
        base.Start();
        StartCoroutine(InitialDropRoutine()); // Початок корутини для
початкового спуску
    }

    // Корутина для початкового спуску боса
    private IEnumerator InitialDropRoutine()
    {
        while (transform.position.y > initialDropDistance)
        {
            // Рух вниз до досягнення початкової відстані
            transform.Translate(Vector2.down * speed * Time.deltaTime);
            yield return null;
        }
        initialDropComplete = true; // Позначаємо, що спуск завершено
        StartCoroutine(ChangeDirectionRoutine()); // Починаємо корутину
зміни напрямку руху
    }

    // Перевизначена функція для здійснення руху
    protected override void Move()
    {
        if (initialDropComplete)
        {
            // Здійснення руху в поточному напрямку
            transform.Translate(currentDirection * speed * Time.deltaTime);
            ClampPosition(); // Обмеження позиції боса в межах встановлених
меж
        }
    }

    // Функція для обмеження позиції боса
    private void ClampPosition()
    {
        Vector3 pos = transform.position;
        // Обмежуємо координати x та y в межах встановлених значень
        pos.x = Mathf.Clamp(pos.x, -boundaryWidth, boundaryWidth);
        pos.y = Mathf.Clamp(pos.y, initialDropDistance - boundaryHeight,
initialDropDistance + boundaryHeight);
        transform.position = pos; // Встановлюємо нову позицію боса
    }

    // Корутина для зміни напрямку руху
    private IEnumerator ChangeDirectionRoutine()
    {
        while (true)
        {
            // Генеруємо випадковий напрямок і нормалізуємо його
            currentDirection = new Vector2(Random.Range(-1f, 1f),
Random.Range(-1f, 1f)).normalized;
            yield return new WaitForSeconds(changeDirectionInterval); //
Чекаємо заданий інтервал
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class BossCircularMovement : EnemyMove
{
    public float initialDropDistance = 5f;
    public float radius = 5f;
    public float speedRotation = 1f;
    private bool initialDropComplete = false;
    private float angle = 0f;

    protected override void Start()
    {
        base.Start();
        StartCoroutine(InitialDropRoutine());
    }

    private IEnumerator InitialDropRoutine()
    {
        // Спуск боса вниз на задану відстань
        while (transform.position.y > initialDropDistance)
        {
            transform.Translate(Vector2.down * speed * Time.deltaTime);
            yield return null;
        }
        initialDropComplete = true;
    }

    protected override void Move()
    {
        if (initialDropComplete)
        {
            angle += speedRotation * Time.deltaTime;
            float x = Mathf.Cos(angle) * radius;
            float y = Mathf.Sin(angle) * radius;
            transform.position = new Vector2(x, initialDropDistance + y);
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BossCyclicMovement : EnemyMove
{
    public float initialDropDistance = 5f;
    public float boundaryWidth = 10f;
    public float boundaryHeight = 5f;
    private bool initialDropComplete = false;
    private Vector2[] waypoints;
    private int currentWaypointIndex = 0;
    public float waypointReachThreshold = 0.1f;

    protected override void Start()
    {
        base.Start();
        StartCoroutine(InitialDropRoutine());
    }

    private IEnumerator InitialDropRoutine()
```

```
    {
        // Спуск боса вниз на задану відстань
        while (transform.position.y > initialDropDistance)
        {
            transform.Translate(Vector2.down * speed * Time.deltaTime);
            yield return null;
        }
        initialDropComplete = true;
        CreateWaypoints();
    }

    protected override void Move()
    {
        if (initialDropComplete)
        {
            Vector2 targetPosition = waypoints[currentWaypointIndex];
            Vector2 direction = (targetPosition -
(Vector2)transform.position).normalized;
            transform.Translate(direction * speed * Time.deltaTime);

            if (Vector2.Distance(transform.position, targetPosition) <
waypointReachThreshold)
            {
                currentWaypointIndex = (currentWaypointIndex + 1) %
waypoints.Length;
            }
        }
    }

    private void CreateWaypoints()
    {
        waypoints = new Vector2;
        waypoints[0] = new Vector2(-boundaryWidth, initialDropDistance); //
Ліво-верх
        waypoints[1] = new Vector2(boundaryWidth, initialDropDistance);  //
Право-верх
        waypoints[2] = new Vector2(boundaryWidth, initialDropDistance -
boundaryHeight); // Право-низ
        waypoints[3] = new Vector2(-boundaryWidth, initialDropDistance -
boundaryHeight); // Ліво-низ
    }
}
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class GemsUIInfo : MonoBehaviour
{
    public static GemsUIInfo instance;

    [SerializeField]
    public TMP_Text gemsText;
    void Start()
    {
        instance = this;
        UpdateGemsText();
```

```
    }

    public void UpdateGemsText() {
        int playerGems = PlayerPrefs.GetInt("gems");
        gemsText.text = playerGems.ToString();
    }
}
using System.Collections;
using UnityEngine;

public class MainSpawnController : MonoBehaviour
{
    // Порожні об'єкти для кожної спавн-зони
    public GameObject topSpawnZone;
    public GameObject leftSpawnZone;
    public GameObject rightSpawnZone;
    public GameObject pointManagerGameObject; // посилання на GameObject з
PointManager

    // Посилання на спавн-контролери
    private SpawnController topSpawnController;
    private SpawnController leftSpawnController;
    private SpawnController rightSpawnController;
    private PointManager pointManager; // змінна для зберігання
PointManager

    // Змінна для збереження очок гравця
    public int playerScore;

    // Час між спавнами ворогів у кожній зоні
    public float spawnInterval = 2.0f;

    // Початкова швидкість ворогів
    public float initialEnemySpeed = 1.0f;

    // Збільшення складності після кожного циклу
    public float difficultyIncrease = 1.1f;

    // Очки для спавна боса
    public int bossSpawnScoreThreshold = 5000;

    // Лічильник рівнів (кількість спавнень босів)
    private int lvl = 0;

    void Start()
    {
        // Отримуємо спавн-контролери з порожніх об'єктів
        topSpawnController = topSpawnZone.GetComponent<SpawnController>();
        leftSpawnController =
leftSpawnZone.GetComponent<SpawnController>();
        rightSpawnController =
rightSpawnZone.GetComponent<SpawnController>();
        pointManager = pointManagerGameObject.GetComponent<PointManager>();

        // Перевірка, чи PointManager знайдено
        if (pointManager == null)
        {
```

```
            Debug.LogError("Не вдалося знайти PointManager в переданому
GameObject.");
            return;
        }

        // Перевірка наявності спавн-контролерів
        if (topSpawnController == null || leftSpawnController == null ||
rightSpawnController == null)
        {
            Debug.LogError("Не вдалося знайти один або більше спавн-
контролерів.");
            return;
        }

        // Починаємо цикл контролю спавна
        StartCoroutine(ManageSpawn());
    }

    IEnumerator ManageSpawn()
    {
        while (true)
        {
            // Отримуємо очки гравця з PointManager
            playerScore = pointManager.score;

            // Обчислюємо рівень на основі очок гравця та порогу очок для
спавна боса
            int level = playerScore / bossSpawnScoreThreshold;

            // Перевіряємо, чи рівень більше, ніж лічильник рівнів (`lvl`)
            if (level > lvl)
            {
                // Спавнимо боса у випадковій зоні спавна
                SpawnBossRandomly();

                // Оновлюємо лічильник рівнів (`lvl`)
                lvl = level;

                // Збільшуємо складність гри
                IncreaseDifficulty();
            }

            // Затримка перед наступним спавном
            yield return new WaitForSeconds(spawnInterval);
        }
    }

    // Функція для спавна босів у випадковій зоні спавна
    void SpawnBossRandomly()
    {
        // Випадково вибираємо одну з зон спавна
        int randomIndex = Random.Range(0, 3);
        switch (randomIndex)
        {
            case 0:
                topSpawnController.SpawnBoss();
                break;
            case 1:
```

```
                leftSpawnController.SpawnBoss();
                break;
            case 2:
                rightSpawnController.SpawnBoss();
                break;
        }
    }

    // Функція для збільшення складності гри
    void IncreaseDifficulty()
    {
        initialEnemySpeed *= difficultyIncrease;

        // Збільшуємо швидкість ворогів у кожному спавн-контролері
        topSpawnController.SetEnemySpeed(initialEnemySpeed);
        leftSpawnController.SetEnemySpeed(initialEnemySpeed);
        rightSpawnController.SetEnemySpeed(initialEnemySpeed);
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Purchasing;
public class Purchase : MonoBehaviour
{

    public void OnPurchaseCompleted(Product product)
    {
        switch (product.definition.id)
        {
            case "space.inviders.50gems":
                AddGems(50);
                break;
            case "space.inviders.100gems":
                AddGems(100);
                break;
            case "space.inviders.150gems":
                AddGems(150);
                break;
            case "space.inviders.200gems":
                AddGems(200);
                break;
            case "space.inviders.500gems":
                AddGems(500);
                break;
            case "space.inviders.1000gems":
                AddGems(1000);
                break;
            case "space.inviders.5000gems":
                AddGems(5000);
                break;
            case "space.inviders.9999gems":
                AddGems(9999);
                break;
        }

    }
    private void AddGems(int gems)
```

```
    {
        int playerGems = PlayerPrefs.GetInt("gems");
        playerGems += gems;
        PlayerPrefs.SetInt("gems", playerGems);
        PlayerPrefs.Save();
        Debug.Log("Add gems " + gems);
        GemsUIInfo.instance.UpdateGemsText();
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PurchaseCoin : MonoBehaviour
{
    public void OnClick500()
    {
        int playerGems = PlayerPrefs.GetInt("gems");
        if (playerGems >= 50)
        {
            playerGems -= 50;
            PlayerPrefs.SetInt("gems", playerGems);
            PlayerPrefs.Save();

            GemsUIInfo.instance.UpdateGemsText();
            Game.Instance.AddCoins(500);
            Debug.Log("Button clicked! Coin 500 purchased.");
        }
    }

    public void OnClick2000()
    {
        int playerGems = PlayerPrefs.GetInt("gems");
        if (playerGems >= 150)
        {
            playerGems -= 150;
            PlayerPrefs.SetInt("gems", playerGems);
            PlayerPrefs.Save();

            GemsUIInfo.instance.UpdateGemsText();
            Game.Instance.AddCoins(2000);
            Debug.Log("Button clicked! Coin 2000 purchased.");
        }
    }
    public void OnClick5000()
    {
        int playerGems = PlayerPrefs.GetInt("gems");
        if (playerGems >= 350)
        {
            playerGems -= 350;
            PlayerPrefs.SetInt("gems", playerGems);
            PlayerPrefs.Save();

            GemsUIInfo.instance.UpdateGemsText();
            Game.Instance.AddCoins(5000);
            Debug.Log("Button clicked! Coin 5000 purchased.");
        }
    }
```

```
    public void OnClick10000()
    {
        int playerGems = PlayerPrefs.GetInt("gems");
        if (playerGems >= 600)
        {
            playerGems -= 600;
            PlayerPrefs.SetInt("gems", playerGems);
            PlayerPrefs.Save();

            GemsUIInfo.instance.UpdateGemsText();
            Game.Instance.AddCoins(10000);
            Debug.Log("Button clicked! Coin 10000 purchased.");
        }
    }
    public void OnClick50000()
    {
        int playerGems = PlayerPrefs.GetInt("gems");
        if (playerGems >= 1400)
        {
            playerGems -= 1400;
            PlayerPrefs.SetInt("gems", playerGems);
            PlayerPrefs.Save();

            GemsUIInfo.instance.UpdateGemsText();
            Game.Instance.AddCoins(50000);
            Debug.Log("Button clicked! Coin 10000 purchased.");
        }
    }
    public void OnClick100000()
    {
        int playerGems = PlayerPrefs.GetInt("gems");
        if (playerGems >= 2599)
        {
            playerGems -= 2599;
            PlayerPrefs.SetInt("gems", playerGems);
            PlayerPrefs.Save();

            GemsUIInfo.instance.UpdateGemsText();
            Game.Instance.AddCoins(100000);
            Debug.Log("Button clicked! Coin 10000 purchased.");
        }
    }
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ShopGemMenu : MonoBehaviour
{
    [SerializeField] public GameObject ShopPanel;
    public void OpenShop()
    {
        ShopPanel.SetActive(true);
    }

    public void CloseShop()
    {
        ShopPanel.SetActive(false);
```

```
    }

}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpawnController : MonoBehaviour
{
    // Списки ворогів для різних рівнів
    public List<GameObject> level1Enemies;
    public List<GameObject> level2Enemies;
    public List<GameObject> level3Enemies;

    // Список префабів босів для спавна
    public List<GameObject> bossPrefabs;

    // Інтервал між спавнами ворогів
    public float spawnInterval = 2.0f;

    // Початкова швидкість ворогів
    private float enemySpeed = 1.0f;

    // Рівні очок для переходу між рівнями
    public int level1Threshold = 5000;
    public int level2Threshold = 10000;

    void Start()
    {
        // Починаємо цикл спавна ворогів
        StartCoroutine(SpawnEnemies());
    }

    IEnumerator SpawnEnemies()
    {
        while (true)
        {
            // Отримуємо список ворогів в залежності від рівня гравця
            List<GameObject> currentEnemies = GetEnemiesForLevel();

            // Перевірка, чи список ворогів не порожній
            if (currentEnemies.Count > 0)
            {
                // Вибір випадкового ворога зі списку
                int randomIndex = Random.Range(0, currentEnemies.Count);
                GameObject enemyPrefab = currentEnemies[randomIndex];

                // Спавнимо ворога в межах зони спавна
                Vector3 randomPosition = GetRandomPositionInSpawnZone();
                GameObject enemy = Instantiate(enemyPrefab, randomPosition,
Quaternion.identity);

                // Встановлюємо швидкість ворога (якщо вороги мають метод
встановлення швидкості)
                // enemy.GetComponent<Enemy>().SetSpeed(enemySpeed);
            }

            // Затримка перед наступним спавном
```

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

    // Метод для обчислення випадкової позиції в межах зони спавна
    Vector3 GetRandomPositionInSpawnZone()
    {
        // Отримуємо 2D-колайдер зони спавна
        Collider2D spawnArea = GetComponent<Collider2D>();

        // Отримуємо межі колайдера зони спавна
        Bounds bounds = spawnArea.bounds;

        // Обчислюємо випадкові координати в межах bounds
        float randomX = Random.Range(bounds.min.x, bounds.max.x);
        float randomY = Random.Range(bounds.min.y, bounds.max.y);

        // Повертаємо випадкову позицію в межах зони спавна
        return new Vector3(randomX, randomY, 0); // 0 для z-координати,
оскільки ми працюємо в 2D
    }

    // Метод для обрання списку ворогів для відповідного рівня
    List<GameObject> GetEnemiesForLevel()
    {
        int playerScore = FindObjectOfType<PointManager>().score; //
Отримуємо очки гравця з PointManager

        if (playerScore < level1Threshold)
        {
            return level1Enemies;
        }
        else if (playerScore < level2Threshold)
        {
            return level2Enemies;
        }
        else
        {
            return level3Enemies;
        }
    }

    // Метод для встановлення швидкості ворогів
    public void SetEnemySpeed(float speed)
    {
        enemySpeed = speed;
    }

    // Метод для спавна босів
    public void SpawnBoss()
    {
        // Перевірка, чи є доступні префаби босів
        if (bossPrefabs.Count > 0)
        {
            // Вибір випадкового префабу босів зі списку
            int randomIndex = Random.Range(0, bossPrefabs.Count);
            GameObject bossPrefab = bossPrefabs[randomIndex];
```

```
            // Вибір випадкової позиції в межах зони спавна
            Vector3 randomPosition = GetRandomPositionInSpawnZone();

            // Спавнимо боса
            Instantiate(bossPrefab, randomPosition, Quaternion.identity);
        }
    }
}
```