МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ
ВІДДІЛЕННЯ «ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП'ЮТЕРНІ НАУКИ»
ЦИКЛОВА КОМІСІЯ «ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП'ЮТЕРНІ НАУКИ»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи освітнього ступеня «фаховий молодший бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма 122 «Комп'ютерні науки»)
на тему:

«Розробка ігрового застосунку "Танкові бої" на основі рушія Unity: проєктування, оптимізація та реалізація основних ігрових механік»

Студента IV курсу П-41 групи
Василевського Олександра Романовича
(прізвище, ім'я та по-батькові)

Керівник Можаровський Сергій Володимирович
(прізвище, ім'я та по-батькові)

Рецензент
(прізвище, ім'я та по-батькові)

Національна шкала

Кількість балів: Оцінка: ECTS

Члени комісії Ісаєв А.М.
(підпис) (прізвище та ініціали)

Габрійчук Н.І.
(підпис) (прізвище та ініціали)

Устименко Л.М.
(підпис) (прізвище та ініціали)

Устименко Я.І.
(підпис) (прізвище та ініціали)

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ
ВІДДІЛЕННЯ «ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП'ЮТЕРНІ НАУКИ»
ЦИКЛОВА КОМІСІЯ «ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП'ЮТЕРНІ НАУКИ»

«ЗАТВЕРДЖУЮ»
Голова циклової комісії «Інженерна інфраструктура та комп'ютерні науки»
_____ Д. М. Палій
«06» листопада 2024 р.

ЗАВДАННЯ
на дипломну роботу

Здобувач фахової передвищої освіти: **Василевський Олександр Романович**

Керівник роботи: **Можаровський Сергій Володимирович**

Тема роботи: **«Розробка ігрового застосунку «Танкові бої» на основі рушія Unity: проєктування, оптимізація та реалізація основних ігрових механік»**, затверджена наказом закладу вищої освіти від «3» грудня 2024 р., №496н.

Вихідні дані для роботи: об’єктом дослідження є ігрові застосунки на рушії Unity в жанрі аркадної симуляції боїв важкої військової техніки, предметом дослідження: випробовування можливостей рушія Unity та використання об’єктно-орієнтованого програмування (ООП) у розробці аркадних шутерів від третьої особи.

Консультанти з дипломної роботи із зазначенням розділів, що їх стосується:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Можаровський С. В.	05.12.2024	15.04.2025
2	Можаровський С. В.	15.04.2025	20.05.2025
3	Можаровський С. В.	20.05.2025	09.06.2025

Календарний план

№ з/п	Етап роботи	Термін виконання	Примітка
1	Визначення мети роботи та цілей, встановлення об’єкта та предмета дослідження; з'ясування завдань розробки	05.12.2024	Виконано
2	Окреслення проекту розробки	21.01.2025	Виконано
3	Встатовлення ТЗ та вивчення джерел	11.02.2025	Виконано
4	Планування структури і форми ДР	25.03.2025	Виконано
5	Створення коду, його оптимізація та відлагождення	15.04.2025	Виконано
6	Тестування програми	07.05.2025	Виконано
7	Підготовка та оформлення ПЗ	20.05.2025	Виконано
8	Попередній захист роботи	09.06.2025	Виконано

Здобувач фахової передвищої освіти _____ Олександр ВАСИЛЕВСЬКИЙ

Керівник _____ Сергій МОЖАРОВСЬКИЙ

РЕФЕРАТ

Записка: 76 стор., 19 рис., 1 табл., 1 додаток, 20 джерел.

КОМП’ЮТЕРНА ГРА, РЕЖИМ РЕАЛЬНОГО ЧАСУ, ВИГЛЯД ЗВЕРХУ, ОДИН ГРАВЕЦЬ, ГРАВЕЦЬ ПРОТИ КОМП’ЮТЕРА.

Об’єкт досліждення – програми ігрових додатків у жанрі аркада з ігроладом у режимі реального часу.

Мета роботи – дослідження, проєктування та тестування гри-аркади з ігроладом у режимі реального часу з використанням можливостей рушія Unity.

Методи дослідження: аналіз доступної літератури та працюючих популярних додатків у жанрі аркад з видом зверху; проєктування програмних рішень у рушії Unity; тестування готового додатку; розгляд результатів тестування з метою оцінки функціоналу та ефективності програми.

Результати: розроблено гру-аркаду з геймплеєм у режимі реального часу; розглянуто та порівняно розроблений додаток з існуючими ігроладними рішеннями; зроблено висновки щодо швидкодії, ефективності та варіантів використання розробленого додатку.

ABSTRACT

The work consists of 3 chapters; list of abbreviations, list of cited sourses, conclusions and the sourse code. The paper consist of 76 pages of printed text, contains 19 figures, 1 table and 20 cited sourses.

The purpose of the work is to research, design, and test an arcade game with a real-time gameplay using the capabilities of the Unity engine.

KEY WORDS: COMPUTER GAME, REAL-TIME MODE, TOP-DOWN VIEW, SINGLE PLAYER, PLAYER VS. COMPUTER.

ЗМІСТ

ВСТУП

Актуальність теми.

Ігрові дотатки вже давно не є просто розвагою для дітей. Окрім того, що вони дозволяють відволіктись від рутинниз задач, вони стали засобом донесення своїх думок для різних авторів, підняння важливих тем та навіть тренування майбутніх фахівців для різних галузей виробництва в безпечному середовищі. Програми ігрового типу також активно застосовуються у військовій сфері для тренування механіків, водіїв, операторів, відпрацювання злагоджених дій загону та багато іншого. Тобто сучасний ігробуд є багатомільярдним бізнесом і може слугувати як для розваг так і для серйозних навчальних та тренувальних цілей у різних сферах діяльності людини.

Особливої увагу заслуговують аркадні ігри, тобто сильно спрощені на відміну від симуляції, які будуть привітними для новачків та гравців без досвіду на попередньої підготовки. Цей тип та жанр ігрових додатків до цього часу є одним з найпопулярнішим через очевидні причини, зокрема через низькі вимоги до гравця. В більшості випадків, аркадні ігри проєктуються з видом від 3-ї особи (який дає гравцю перевагу); супротивником є бот, тобто ігролад, що називається PvE (PvE — Player versus Environment). За інформацією з відкритих джерел, до 90% гравців мають у своїй ігротеці Steam хоча би одну гру-аркаду, це тренд є стабільним, чи навіть має тенденцію до зростання, вже принаймні 10 років. Нові та «старі» гравці купують все більше ігор даного жанру, а розробники та видавці цих додатків заробляють величезні статки.

Військові ігри також користуються непідробною популярністю і займають далеко не останні рейтинги в різноманітних чартах.

Мета: програмування гри-аркади симуляції танкових боїв.

Об’єктом дослідження: є ігрові застосунки на рушії Unity в жанрі аркадної симуляції боїв важкої військової техніки.

Предмет дослідження: випробовування можливостей рушія Unity у розробці аркадних шутерів від третьої особи.

Методи дослідження:

- пошук та аналіз існуючих аркадних ігор;
- розробка та кодування гри в середовищі Unity; тестування та валідація готового додатку;
- обробка результатів тестування з метою оцінки функціональності та ефективності програми.

Структура дипломної роботи. Робота складаєть зі вступу; аналізу електронних джерел, щодо заданої теми дипломної роботи; проектування додатку; його тестування; розгляду роботи програми, характерних властивостей встановлення та користування.

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

1.1. Постановка задачі розробки програмного продукту

Визначною ціллю дипломної роботи є створення аркадного ігрового додатку для симуляцію танкових боїв в режимі реального часу з виглядом від третьої особи. Для досягнення цілі були визначені кроки:

1. Розляд існуючих рішень в “глобальній павутині” та за її межами;
2. Підготовка на обробка графічного інтерфейсу гри;
3. Розробка структури, логіки та геймплею додатку;
4. Окреслення основних класів програми;
5. Підготовка визначеного користувацького інтерфейсу гри.

Програма являє собою ігровий додаток, для гравців зацікавлених в військовиї аркадних симуляторах, які надають можливість протиставити свої сили запрограмованим противникам, обробляючи команди гравця з пристрою введення та визначаючи долю гравця та командного пункту, який треба захищати.

Цільова аудиторя додатку – геймери, які полюбляють аркадні комп’ютерні ігри-симулятори, з PvE ігроладом, що випробовують адаптаційні навички гравця.

Створення додатку планується виконати на базі ігрового рушія (середовища для проєктування) під назвою «Unity» використовуючи Visual Studio 2022 року, що керується об’єктно-орієнтованою мовою програмування C#, а також додаток Windows Forms від Microsoft. Підключення до мережі інтернет – опціанальне, додаток функціонуватиме автономно. За планом програма вимагатиме ~ 150 МБ RAM (Random Access Memory), щодо інших ресурсів персонального комп’ютера (ПК), гра не стане викликом ні для процесора, ні для відеокарти. Остаточний білд гри перевірятиметься віруси та інше шкідливе ПЗ використовуючи антивірусний софт, що знаходиться у вільному доступі.

Додаток повинен працювати стабільно, тобто, без критичних помилок так «крашів», має бути відкритий до модернізаціх і змін, для чого Unity найкращий вибір.

1.2. Аналіз та порівняння подібних додатків

Жанр аркадних ігор з дією в реальному часі та вигляді зверху є напевно одним з найпорулярніших серед усіх геймерів, враховуючи різний вік так уподобання такої широкої групи людей. Більше того аркади були вже досить популярні на ігрових автоматах, ще до широкого поширення ПК у світі. Аркадні ігри, особливо виконані в “real time” (реальному часі) кофігурації з “top-down perspective” (вид зверху вниз) стали одним з перших проявів гейм-культури. У 70-80-ті роки XX століття, аркадні автомати стали справжнісіньким соц. культурним феноменом [1]. Даний жанр і в наші дні зберігає свою популярність, розвиваючи основі ідеї, але не відходячи від свого ядра – динамічного ґеймплею, низького “порогу входження” та інтенсивності перебігу гри. Сучасні представники, такі як “Enter the Gungeon”, “Nuclear Throne” або надпопуоярна “Hades”, показують як прадавні механіка розвиваються і комбінуються з іншими жанрами та стилями, додаючи елементи rogue-like, RPG та ін. Дослідники, зокрема [2,3] вважають, що це не лише прояв ностальгії, але також і глибокою психологією людини: короткі ігрові сесії, швидка динаміка перебігу ігрових подій та аудіовізуальні нагороди гравця, стимулюють мозок та надають почуття миттевого задоволення. Тобто, аркади в “real time” конфігурації – це не тільки історія, а й активний жанр, що продовжує розвиватись та зберігати свою актуальність завдяки гнучкості, різноманітній естетиці та легкому на веселому ігроладу [4,16].

Проведений диференційований пошук дозволив виявити низку аналогічних продуктів, які подібні до досліджуваного застосунку за змістовною ідеєю, принципами навігації, ігровою логікою та візуальним оформленням, зокрема: «Tiny Tanks», «TANKNAROK», «Super Tank Battle – MobileArmy», «Steel Armor: Blaze of War», «Anarchy Arcade: Tank Zone (мод до Source Engine)» та купа інших для різноманітних операційних систем ПК, смартфонів,

портанивних на стаціонарних консолей.

Чудовий прикладом гри даного жанру є «Tiny Tanks» [5], головне меню якої показано на рисунку 1.1.

Рисунок 1.1 – Загальний вигляд головного меню «Tiny Tanks»

Якщо вибрано опцію «Local» запускається PvE режим, ігровий процес в цьому разі виглядає як зображено на рисуноку 1.2.

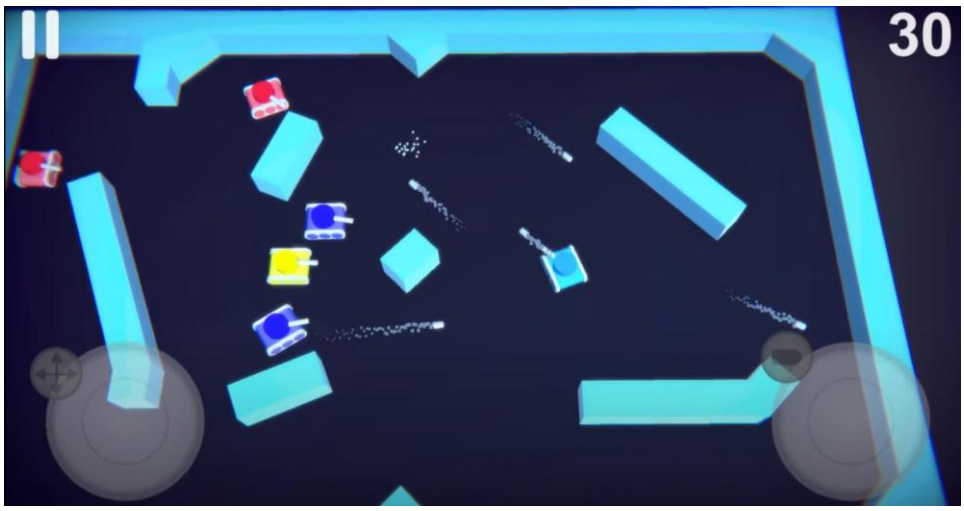

Рисунок 1.2 – Приклад ігрового моменту гри «Tiny Tanks»

Меню гри максимально мінімалістисне навіть для аркадного жанру, однак показує кількість очків, траєкторію снарядів гравця та противників, також наявна

індикація натиснутих клавіш чи перемикачів (в даному випадку гравець грає на геймпаді). Також гра надає можливість грати проти інших гравців в режимі PvP.

Вцілому додаток «Tiny Tanks» є легкою аркадою усіх типів гравців, кого цікавлять подібного типу ігри. Програма доволі примітивна, одна надійна, оскільки немає складних структур. Інтерфейс простий і зрозумілий після навіть без додаткових пояснень. Програма займає близько 1 ГБ простору диску, що за сучасними характеристиками є малим проєктом. Рекомендовані вимоги до ПК: ОС – Windows 7 і новіше; процесор – від Intel i5; RAM – 2 ГВ; відеокарта – Geforce 600 series чи подібна. Гра потребує постійного широкосвугового підключення до Інтернету. Гра не є безкоштовною і вартує близько 10 EUR (без урахування регіональних цін).

Наступним прикладом танкової аркади є «Super Tank Battle» [6], це гра на для мобільних платформ, яку можна завантажити з Apple «AppStore» чи «Google Play» для ОС IOS та Android відповідно. Ця гра більш схожа на старі ігри на консолях NES, та її неліцензованої копії відомою в Україні як «Dendy». Головне меню гри презентовано на рисунку 1.3.

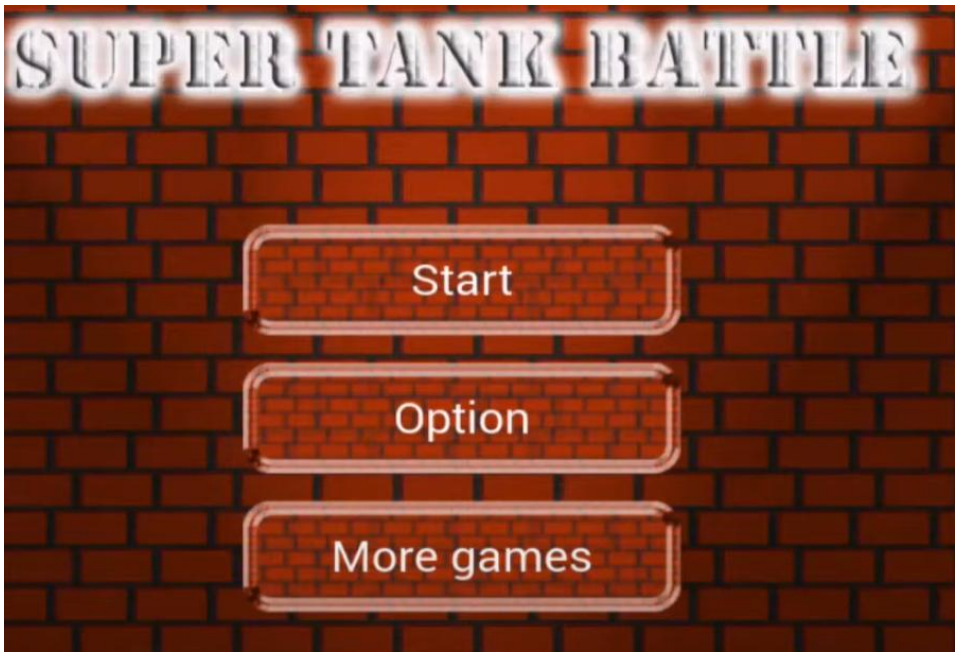

Рисунок 1.3 – Головне меню «Super Tank Battle»

Візуальний стиль гри зазанав незначних змін, покращилась графіка, додано понад 500 нових рівнів. Головне меню має 3 кнопки, серед яких «Start», що веде

до початку гри; «Option» – меню налаштувань та «More games» – більше проєктів від розробника.

При натисканні «Start» гра переходить до проміжного меню, що показано на рисунку 1.4., де можна розпочати нову гру – «New Game», використати зароблені під час гри спеціальні монети для покращень в ігровому магазині – «Use Coins», чи повернутись до головного меню написнувши кнопку «Return».

Рисунок 1.4 – Проміжне меню «Start»

У проміжному меню «Start» при виборі «New Game», відкривається підменю з вибором рівня та складності гри, яке виглядає як показано на рисунку 1.5. Замок на рівні (або за «Stage» за термінологією гри) означає, що він ще не доступний, його буде розблоковано у разі успішного заверешення попереднього рівня. Після проходження певної кількості рівнів, складність автоматично збільшується, покращується «штучний інтелект» (ШІ) противників, збільшується їх кількість, шкода та швидкість появи нової хвилі.

Рисунок 1.5 – Підменю вибору рівня та складності

Геймплей, знімок якого показано на рисунку 1.6, не є дуже складним, враховуючи що це все ж таки аркадна гра. Фактично гравець контролює танк 5-ма кнопками, 4 з них слугують для направлення руху техніки, а 5-та для пострілів. Моделі танків виглядають футуристично, постріли стиліховані під лазерні промені. На мапі є різноманітні об’єкти, серед них штаб (зірка оточена цегляною стіною); коричневі цегляні блоки, які можна знищити пострілом, та світліші блоки, які неможна знищити звичайним пострілом, на вищих рівняє з’являються також і інші типи об’єктів так як, наприклад хащі, в них можуть проїхати танки противників та гравця, знищити їх неможна. Також далі може бути текстура льоду пересуваючись по якому танк гравця має підвищену інерцію руху і ковзає. На екрані також присутня довідка, де можна прочитати про керування, підсилення, умови поразки та перемоги. Також надано інформацію про кількість життів гравця, та кількість противників на рівні, що залишилось для знищення. Крім того, також підсвічені можливі підсилення гравця, серед них може бути виклик підкріплення з кількох союзних танків.

Рисунок 1.6 – Скріншот геймплею гри

Умовою поразки є втрата останнього життя (що і сталось на рисунку 1.7.) чи знищення штабу гравця, перемоги – анігіляцію усіх ворогів на рівні.

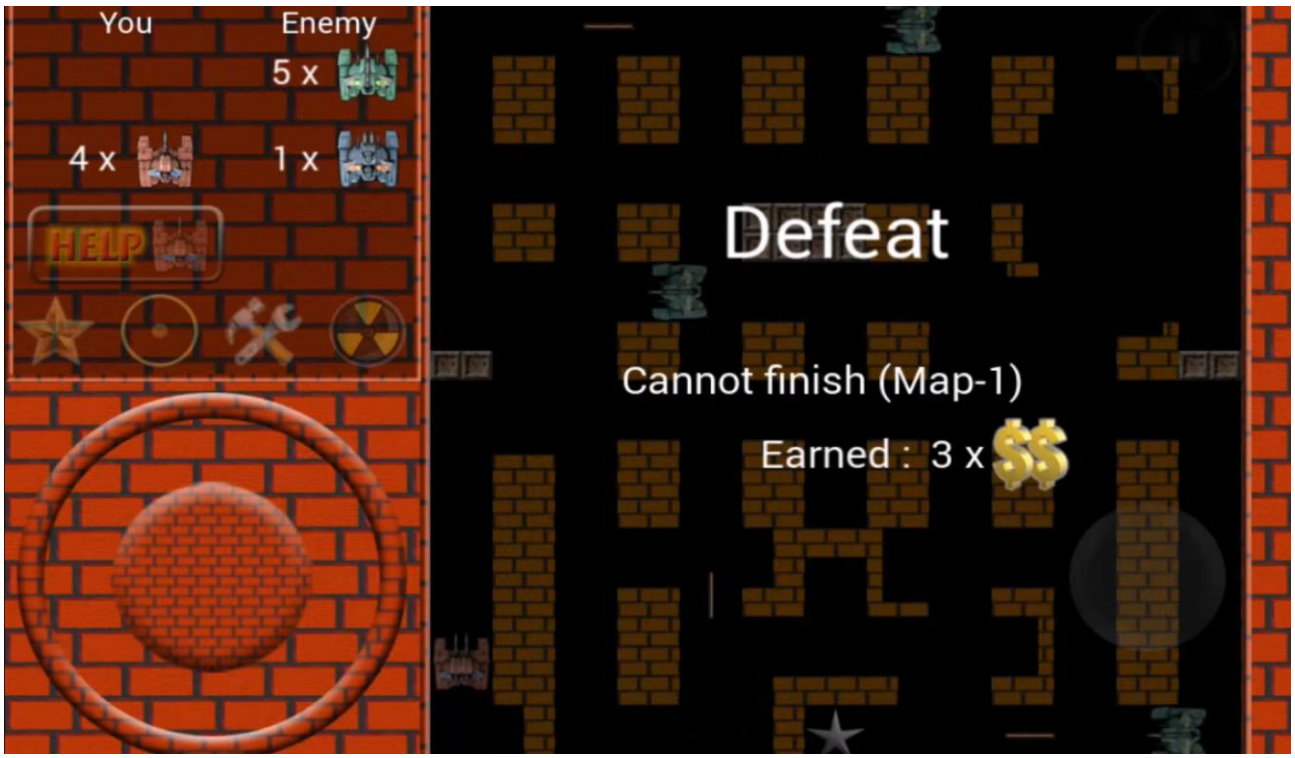

Рисунок 1.7 – Скріншот екрану поразки

У будь-якому випадку, перемога чи поразка, гравець отримує ігрову

валюту ($$), яку можна витратити на покращення, прибання нових моделей танка та інших візуальних змін та ефектиних підсилень.

Під час тестування гри на емуляторі Android, гра працювала стабільно, без помилок, багів, просідання FPS та інших технічних проблем. Інтерфейс гри простий, зрозумілий та інформативний, програма має ряд простих налаштувань, яких цілком достатньо для кастомізації комфортної гри. Оскільки гра розроблена на Android і її код по суті є відкритим, читерський софт, як от «Cheat Engine» працює безвідмовно, однак в однокористувацькій грі, це не є критичною проблемою. Кілька антивірусів не побачили в грі «Super Tank Battle» жодних вірусів чи потенційних загроз.

«Super Tank Battle» вартує ~ 10 EUR (без урахування регіональних цін), що є немалою сумою, при цьому добре працює навіть на телефонах з відносто старими чіпами.

1.3. Вибір та обґрунтування технологій розробки промного продукту

Враховуючи особливості вибраного жанру ігор та вже розглянуті подібні проєкти, можна приорітизувати графічне оформлення гри (при цьому не сильно виходити за рамки прийнятого для жанру мінімальзму), звичайно ж технічно гра має бути чудово працюючою та оптимізованою для вибраного девайсу і ОС, повністю українізованою і при можливості мати захист від недобросовісного використання.

Доцільно буде зробити програму безкоштовною та доступною до розповсюдження за ліцензією «CC» («креатів комонс»), що дозволяє будь-яке використання продукту, при умові, що буде вказано оригінально автора.

Крім іншого приорітетною задачею буде забезпечити швидкою, так як це критично важливо для будь-якої гри в реальному часі. Зконцетруватись варто на скелеті подібних ігор-аркад, а вже 2-м кроком додати додаткові і унікальні «фішки» до версії для дипломної роботи.

Для реалізації буде мудрим рішенням обрати гнучку і розгалужену систему, де можна реалізувати широкий спектр ідей. Саме тому об’єктно-орієнтовну мова програмування C# [7,8,12] на ігровому рушії Unity [9,10] з

допомогою Windows Forms [11] є доречним рішенням. Це допоможе краще модерувати ресурси ПК необхідні для комфортної гри, важливо, щоб вимоги не перевищували ті, які характерні для жанрових додатків, оскільки гравці зі «слабкими» ПК нерідко обирають цей жанр.

Розглянуті ігри-аркади є вінтажними і майже не підтримуються і не оновлюються розробниками, що планується врахувати і поправити при розробці програми дипломної роботи враховуючи матеріали [13-15].

Висновки до розділу 1.

Було проведено порівняння 2-х програмних продуктів у жанрі аркада та примітивна симуляція, на основі яких було продумано структуру та клювочі функції майбутної програми. Крім того було сформовано мету роботи та встановлено перелік завдань, реалізація яких є необхідною та приорітетною для досягнення функціональної та концептуальної цілісності, не порушуючи жанровиз канонів. Особлива увага надана механіці гри та логіці взаємодії гравця з інтерфейсом та ймовірному розширенню функціоналу ігроладу та особливих «фішок». Вцілому можна розглядати додаток, як такий, що має потенціал до масштабування та подальшого вдосконалення.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональні можливості та використання програмого продукту

Ігрову програму проєктуємо зважаючи на загальні вимоги до подібних додатків вцілому та аркадних симуляторів зокрема, враховуючи ігролад в реальному часі [4].

Оскільки єдиним актором, що ініціює всі сценарії використання програмного продукту є гравець – застосунок має забезпечити повноцінний і комфортний досвід взаємодії незалежно від того, чи мав користувач досвід із проєктами цього жанру [15,17,18]. Приймаючи до уваги обмежений обсяг функціональних можливостей програми, раціонально зосередити увагу діаграми варіантів використання саме на аспектах взаємодії між користувачем та додатком, а також на структурі ігроладу [19,20].

Для додатку передбачено наступні сценарії використання:

1. Запустити гру (початок роботи з програмою, ініціалізація користувацького інтерфейсу);
2. Вибрати режим гри;
3. Почати нову гру (ініціація нового сеансу гри, завантаження першого рівня);
4. Керувати танком (переміщення, стрільба, ухилення від снарядів противників);
5. Збирати бонуси (підбірання предметів: додаткові життя, броня, апгрейди);
6. Захищати штаб (утримання центрального об’єкта від атак противника);
7. Знищувати ворогів;
8. Програти або перемогти (завершення рівня перемогою (всі вороги знищені) або поразкою (втрачена база чи всі життя));
9. Побачити результат рівня (відображення очок, статистики);

10. Вихід з гри (завершення сесії гри або вихід у головне меню).

Цей процес узагальнено на рисунку 2.1, де акцент зроблено на ключових сценаріях використання, що відповідають геймплейній логіці застосунку.

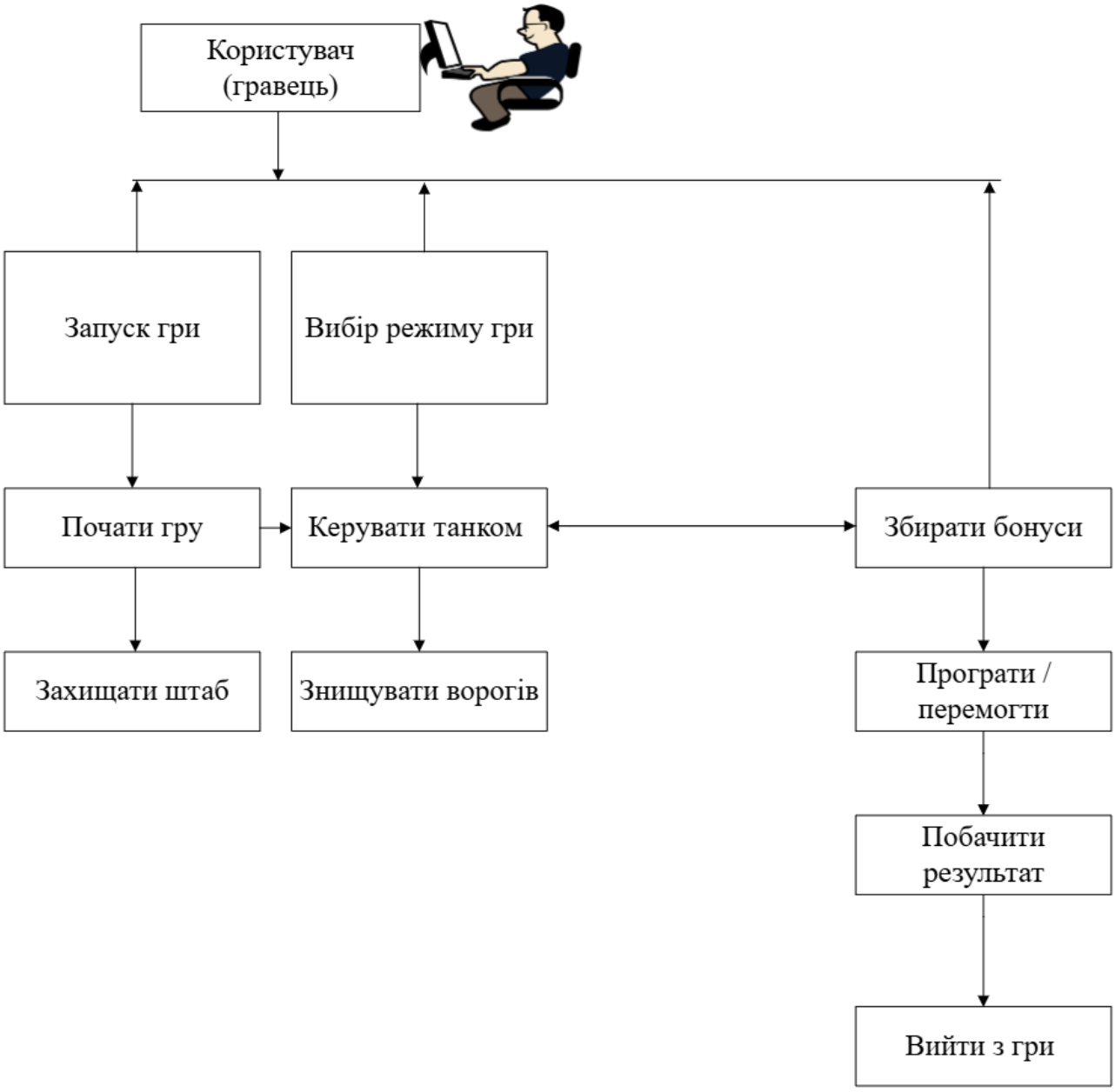

Рисунок 2.1 – Діаграма варіантів використання додатку

2.2. Проєктування діаграм активностей, послідовностей та класів

Після ініціалізації файлу завантаження гри, гравця зустрічає мінімалістичне меню гри, де є кнопки для початку гри та завершення роботи додатку. У разі вибору опції почати грати користувача одразу кидає до виру подій, грає динамічна музика, промальовується карта, спавняться противники.

Далі гравець вже перебуває безпосередього в грі, де його дії (рух та стрільба) впливають на результат рівня, який може закінчитись перемогою (всі противники знищені), або ж поразкою (знищення танка гравця, коли в нього немає додаткових життів, або ж руйнування головного штабу), після чого зявляється відповідний екран перемоги чи поразки.

Основний алгоритм роботи додатку зображено на рисунку 2.2.

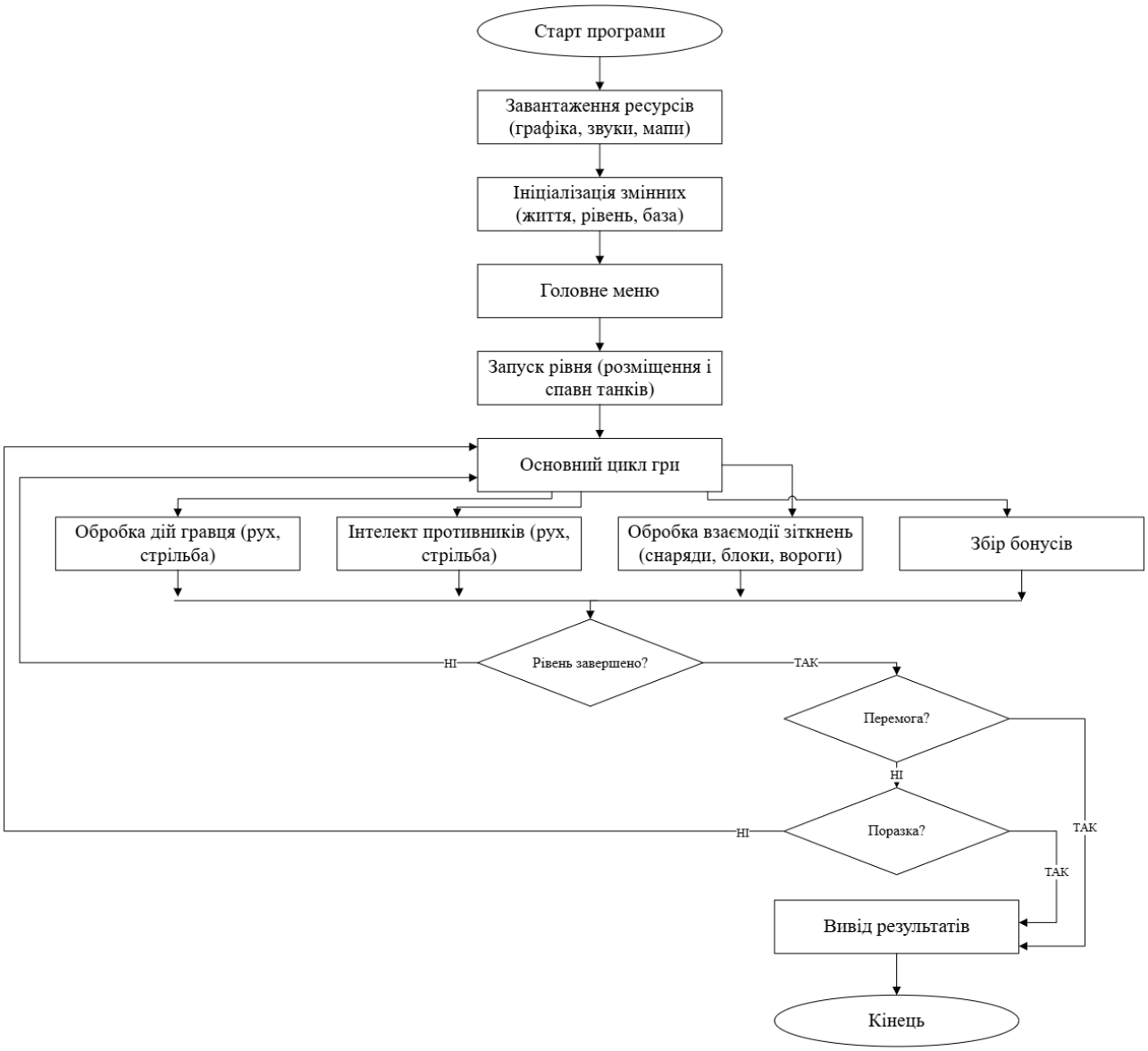

Рисунок 2.2 – Блок-схема алгоритму роботи гри «Танкові бої»

Після намічення копцепції жанру і стилю та попередньо функціоналу, наступним кроком стало програмування логіки додатку відповідно до вигаданих правил. Спершу логічно створити оточення для чого оголошуємо клас Block, де

буде створено і налаштовано функції і опції усіх пасивних об’єктів оточення. До цього класу входитимуть підкласи: CityBlock (тобто штаб); BreakableBlock (цегляна стіна, яку можна знищити пострілом); GameFieldWallBlock (обмежуваний периметр «бойових дій»); BushBlock (зелені хащі серед яких техніка замаскована, але вільно пересувається); SteelWallBlock (невразлива до пострілів стальна стіна). Підкласи класу оточення Block та їх звязки показані на рисунку 2.3.

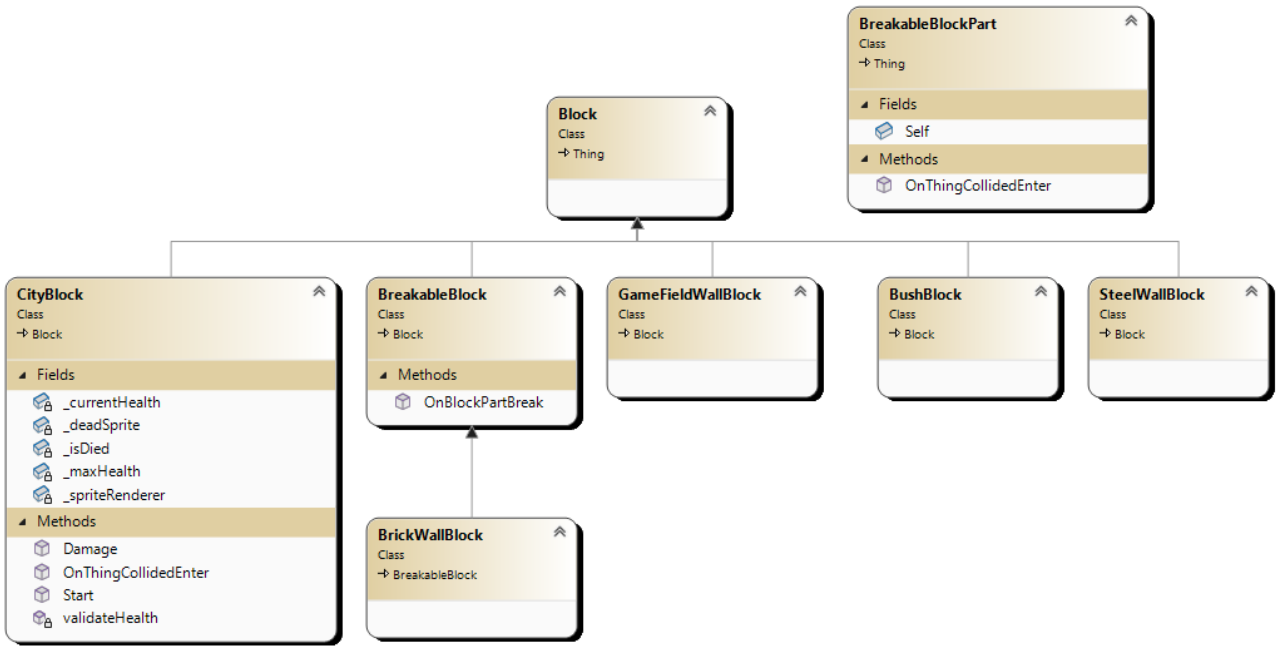

Рисунок 2.3 – Діаграма класу Block

Наступним створено клас Entity, який у підкласах задає характеристики снарядів, якими стріляють танки (підклас Bullet); підклас Tank, яких є базовим і якого формуються підкласи PlayerTank та EnemyTank. До цього класу також увішли підкласи: EnemyTank – що відповідає за “штучний інтелект” та характеристики танків ворогів; Explosion – ефекти руйнувань під час зіткнень та пострілів та PlayerTank – характеристики танку гравця.

Далі в черзі створення класу Game, що реалізовуватиме спавн (тобто появу) танків ворогів, керуватиме хвилями ворогів на рівні, та рахуватиме і записуматиме статистику гравця. Після чого додаємо клас Gun з підкласами EnemyGun і PlayerGun, що відповідатимуть за пушки танків противників та

гравця відповідно. Це клас задає дальність польоту снаряду та звук пострілу. І останнім, але не менш важливим, є клас SceneEvent, де закладена логіка тригеру меню та подій у грі, такі як поразка (через знищення танку гравця і окремо через знищення штабу), перемога, закінчення рівня, виклик меню та ін. Повну діаграму основних класів, їх взамодію та спадковіть зведнено на рисунку 2.4.

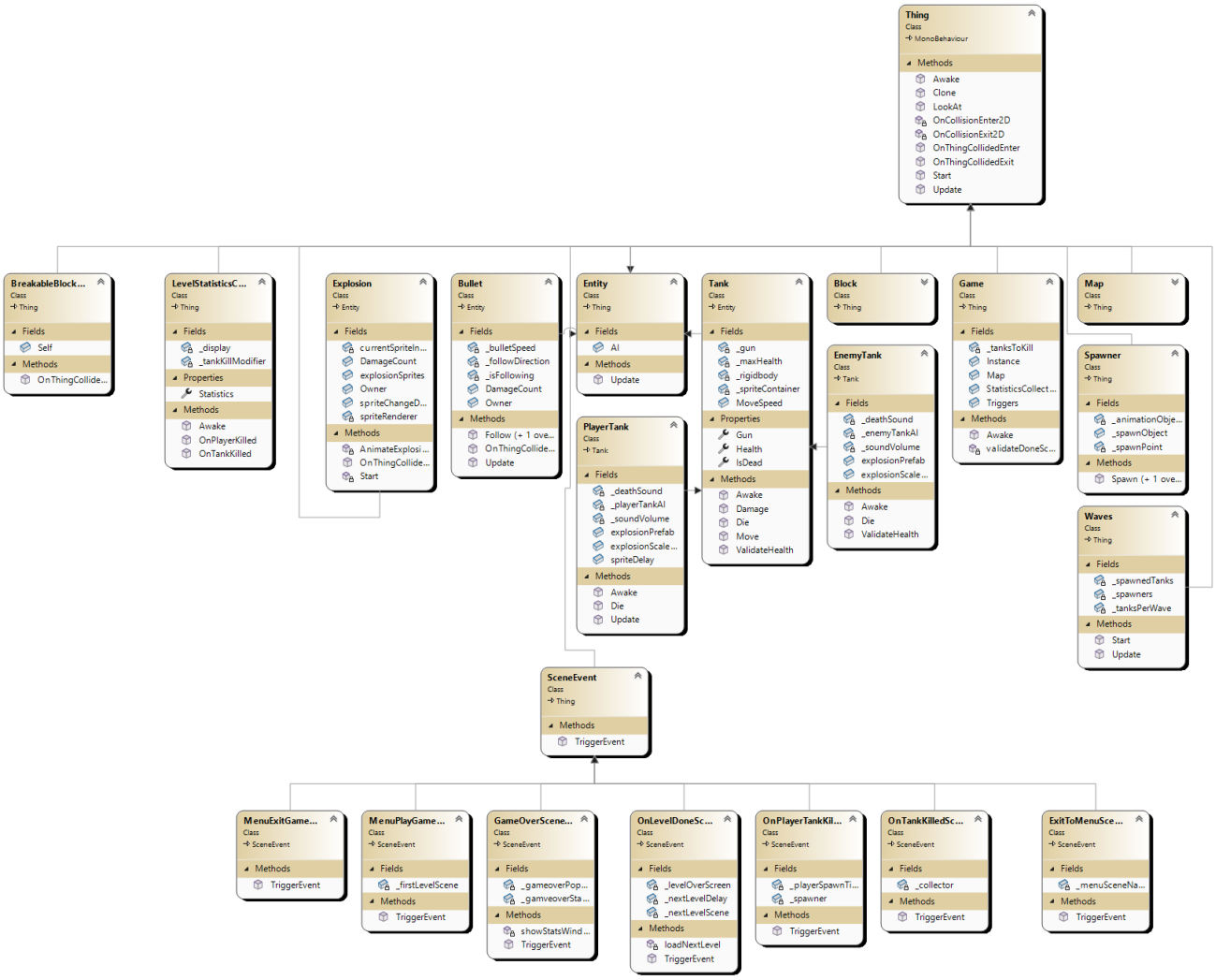

Рисунок 2.4 – Діаграма основних класів додатку «Танкові бої»

Також було створено мастер клас Thing, що запускає гру відповідає за оновлення так поведінку техніки, зокрема колізію та вигляд згори.

2.3. Етапи розробки системи. Розробка алгоритму конвертації програмного продукту

Враховуючи інтереси потенційних користувачів програми, аналізуючи їх

потреби, а також цілі та задачі розробки було вирішено реалізовувати гру в середовищі Unity, використовуючи в мову програмування C# та користуючить Visual Studio.

Початковим етапом розробки програмного забезпечення в даному випадку буде створення материнького класу танку, який можна успадкувати як танком гравця так і танком супротивника. Це реалізується у коді наступним чином:

```
using Guns;
using System;
using UnityEngine;
using GameUtils;

namespace Entities
{
    public class Tank : Entity
    {
        public Gun Gun
        {
            get
            {
                return _gun;
            }
            set
            {
                _gun = value;
                _gun.Init(this);
            }
        }
        public float MoveSpeed;
        public int Health { get; private set; }
        public bool IsDead { get; private set; }
```

```
[SerializeField] private int _maxHealth;
[SerializeField] private Rigidbody2D _rigidbody;
[SerializeField] private Transform _spriteContainer;
[SerializeField] private Gun _gun;

public override void Awake()
{
    base.Awake();

    Health = _maxHealth;

    if (_gun != null)
    {
        _gun.Init(this);
    }
}
public virtual void Move(Vector2 moveVector)
{
    _rigidbody.velocity = (moveVector * MoveSpeed);

    if (moveVector.x != 0 || moveVector.y != 0)
    {
        LookAt(moveVector);
    }
}
        public virtual void Damage(int damageCount, Entity damageOwner)
{
    if(damageCount <= 0)
    {
```

```
                throw new ArgumentException("Damage must be greater than 0");
            }

            Health -= damageCount;

            ValidateHealth(damageOwner);
        }
        public virtual void ValidateHealth(Entity validateOwner)
        {
            if (Health <= 0)
            {
                Die(validateOwner);
            }
        }
        public virtual void Die(Entity deathOwner)
        {
            IsDead = true;

            Destroy(gameObject);
        }
    }
}
```

Таким чином характеристики танку гравця кодуватимуться:

```
using GameUtils;
using UnityEngine;

namespace Entities
{
    public class PlayerTank : Tank
    {
```

```
[SerializeField] private PlayerTankAI _playerTankAI;
[SerializeField] private AudioClip _deathSound; // Звук смерті гравця
[SerializeField] private float _soundVolume = 1f; // Гучність звуку смерті
public GameObject explosionPrefab; // Префаб вибуху
public float explosionScaleMultiplier = 1.5f;
public float spriteDelay = 1f;


public override void Awake()
{
   base.Awake();

   AI = _playerTankAI;
   AI.Init(this);
}

public override void Update()
{
   base.Update();
}

public override void Die(Entity deathOwner)
{
   // Програємо звук смерті гравця в позиції танка
   if (_deathSound != null)
   {
      AudioSource.PlayClipAtPoint(_deathSound, transform.position,
_soundVolume);
   }
```

```
            // Створюємо вибух на місці танка
            GameObject explosion = Instantiate(explosionPrefab,
transform.position, Quaternion.identity);
            //explosion.GetComponent<Explosion>().Owner = this;
            explosion.GetComponent<Explosion>().spriteChangeDelay =
spriteDelay;
            explosion.transform.localScale = transform.localScale *
explosionScaleMultiplier;

            Game.Instance.Triggers.OnPlayerKilled.Invoke();

            base.Die(deathOwner);
        }
    }
}
```

А танка противника в свою чергу:

```
using GameUtils;
using UnityEngine;

namespace Entities
{
    public class EnemyTank : Tank
    {
        [SerializeField] private EnemyTankAI _enemyTankAI;
        [SerializeField] private AudioClip _deathSound; // Звук смерті
        [SerializeField] private float _soundVolume = 1f; // Гучність звуку смерті
        public GameObject explosionPrefab; // Префаб вибуху
        public float explosionScaleMultiplier = 1.5f;

        public override void Awake()
```

```
{
    base.Awake();
    AI = _enemyTankAI;
    AI.Init(this);
}

public override void ValidateHealth(Entity validateOwner)
{
    if (Health <= 0)
    {
        if (validateOwner != null)
        {
            if ((validateOwner as Bullet).Owner != null)
            {
                if ((validateOwner as Bullet).Owner is PlayerTank)
                {
                    Die((validateOwner as Bullet).Owner);
                }
            }
        }
    }
}

public override void Die(Entity deathOwner)
{
    // Програємо звук смерті в позиції танка
    if (_deathSound != null)
    {
        AudioSource.PlayClipAtPoint(_deathSound, transform.position,
_soundVolume);
```

```
            }

            // Створюємо вибух на місці танка
            GameObject explosion = Instantiate(explosionPrefab, transform.position, Quaternion.identity);
            //explosion.GetComponent<Explosion>().Owner = this;
            explosion.transform.localScale = transform.localScale * explosionScaleMultiplier;

            Game.Instance.Triggers.OnTankKilled.Invoke();

            base.Die(deathOwner);
        }
    }
}
```

Завдання шкоди ворогам і об’єктам фіксується і модерується наступним чином:

```
using UnityEngine;

namespace Entities
{
    public class Bullet : Entity
    {
        [HideInInspector] public Entity Owner;
        public int DamageCount = 1;

        [SerializeField] private float _bulletSpeed;
        private Vector2 _followDirection;
        private bool _isFollowing;
        public virtual void Follow(Vector2 followDirection)
```

```
        {
            _isFollowing = true;
            _followDirection = followDirection;
        }

        public virtual void Follow(Vector2 followDirection, Entity owner)
        {
            _isFollowing = true;
            _followDirection = followDirection;
            Owner = owner;
        }

        public override void Update()
        {
            base.Update();

            if (_isFollowing)
            {
                LookAt(_followDirection);
                transform.Translate((transform.right * -_followDirection.x +
transform.up * _followDirection.y) * _bulletSpeed * Time.deltaTime);
            }
        }

        public override void OnThingCollidedEnter(Thing thing)
        {
            base.OnThingCollidedEnter(thing);

            if(Owner != null)
            {
```

```
                if(thing != Owner)
                {
                    try
                    {
                        (thing as Tank).Damage(DamageCount, this);
                    }
                    catch
                    {

                    }

                    Destroy(gameObject);
                }
            }
        }
    }
```

Наступний крок це програмування методу, який описує властивості снаряду:

```
using UnityEngine;


namespace Entities
{
    public class Bullet : Entity
    {
        [HideInInspector] public Entity Owner;
        public int DamageCount = 1;


        [SerializeField] private float _bulletSpeed;
```

```
private Vector2 _followDirection;
private bool _isFollowing;

/// <summary>
/// Stars bullet to follow direction
/// </summary>
/// <param name="followDirection"></param>
public virtual void Follow(Vector2 followDirection)
{
    _isFollowing = true;
    _followDirection = followDirection;
}

/// <summary>
/// Stars bullet to follow direction with owner
/// </summary
/// <param name="followDirection">Bullet move direction</param>
/// <param name="owner">Owner of bullet</param>
public virtual void Follow(Vector2 followDirection, Entity owner)
{
    _isFollowing = true;
    _followDirection = followDirection;
    Owner = owner;
}

public override void Update()
```

```
        {
            base.Update();

            if (_isFollowing)
            {
                LookAt(_followDirection);
                transform.Translate((transform.right * -_followDirection.x +
transform.up * _followDirection.y) * _bulletSpeed * Time.deltaTime);
            }
        }

        public override void OnThingCollidedEnter(Thing thing)
        {
            base.OnThingCollidedEnter(thing);

            if(Owner != null)
            {
                if(thing != Owner)
                {
                    try
                    {
                        (thing as Tank).Damage(DamageCount, this);
                    }
                    catch
                    {
```

```
                }

                Destroy(gameObject);
            }
        }
    }
}
```

Створення ШІ противника виконано методом:

```
using System;
using UnityEngine;

namespace Entities
{
    [Serializable]
    public class EnemyTankAI : EntityAI
    {
        private Vector2 _moveVector;
        [SerializeField] private EnemyTankAIState _state;
        private EnemyTankAIState _lastState;

        [SerializeField] private IdleEnemyTankAIState _idleState;
        [SerializeField] private MovingToPointEnemyTankAIState
_movingToPointState;
        [SerializeField] private ShootingEnemyTankAIState _shootingState;
        private bool _isStateRunning;
```

```
public override void Init(Entity entity)
{
    base.Init(entity);

    _idleState.Init(Self);
    _movingToPointState.Init(Self);
    _shootingState.Init(Self);

    setNewAiState();
}
public override void UpdateAI()
{
    base.UpdateAI();

    if(!_isStateRunning) setNewAiState();

    _idleState.UpdateState();
    _movingToPointState.UpdateState();
    _shootingState.UpdateState();
}

private void setNewAiState()
{

    if (_isStateRunning) return;
```

```
            Array values = Enum.GetValues(typeof(EnemyTankAIState));

            EnemyTankAIState randomBar =
(EnemyTankAIState)values.GetValue(UnityEngine.Random.Range(0,
values.Length));

            if(_lastState == randomBar)
            {
                _state = _state = EnemyTankAIState.Idle;
            }
            else
            {
                _state = randomBar;
            }

            _lastState = _state;

            controllTankStateSwitch();
        }

        private void controllTankStateSwitch()
        {
            _isStateRunning = true;
```

```
        switch (_state)
        {
            case EnemyTankAIState.Idle:
                RunIdleState();
                break;
            case EnemyTankAIState.MovingToPoint:
                RunMovingToPointState();
                break;
            case EnemyTankAIState.Shooting:
                RunShootingState();
                break;
            default:
                RunIdleState();
                break;
        }
    }

    public virtual void RunIdleState()
    {
        _idleState.RunState(OnStateEnd);
    }
    public virtual void RunMovingToPointState()
    {
        _movingToPointState.RunState(OnStateEnd);
    }
    public virtual void RunShootingState()
```

```
        {
            _shootingState.RunState(OnStateEnd);
        }

        public virtual void OnStateEnd()
        {
            _isStateRunning = false;
            //setNewAiState();
        }
    }

    public enum EnemyTankAIState
    {
        Idle,
        MovingToPoint,
        Shooting
    }
}
```

Реалізацію вибухів покладено на метод:

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

public class Explosion : Entity
{
    [HideInInspector] public Entity Owner;
```

```
    public int DamageCount = 5;

    public Sprite[] explosionSprites; // Масив спрайтів для вибуху
    private SpriteRenderer spriteRenderer;
    private int currentSpriteIndex = 0;
    public float spriteChangeDelay = 0.5f; // Затримка між спрайтами

    private void Start()
    {
        spriteRenderer = GetComponent<SpriteRenderer>();
        StartCoroutine(AnimateExplosion());
    }

    private IEnumerator AnimateExplosion()
    {
        while (currentSpriteIndex < explosionSprites.Length)
        {
            spriteRenderer.sprite = explosionSprites[currentSpriteIndex];
            currentSpriteIndex++;
            yield return new WaitForSeconds(spriteChangeDelay); // Затримка між
спрайтами
        }

        Destroy(gameObject); // Знищуємо об'єкт після завершення анімації
    }

    public override void OnThingCollidedEnter(Thing thing)
    {
        base.OnThingCollidedEnter(thing);
        try
```

```
            {
                (thing as Tank).Damage(DamageCount, this);
            }
            catch
            {
                // Ловимо виняток, якщо thing не є Tank
            }

            // Знищуємо об'єкт, якщо він не був знищений раніше
            if (gameObject != null)
            {
                Destroy(gameObject);
            }
        }
    }
```

Якщо усіх противників на рівні знищено, показується екран перемоги на рівні з кільстю анігільованих танків противника, інакше – екран поразки.

Щодо візуального оформлення: інтерфейс гри-акради має бути мінімалістичним, що не відволікати гравці від драйвового геймплею. Особливо важливо дотримуватися аудіо-візуальної стилістики, яка відповідає жанровим особливостям, щоб підкреслити сильні сторони проєкту і не зіпсувати вайб. Графічні елементи не повинні бути надто деталізованими, проте анімація – має бути динамічною і функціональною.

Висновки до розділу 2.

У другому розділі дипломної роботи було розглянуто ключові кроки проєктування та реалізації програмного продукту у жанрі аркадного симулятора з виглядом зверху, натхненного класичним ігровим досвідом. Першочергову увагу сфокусовано на формуванні функціональних сценаріїв застосування програми, що охоплюють базові дії користувача – від запуску до виходу з гри. Детерміновано, що гравець – єдиний актор системи, саме тому процедури

взаємодії побудовані навколо найпростішого, інтуїтивного та водночас захопливого геймплею.

Задля кращого розуміння структури додатку було побудовано діаграму варіантів використання та алгоритмічну блок-схему, що демонструє загальну логіку гри. На наступному етапі розробки акцент перенесено на об’єктно-орієнтоване моделювання внутрішньої архітектури гри. Зокрема, створено класи, прописано їх ієрархію, серед яких виділено ключові сутності: Tank, PlayerTank, EnemyTank, Bullet, Gun, Explosion, SceneEvent, що визначають вихідні параметри та взаємодіють відповідно до заданих умов та тригерів.

Особливу увагу приділено реалізації прото-ШІ супротивників. Поведінка ворожих танків ґрунтується на змінних станах (Idle, MovingToPoint, Shooting), які перемикаються випадковим чином, забезпечуючи надзвичайну варіативність ігрових ситуацій. Це дозволяє досягти ефекту непередбачуваності дія противника до подій на полі бою.

Окрім вищезгаданого, було також впроваджено й логіку завдання шкоди, знищення об'єктів та відтворення візуальних і звукових ефектів для вибухів та пострілів. Ці елементи підкреслюють графічний стиль та посилюють емоційне залучення гравця.

Щодо візуалу – розробка інтерфейсу базувалася на принципах мінімалізму та жанрової стилізації. Основна мета – не перевантажити користувача графічними елементами, при цьому зберігаючи візуальну цілісність і відповідність жанру гри-аркади.

Як результат, у цьому розділі було сформовано логічну, структуровану основу проєкту, закладено програмні, аудіовізуальні та ігроладні основи, реалізовано ключові елементи ігрового процесу, що дозволяє перейти до тестування, відлагодження та фінального вдосконалення ПЗ на наступному етапі роботи.

РОЗДІЛ 3. ОПИС РОБОТИ ПРОГРАМНОГО ДОДАТКУ, ЙОГО ВСТАВНОВЛЕННЯ, КОРИСТУВАННЯ ТА ТЕСТУВАННЯ

3.1. Керівництво встановлення програмного продукту

Розроблений додаток – ігрова аркадна симуляція танкових боїв з видом зверху реалізовано на програмних потужностях середовища розробки Unity з використанням Windows Forms. Додаток дає змогу насолоджуватись динамічним ігроладом, битви з противником, руйнування структур, різні рівні в тому числі і з 2-ма боссами (значно посиленими ворогами).

Системні вимоги:

- ✓ ОС: від Windows 10;
- ✓ Процесор: Intel Core i5 чи подібний від інших розробників;
- ✓ RAM: > 128 МБ;
- ✓ Вільне місце на диску: > 150 МБ;

Алгоритм інсталювання:

- Завантажити коди елементів гри (представлено в додатку А);
- Відкрити Visual Studio та Unity;
- Обрати варіант "Відкрити проект або рішення" у мейн меню;
- Обрати теку, де розташовані дані проєкту гри, та "Відкрити" їх.

Програма буде дистрибуватись вже з готовим робочим білдом (збіркою готовою до гри), тому вище описані дії треба виконати, якщо необхідно редагувати вже сам проєкт.

Активація та реєстрація не потрібні, оскільки гра розповсюджується безкоштовно. При перевірка критично переконатись, що проект компілюється та запускається без критичних помилок у Unity.

Підтримка – при виникненні проблем чи помилок з ПЗ, необхідно зв’язатись з автором проєкту, після спроби виправити проблему самостійно.

При виникненні помилок при встановленні чи використанні додатку, необхідно:

- Переконатись, що всі вимоги до системи задовільняються;

- Перевірити, чи Visual Studio та Unity встановлено правильно з усіма оновленнями;
- Перезавантажити ПК та перевстановити додаток ще раз;
- Звернутись до документації та/або форумів спільноти Unity для отримання цільової довідки та саппорту.

Якщо нічого з переліченого не допомагає, і проблеми все ще виникають, необхідно звернутись до розробника чи служби підтримки для отримання додаткової допомоги.

3.2. Керівництво користування програмним продуктом

Для ініціалізації запуску додатку, необхідно скористатись двічі клікнути на файл «Tankovi boi», після короткого завантаження користувача бачить основне меню гри (рисунок 3.1.).

Рисунок 3.1 – Головне меню гри «Танкові бої»

Оскільки гра має мінімалістисний стиль в меню лише 2 активних кнопки «Грати» та «Вихід». Натискання першої переносить гравця безпосередньо до гри початок якої відображено на рисунку 3.2., а «Вихід» ініціює закриття вікна програми.

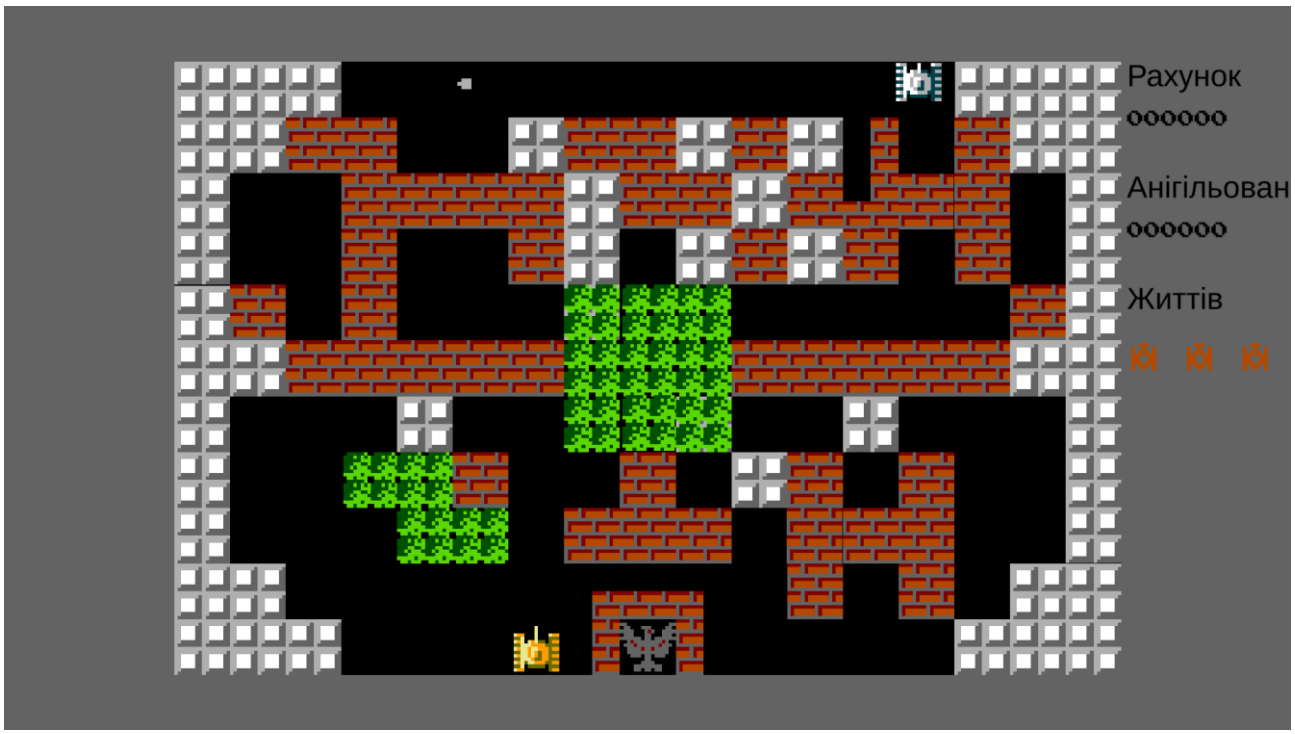

Рисунок 3.2 – Основний екран гри-аркади

Модель танку гравця пофарбовано золотистим кольором, а противників сріблястим, щоб їх можна було легко розрізнити. Керування машиною максимально просте, лище рух і посріли, управління передбачає використання лише клавіатури, клавіатури і часково миші, або геймпаду. Танк рухається класично вліво, вправо, доверху на донизу при натисканні відповідних стрілок на клавіатурі, чи клавіш WASD та пробілу чи ЛКМ для пострілів, цікавим нововеленням стала можливіть рухатись по діагонаді як показана на рисунку 3.3. В деякі геймплейні моменти це може надати неабияку тактичну перевагу гравцеві. Використовуючи просте та лаконічне керування ціль гравця – знищити (анігілювати) усіх танки противника на рівні, після чого гра перекидає показує екран завершення рівня (рисунок 3.4) і готує наступний рівень для проходження. Якщо ж у гравця закінчились додаткові танки (зазвичай він має 2 додаткових і одне основне), що показується на зліва на палелі в індикаторі «Життів»; або ж знищено штаб гравця, який знаходиться внизу в цетрі, оточений цегляною стіною – гравець зазнав поразки, показується відповідне повідомлення, приклад якого показано на рисунку 3.5, та екран поразки (рисунок 3.6), що дозволяє лише

повернутись до головного меню, що кидає гравцю виклик.

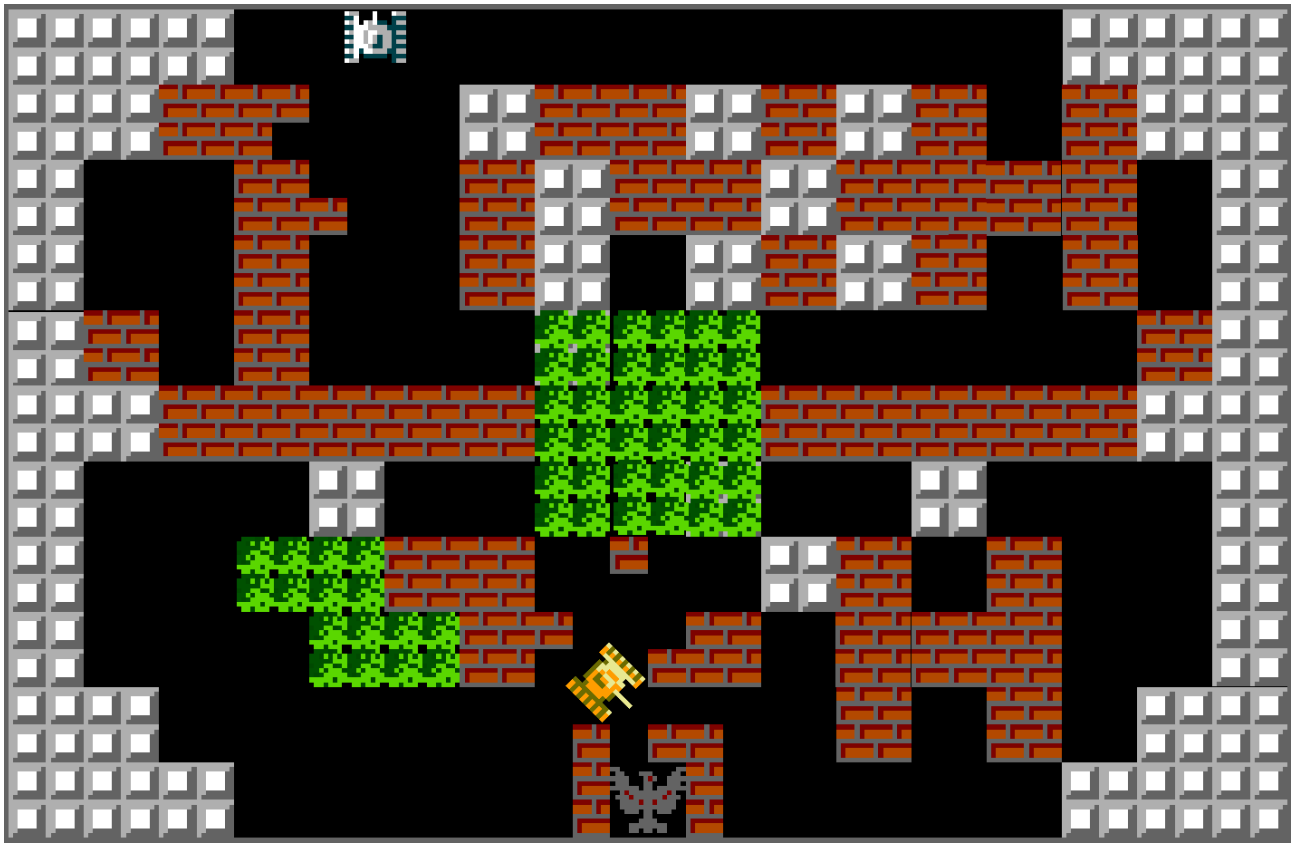

Рисунок 3.3 – Діагональний рух машини гравця

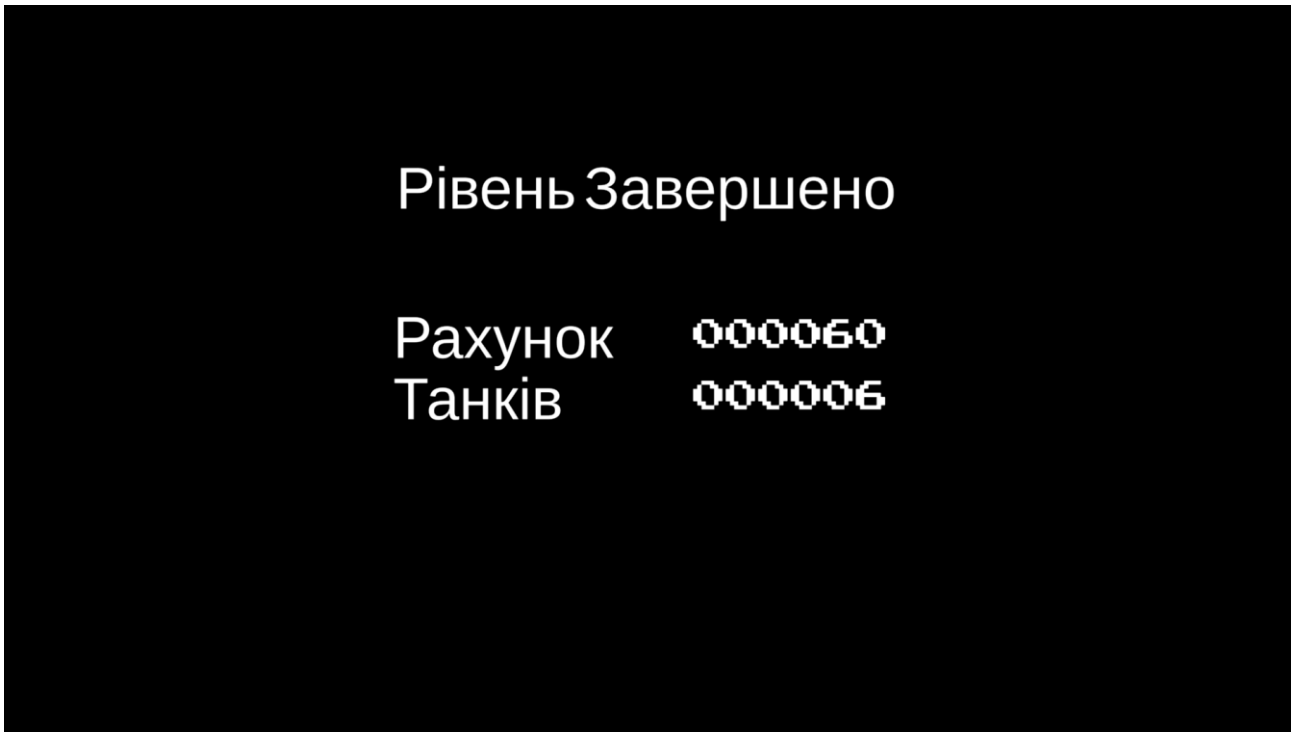

Рисунок 3.4 – Екран успішного завершення рівня

Рисунок 3.5 – Повідомлення про поразку гравця

Рисунок 3.6 – Екран поразки гравця

Останній рівень, що проілюстровано на рисунку 3.7 вирізняється двома

великими танками-босами, які мають 5-ти кратну шкоду і сильно збільшену кількість здоров'я. У разі якщо гравець здолає їх, демонструється екран остаточної перемоги як на рисунку 3.8.

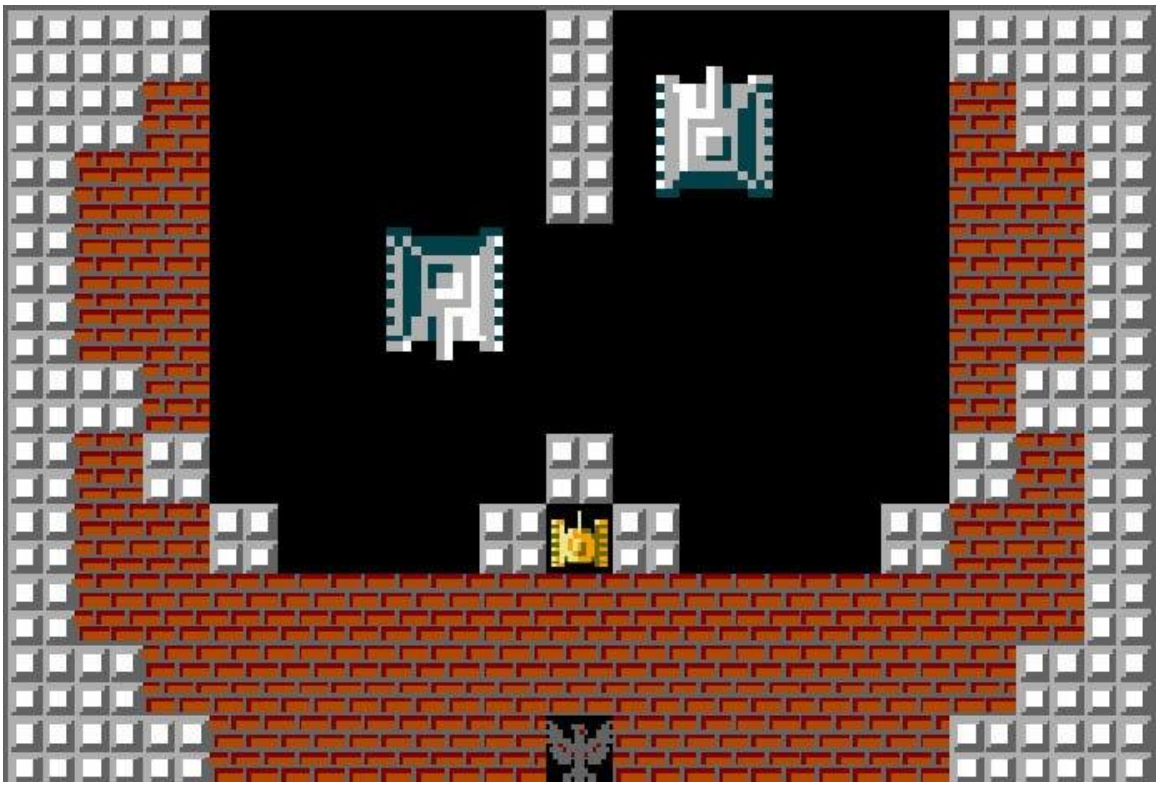

Рисунок 3.7 – Останній рівень гри

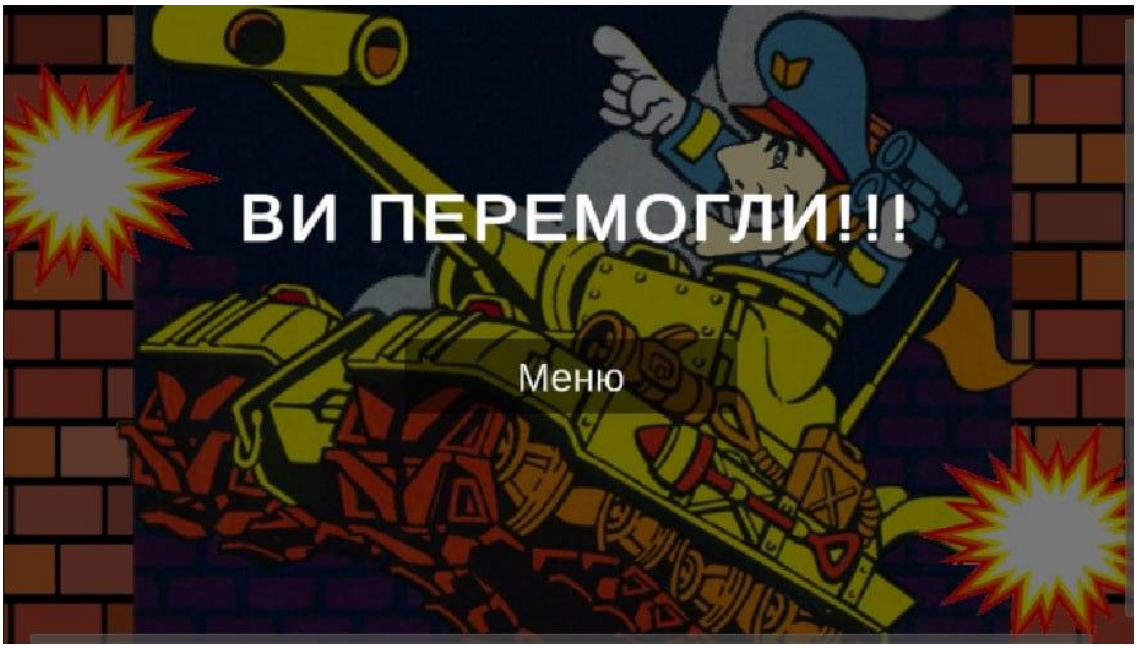

Рисунок 3.8 – Екран остаточної перемоги

Повідомлення про перемогу, поразку, кількість знищених противників та

інші елементи, спрямовані на урізноманітнення ігрового досвіду, заохочення гравця спробувати свої сили ще раз, що стимулює робити висновки та удосконалювати ігрові стратегії та тактики. Ці елементи значно покращують емоційний зв'язок гравця з віртуальним світом гри.

3.3. Тестування роботи програмного продукту

Перевірка класів процедур і методів програми «Танкові бої» з метою забезпечення її правильного та оптимального функціонування для задоволення потреб гравця. Аналізуючи і оцінюючи результать тестування в «ріал тайм», можна заключити, що тестування проведено раціонально, дотримуючись часових рамок та критеріїв. В таблиці 3.1. наведено критерії, за якими була проведено тестування роботи додатку.

Таблиця 3.1

Тестування роботи додатку «Танкові бої»

Критерії	Виконання
Зміна вікон	✓
Калькулятор очок	✓
Екрани перемоги та поразки	✓
«ШІ» противника	✓
Коректне завершення гри	✓

Висновки до розділу 3.

У розділі 3 було детально описано процес інсталяцію, застосування та тетування роботи розробленого ПЗ аркадної гри «Танкові бої» з виглядом зверху. Було описано базовий алгоритму встановлення, який включає запуск проєкту через середовище розробки Unity та Visual Studio. При цьому враховано можливість використання заздалегідь зібраного білду гри, що робить запуск простим та зручним навіть для непідготовлених юзерів. Системні вимоги – помірні, що робить програму доступною для широкого кола юзерів.

Подано керівництво використання програми, яке показує функції в грі, від запуску меню до завершення гри. Оболонка гри має мінімалістичну структуру, що дозволяє зконцентруватись на геймплеї. Механіка керування реалізована нескладно та інтуїтивно – підтримується клавіатура, миша або геймпад, а можливі дії, зокрема рух, стрільба та ухилення, виконуються у режимі «real time». Важливим аспектом є впровадження діагонального руху, що додає тактичної глибини ігровому процесу.

Прихованим сюрпризом для гравця стане фінальний рівень з «босами», які мають підвищені характеристики шкоди та здоров’я. Це кидає виклик гравцеві та слугує чудовим кульмінаційним моментом проходження гри. Візуальні повідомлення про перемогу, поразку та прогрес анігіляцію значно підсилюють емоційний зв'язок користувача з грою, сприяють залученню та збільшують імовірність повторного проходження.

У даному розділі розглянуто і процес тестування додатку. Було проведено перевірку основних функцій: зміни екранів, системи підрахунку очок, механіки бою, штучного інтелекту супротивника та завершення гри. Усі протестовані компоненти показали стабільну та коректну роботу, що свідчить про грамотну реалізацію програмної логіки та зв’язків між елементами.

Таким чином, у третьому розділі продемонстровано повноцінну завершену програму для усіх етапів її життєдіяльності. Результати підтверджують працеспроможність ПЗ, його доступність, зручність у використанні та функціональну відповідність визначеним критеріям.

ВИСНОВКИ

У даній дипломній роботи було здійснено повний цикл аналіз, проєктування, реалізація та тестування аркадної гри «Танкові бої» з виглядом зверху в режимі реального часу. Основна мета роботи – дослідження подібних та створення програми аркадної гри, що включає простоту керування, динамічний ігролад та сучасні рішення.

У першому розділі проведено глибокий аналіз жанру ігор-аркад та сформульовано ТЗ. Було розглянуто приклади схожих додатків, зокрема Tiny Tanks та Super Tank Battle, що дозволило підкреслити ключові жанрові характеристики та орієнтувати розробку на перевірену гейм-дизайнерську модель. На основі аналізу було вирішено використовувати ігровий рушій Unity, мову програмування C# та середовище розробки Visual Studio Community, що є гарним поєднанням для реалізації кросплатформового та візуально орієнтованого продукту.

Другий розділ присвячено проєктуванню гри. У розділі було детально описано функціональні можливості додатку, сценарії використання та основний алгоритм роботи гри. Створено діаграму варіантів використання додатку, діаграму класів та активностей, що демонструють структурну організацію гри та взаємозв’язки між її класами, підкласами та методами. Детально розписано логіку гри, зокрема класів Tank, Bullet, Gun та Explosion, а також доданого інтелекту противника, що забезпечує поведінкову варіативність.

У третьому розділі було зосереджено увагу на практичному аспекті розробки – інсталюванні, експлуатації та тестуванні додатку. Було описано процес інсталяції ПЗ, визначено вимоги до системи та надано інструкції для запуску гри. Інструкції для користувача розглядають повний цикл взаємодії з додатком – від запуску до фінального бою з босами. Результати тестування засвідчили коректну роботу всіх ключових функцій гри: переходів між екранами, механіки бою, логіки перемоги/поразки та інтелекту противників. Ігровий процес показав себе стабільним, інтуїтивно зрозумілим, а інтерфейс –

лаконічним та ергономічно вивіреним.

Таким чином, у дипломній роботі було успішно реалізовано повнофункціональну програму для гри-акради з дотриманням усіх етапів життєвого циклу ПЗ. Проєкт має потенціал до масштабування, модернізації та застосування у освітніх, індивідуальних чи комерційних цілях. Результати пошуку підтверджують актуальність жанру, ефективність обраних інструментів та наявність попиту серед широкої категорії гравців на подібні продукти.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

PvE (*Player versus Environment*) – противником гравця є програма.

PvP (*Player versus Player*) – противником гравця є інший гравець.

FPS (*Frames Per Second*) – кількість кадрів за період в 1 секунду.

RAM (*Random Access Memory*) – оперативна пам'ять.

ООП – об’єктно-орієнтоване програмування.

ОС – операційна система.

ПЗ – програмне забезпечення.

ПК – персональний комп’ютер.

ТЗ – технічне завдання.

ШІ – штучний інтелект.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chen, Huai-Chi. The Evolution of Arcade Gaming Culture, 2009.
2. King, D. L., Delfabbro, P. H., & Griffiths, M. D. (2010). Video Game Structural Characteristics: A New Psychological Taxonomy. International Journal of Mental Health and Addiction.
3. Hull, D. C., Williams, G. A., & Griffiths, M. D. (2013). Video game characteristics, motivations, and player performance. Computers in Human Behavior, 29(3), 1033–1040.
4. Реальний час проти покрокової механіки в навчальних іграх: веб-сайт. URL: https://www.filamentgames.com/blog/real-time-vs-turn-based-mechanics-learning-games/ (дата звернення: 11.03.2025).
5. Steam. Гра «Tiny Tanks»: веб-сайт. URL: https://store.steampowered.com/app/836850/Tiny_Tanks/ (дата звернення: 12.04.2025).
6. Google Play. Гра «Super Tank Battle»: веб-сайт. URL: https://play.google.com/store/apps/details?id=com.unknownprojectx.tank.free (дата звернення: 12.04.2025).
7. Microsoft Learn. Документація по C#: веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 11.03.2025).
8. Вікіпедія. C#: веб-сайт. URL: https://en.wikipedia.org/wiki/C_Sharp_(programming_language) (дата звернення: 11.03.2025).
9. Unity Technologies. Unity Manual: Scripting Overview: веб-сайт. URL: https://docs.unity3d.com/Manual/UnderstandingScripting.html (дата звернення: 10.03.2025).
10. Unity Asset Store. Офіційний маркетплейс ресурсів для Unity: веб-сайт. URL: https://assetstore.unity.com/ (дата звернення: 15.03.2025).
11. Microsoft Learn. Документація по Windows Forms: веб-сайт. URL: https://learn.microsoft.com/en-us/visualstudio/ide/create-a-visual-basic-winform-in-

visual-studio (дата звернення: 11.03.2025).

12. Microsoft Learn. Документація по ООП: веб-сайт. URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/tutorials/oop (дата звернення: 11.03.2025).

13. Нікітін С.О., Нікітіна, Л.О. Основи комп'ютерних ігор та ігрових програм: довідник модуля. Харків, 2018. 140 с.

14. Лугова Т.А., Блажко О.А. Проектування комп’ютерних ігор для навчання: навчальний підручник. Одеса, 2018. 209 с.

15. Michael Moore. Basics of Game Design. CRC Press, 2016. 400 с.

16. Donovan Tristan. Replay: The History of Video Games. Yellow Ant, 2010. 516 p.

17. Schell, Jesse. The Art of Game Design: A Book of Lenses. CRC Press, 2019. 600 p.

18. Rollings, Andrew, and Ernest Adams. Fundamentals of Game Design. New Riders, 2012. 672 p.

19. Bogost, Ian. Persuasive Games: The Expressive Power of Videogames. MIT Press, 2007. 464 p.

20. Adams, Ernest. Game Mechanics: Advanced Game Design. New Riders, 2012. 360 p.

ДОДАТКИ

Додаток А

Вихідний код гри

```
using System;
using UnityEngine;
public class Thing : MonoBehaviour, ICloneable
{
    public virtual void Awake()
    {

    }
    public virtual void Start()
    {

    }
        public virtual void Update()
    {

    }


    /// <summary>
    /// LookAt implimentation for 2D
    /// </summary>
    /// <param name="moveVector"></param>
    public virtual void LookAt(Vector2 lookPosition)
    {
        Vector3 vectorDifference = (new Vector3(lookPosition.x, lookPosition.y, 0) + transform.position) - transform.position;
        vectorDifference.Normalize();
```

```
        float rot_z = Mathf.Atan2(vectorDifference.y, vectorDifference.x) * Mathf.Rad2Deg;
        transform.rotation = Quaternion.Euler(0f, 0f, rot_z - 90);
    }

    /// <summary>
    /// OnCollisionEnter2D implimentation for 2D
    /// </summary>
    /// <param name="collision"></param>
    public virtual void OnThingCollidedEnter(Thing thing)
    {

    }

    /// <summary>
    /// OnCollisionExit2D implimentation for 2D
    /// </summary>
    /// <param name="collision"></param>
    public virtual void OnThingCollidedExit(Thing thing)
    {

    }

    /// <summary>
    /// Clones thing
    /// </summary>
    /// <returns></returns>
    public virtual object Clone()
    {
        return this.MemberwiseClone();
```

```
    }


    private void OnCollisionEnter2D(Collision2D collision)
    {
        Thing collidedThing;

        if (collision.gameObject.TryGetComponent<Thing>(out collidedThing))
        {
            OnThingCollidedEnter(collidedThing);
        }
        else
        {
            throw new MissingReferenceException("No [Thing] class on collided object");
        }
    }
    private void OnCollisionExit2D(Collision2D collision)
    {
        Thing collidedThing;

        if (collision.gameObject.TryGetComponent<Thing>(out collidedThing))
        {
            OnThingCollidedExit(collidedThing);
        }
        else
        {
            throw new MissingReferenceException("No [Thing] class on collided object");
        }
    }
}
```

```
namespace Blocks
{
    /// <summary>
    /// Base class of Block, but breakable
    /// </summary>
    public class BreakableBlock : Block
    {
        /// <summary>
        /// Called on Block break
        /// </summary>
        /// <param name="part">broken part</param>
        /// <param name="collidedThing">thing that breaks part</param>
        public virtual void OnBlockPartBreak(BreakableBlockPart part, Thing collidedThing)
        {
            Destroy(part.gameObject);
        }
    }
}
using Entities;

namespace Blocks
{
    /// <summary>
    /// Breakable part of block
    /// </summary>
    public class BreakableBlockPart : Thing
    {
        public BreakableBlock Self;

        public override void OnThingCollidedEnter(Thing thing)
```

```
        {
            base.OnThingCollidedEnter(thing);

            if (thing is Bullet || thing is Explosion)
            {
                Self.OnBlockPartBreak(this, thing);
            }
        }
    }
}
using UnityEngine;

namespace Entities
{
    /// <summary>
    /// Base class of all moveable objects
    /// </summary>
    public class Entity : Thing
    {
        [HideInInspector] public EntityAI AI;

        public override void Update()
        {
            base.Update();

            AI.UpdateAI();
        }
    }
}
using System;
```

```
using UnityEngine;

namespace Entities
{
    /// <summary>
    /// Base class of Entity AI
    /// </summary>
    [Serializable]
    public class EntityAI
    {
        [HideInInspector] public Entity Self;

        /// <summary>
        /// Initializing AI
        /// </summary>
        /// <param name="entity">Self entity</param>
        public virtual void Init(Entity entity)
        {
            Self = entity;
        }

        /// <summary>
        /// Update translator. Updatets AI every frame
        /// </summary>
        public virtual void UpdateAI()
        {

        }
    }
}
```

```
using System;
using UnityEngine.Events;
using UnityEngine;

namespace Entities
{
    /// <summary>
    /// Base struct class of Entity Statemachine
    /// </summary>
    [Serializable]
    public class EntityAIState
    {
        public bool IsActive { private set; get; }
        [HideInInspector] public Entity Self;

        [SerializeField] private UnityEvent onStateStop;

        /// <summary>
        /// Initializing self
        /// </summary>
        /// <param name="self"></param>
        public virtual void Init(Entity self)
        {
            Self = self;
        }

        /// <summary>
        /// Runs state
        /// </summary>
        /// <param name="callbackevent">Calling on state end</param>
```

```
        public virtual void RunState(UnityAction callbackevent)
        {
            IsActive = true;
            onStateStop.AddListener(callbackevent);
        }

        /// <summary>
        /// Called every frame AI update
        /// </summary>
        public virtual void UpdateState()
        {

        }

        /// <summary>
        /// Stops state. Call on end point of state
        /// </summary>
        public virtual void StopState()
        {
            IsActive = false;
            onStateStop.Invoke();
            onStateStop.RemoveAllListeners();
        }
    }
}
using System.Threading.Tasks;
using UnityEngine;

namespace GameUtils
{
```

```
/// <summary>
/// Base class of spawning things
/// </summary>
public class Spawner : Thing
{
    public Thing _spawnObject;

    [SerializeField] private Transform _spawnPoint;
    [SerializeField] private GameObject _animationObject;

    /// <summary>
    /// Spawns thing
    /// </summary>
    public virtual Thing Spawn()
    {
        Thing instance = Instantiate((Thing)_spawnObject.Clone());

        instance.transform.position = _spawnPoint != null ? _spawnPoint.position : transform.position;

        return instance;
    }

    /// <summary>
    /// Spawns thing with animation and delay
    /// </summary>
    /// <param name="animationTime">Delay (in ms)</param>
    public virtual async void Spawn(int animationTime)
    {
        _animationObject.SetActive(true);
```

```
            await Task.Delay(animationTime);

            _animationObject.SetActive(false);

            Spawn();
        }
    }
}
using GameUtils;
using UnityEngine;

namespace Statistics
{
    /// <summary>
    /// Collects statistics of Level e.g. tanks killed and other scores
    /// </summary>
    public class LevelStatisticsCollector : Thing
    {
        public StatisticsData Statistics { get; private set; } = new StatisticsData();

        [SerializeField] private StatisticsDisplay _display;
        [SerializeField] private int _tankKillModifier = 10;
        [SerializeField] private int _lives=3;

        public override void Awake()
        {
            base.Awake();

            Statistics.PlayerLivesCount = _lives;
```

```
        }

        public void OnTankKilled()
        {
            Statistics.TanksKilled++;
            Statistics.LevelScore += 1 * _tankKillModifier;

            _display.UpdateDisplay(Statistics);
        }

        public void OnPlayerKilled()
        {
            Statistics.PlayerLivesCount--;

            _display.UpdateDisplay(Statistics);

            if(Statistics.PlayerLivesCount <= 0)
            {
                Game.Instance.Triggers.OnPlayerTanksEnd.Invoke();
            }
        }
    }
}
using UnityEngine;

namespace Statistics
{
    /// <summary>
    /// Main data struct of statistics
    /// </summary>
```

```
    [SerializeField]
    public class StatisticsData
    {
        public int TotalScore;
        public int TanksKilled;

        public int LevelScore;
        public int PlayerLivesCount;
    }
}
using TMPro;
using UnityEngine;

namespace Statistics
{
    /// <summary>
    /// Displays statistics
    /// </summary>
    public class StatisticsDisplay : MonoBehaviour
    {
        [SerializeField] private TMP_Text[] _totalScoreTexts;
        [SerializeField] private TMP_Text[] _levelScoreTexts;
        [SerializeField] private TMP_Text[] _tanksKilledTexts;
        [Space]

        [SerializeField] private GameObject _playerRemainingTanksUIContainer;

        /// <summary>
        /// Updates statistics on UI
        /// </summary>
```

```
/// <param name="dataToSisplay"></param>
public void UpdateDisplay(StatisticsData dataToSisplay)
{
    foreach (TMP_Text item in _totalScoreTexts)
    {
        item.text = dataToSisplay.TotalScore.ToString("000000");
    }

    foreach (TMP_Text item in _levelScoreTexts)
    {
        item.text = dataToSisplay.LevelScore.ToString("000000");
    }

    foreach (TMP_Text item in _tanksKilledTexts)
    {
        item.text = dataToSisplay.TanksKilled.ToString("000000");
    }

    displayPlayerTanksCount(dataToSisplay);
}

private void displayPlayerTanksCount(StatisticsData dataToSisplay)
{
    foreach (Transform item in _playerRemainingTanksUIContainer.transform)
    {
        item.gameObject.SetActive(false);
    }

    for (int i = 0; i < dataToSisplay.PlayerLivesCount; i++)
    {
```

```
                _playerRemainingTanksUIContainer.transform.GetChild(i).gameObject.SetActive(true);
            }
        }
    }
}using Entities;
using System.Collections.Generic;
using UnityEngine;

namespace GameUtils
{
    public class Waves : Thing
    {
        [SerializeField] private Spawner[] _spawners;
        [SerializeField] private int _tanksPerWave;

        private List<Tank> _spawnedTanks = new List<Tank>();

        public override void Start()
        {
            base.Start();
        }

        public override void Update()
        {
            base.Update();

            if(_spawnedTanks.Count <= 0)
            {
                for (int i = 0; i < _tanksPerWave; i++)
                {
```

```
                Tank spawnedTank = _spawners[Random.Range(0, _spawners.Length)].Spawn() as Tank;
                _spawnedTanks.Add(spawnedTank);
            }

        }

        foreach (Tank item in _spawnedTanks)
        {
            if (item.IsDead)
            {
                _spawnedTanks.Remove(item);
                return;
            }
        }
    }
}
using UnityEngine;
using Statistics;

namespace GameUtils
{
    /// <summary>
    /// Main class of game. Contains Links to individual classes on scene. Game.Instance to get instance of Game
    /// </summary>
    public class Game : Thing
    {
        public static Game Instance;
```

```
        public GameTriggers Triggers;
        public Map Map;
        public LevelStatisticsCollector StatisticsCollector;

        [SerializeField] private int _tanksToKill = 10;

        public override void Awake()
        {
            base.Awake();

            Instance = this;
            Triggers.OnTankKilled.AddListener(validateDoneScore);
        }

        private void validateDoneScore()
        {
            if(StatisticsCollector.Statistics.TanksKilled >= _tanksToKill)
            {
                Triggers.OnLevelDone.Invoke();
            }
        }
    }
}
using System;
using UnityEngine.Events;

namespace GameUtils
{
    /// <summary>
```

```
    /// All key-event links
    /// </summary>
    [Serializable]
    public class GameTriggers
    {
        public UnityEvent OnCityBlockDied;
        public UnityEvent OnTankKilled;
        public UnityEvent OnPlayerKilled;
        public UnityEvent OnPlayerTanksEnd;
        public UnityEvent OnLevelDone;
    }
}
using Entities;
using UnityEngine;

namespace Guns
{
    /// <summary>
    /// Base class of Gun
    /// </summary>
    public class Gun : MonoBehaviour
    {
        [SerializeField] private Bullet _bulletPrefab;

        /// <summary>
        /// Self tank
        /// </summary>
        protected Tank Tank;

        /// <summary>
```

```
        /// Initializinng gun
        /// </summary>
        /// <param name="tank">Self tank</param>
        public virtual void Init(Tank tank)
        {
            Tank = tank;
        }

        /// <summary>
        /// Shoots bullet
        /// </summary>
        public virtual void Shoot()
        {
            Bullet bullet = SpawnBullet();
            bullet.transform.position = Tank.transform.position;
            bullet.Follow(Tank.transform.up, Tank);
        }

        /// <summary>
        /// Spawns bullet
        /// </summary>
        /// <returns>Bullet reference</returns>
        public virtual Bullet SpawnBullet()
        {
            Bullet bullet = Instantiate((Bullet)_bulletPrefab.Clone());
            bullet.transform.position = Tank.transform.position;

            return bullet;
        }
    }
```

```
}
using UnityEngine;
using UnityEngine.SceneManagement;

namespace SceneEvents
{
    public class ExitToMenuSceneEvent : SceneEvent
    {
        [SerializeField] private string _menuSceneName;

        public override void TriggerEvent()
        {
            base.TriggerEvent();

            SceneManager.LoadScene(_menuSceneName);
        }
    }
}
using System.Threading.Tasks;
using UnityEngine;

namespace SceneEvents
{
    public class GameOverSceneEvent : SceneEvent
    {
        [SerializeField] private GameObject _gameoverPopupShow;
        [SerializeField] private GameObject _gamveoverStatsWindow;

        public override void TriggerEvent()
        {
```

```
            base.TriggerEvent();

            _gameoverPopupShow.SetActive(true);

            showStatsWindow();
        }

        private async void showStatsWindow()
        {
            await Task.Delay(2000);

            _gamveoverStatsWindow.SetActive(true);
        }
    }
}
using UnityEngine;

namespace SceneEvents
{
    public class MenuExitGameSceneEvent : SceneEvent
    {
        public override void TriggerEvent()
        {
            base.TriggerEvent();

            Debug.Log("Exiting game");
            Application.Quit();
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.SceneManagement;

namespace SceneEvents
{
    public class MenuPlayGameSceneEvent : SceneEvent
    {
        [SerializeField] private string _firstLevelScene = "level01";

        public override void TriggerEvent()
        {
            base.TriggerEvent();

            SceneManager.LoadScene(_firstLevelScene);
        }
    }
}using System.Threading.Tasks;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace SceneEvents
{
    public class OnLevelDoneSceneEvent : SceneEvent
    {
        [SerializeField] private string _nextLevelScene;
        [SerializeField] private int _nextLevelDelay = 5000;
        [SerializeField] private GameObject _levelOverScreen;

        public override void TriggerEvent()
```

```
        {
            base.TriggerEvent();

            _levelOverScreen.gameObject.SetActive(true);

            loadNextLevel();
        }

        private async void loadNextLevel()
        {
            await Task.Delay(_nextLevelDelay);
            SceneManager.LoadScene(_nextLevelScene);
        }
    }
}
using GameUtils;
using UnityEngine;

namespace SceneEvents
{
    public class OnPlayerTankKilled : SceneEvent
    {
        [SerializeField] private Spawner _spawner;
        [SerializeField] private int _playerSpawnTime = 3000;

        public override void TriggerEvent()
        {
            base.TriggerEvent();

            _spawner.Spawn(_playerSpawnTime);
```

```
        }
    }
}
using UnityEngine;
using Statistics;

namespace SceneEvents
{
    public class OnTankKilledSceneEvent : SceneEvent
    {
        [SerializeField] private LevelStatisticsCollector _collector;

        public override void TriggerEvent()
        {
            base.TriggerEvent();

            _collector.OnTankKilled();
        }
    }
}
```