Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка 3D-гри на Unity

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

		Ткачук Р. С.
	(підпис)	(прізвище та ініціали)
Керівник		Бойко І. В.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю. М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М. Р.
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль
2024

АНОТАЦІЯ

Кваліфікаційна робота бакалавра на тему «Розробка 3D-гри на Unity» написана Ткачуком Романом Сергійовичем, студентом Тернопільського національного технічного університету імені Івана Пулюя, факультету комп'ютерно-інформаційних систем і програмної інженерії, кафедри програмної інженерії, спеціальності «Інженерія програмного забезпечення», групи СП-42.

Відомості про обсяг роботи: сторінок – 49, рисунків – 18, таблиць – 0, частин – 4, додатків – 4, посилань – 10, формул – 0.

Метою даної кваліфікаційної роботи бакалавра є проектування та розробка 3D-гри жанру «стратегія в реальному часі» під назвою «War of Technique». Варто наголосити що кінцевим результатом роботи є лише бета-версія гри і в подальшому вона підлягатиме великій кількості доопрацювань, оптимізацій та оновлень.

У даній грі користувач має можливість створити велику кількість власних військових баз, захоплювати ворожі бази, території, видобувні точки.

Гравцеві необхідно добувати ресурси для розширення військової бази та модернізації споруджень та транспорту. По всій мапі розкидані видобувні точки ворога, котрі поповнюють баланс ресурсів та матеріалів.

Основна бойова одиниця гри «War of Technique» - військова техніка різних типів, кожна з яких має свої переваги та недоліки. Військова техніка також підлягає покращенню її властивостей за рахунок певної кількості ресурсів.

Під час виконання роботи було пройдено крізь усі основні та необхідні етапи життєвого циклу програмного забезпечення, обрано методологію проектування, розробки, описано зв'язки між компонентами системи за допомогою UML-діаграм, проведено тестування результату виконання одного спринта за методологією Scrum.

ABSTRACT

The bachelor's qualification thesis on the topic "Development of a 3D game on Unity" was written by Roman Sergeyevich Tkachuk, a student of the Ternopil National Technical University named after Ivan Pulyu, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, specialty "Software Engineering", group SP-42.

Information about the volume of work: pages – 49, figures – 18, tables – 0, parts – 4, appendices – 4, references – 10, formulas – 0.

The purpose of this bachelor's thesis is to design and develop a 3D real-time strategy game called "War of Technique". It is worth emphasizing that the final result of the work is only a beta version of the game and in the future it will be subject to a large number of refinements, optimizations and updates.

In this game, the user has the opportunity to create a large number of his own military bases, capture enemy bases, territories, mining points.

The player needs to extract resources to expand the military base and modernize buildings and transport. Scattered throughout the map are enemy mining points that replenish the balance of resources and materials.

The main combat unit of the game "War of Technique" is military equipment of various types, each of which has its own advantages and disadvantages. Military equipment is also subject to improvement of its properties due to a certain amount of resources.

During the execution of the work, all the main and necessary stages of the software life cycle were passed, the design and development methodology was chosen, the connections between the system components were described using UML diagrams, and the results of one sprint were tested according to the Scrum methodology.

ЗМІСТ

ВСТУП……7
1. АНАЛІЗ ВИМОГ ТА ПРЕДМЕТНОЇ ОБЛАСТІ……..9
1.1. Словник предметної області……....9
1.2. Аналіз вимог до системи……10
1.2.1. Постановка задачі……10
1.2.2. Функціональні вимоги……...…11
1.2.3. Нефункціональні вимоги……...…14
1.2.4. Вимоги до захисту, доступу, інтерфейсу системи. Вимоги до програмного забезпечення……..……..14
1.3. Технології системи……...…..15
2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ……...17
2.1. Обраний тип архітектури……...…17
2.2. Проектування відношень між акторами і прецедентами……....17
2.3. Аналіз взаємодії об’єктів за часом……21
2.4. Визначення класів системи……..….24
3. РОЗРОБКА ТА ТЕСТУВАННЯ ГРИ……...35
3.1. Обрана методологія розробки……...35
3.2. Проведення спринтів……..…36
4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ……....43
4.1. Надзвичайні ситуації: визначення причини, класифікація…….43
4.2. Загальні вимоги безпеки з охорони праці для користувачів ПК……..….44
ВИСНОВКИ…….…..48
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ……...…..49
ДОДАТОК А – Лістинги коду класів системи……..…51
ДОДАТОК Б – Діаграми UML……..…66
ДОДАТОК В – Знімки екрану при процесі гри…….…70
ДОДАТОК Г – Диск із кваліфікаційною роботою бакалавра……...72

ВСТУП

Темою кваліфікаційної роботи бакалавра було обрано проектування та розробку комп’ютерної 3D-гри жанру стратегія на основі ігрового двигуна Unity, оскільки це є перспективним варіантом для стартапу у виконанні одного, або декількох розробників у відносно короткі терміни та при затраті мінімальної кількості ресурсів.

Ігровим двигуном що слугує основою для даної гри є Unity - потужний і популярний движок для розробки ігор, який надає розробникам інструменти та середовище для створення інтерактивних 2D і 3D додатків, є легким в освоєнні, має велику кількість навчальних ресурсів, що сприяють поліпшенню навичок роботи із ним та головне – є безкоштовним, при умові що прибуток із гри створеної на його основі не буде перевищувати вказану межу. Крім того, Unity є кросплатформенним, з його допомогою, з легкістю, у короткі терміни, можна буде розробити таку ж відеогру для платформи Android чи IOS застосувавши мінімальні зміни до попереднього проекту.

Результатом даної дипломної роботи є бета-версія гри “War of Technique” – стратегії, суть котрої полягає у створенні своєї військової бази, її розширення, покращення, захист та захоплення ворожих територій, баз та видобувних точок за допомогою військової техніки різних видів та типів.

У ході виконання кваліфікаційної роботи бакалавра було пройдено крізь всі етапи життєвого циклу програмного забезпечення, здійснено їх детальний опис та документацію.

Виконана кваліфікаційна робота бакалавра передбачає собою детальний опис аналіз та документацію усіх етапів життєвого циклу програмного забезпечення, а саме:

- аналіз усіх визначених типів вимог до системи;
- вибір найбільш відповідної архітекрути програмного забезпечення для коректного та комфортного проектування гри;

- створення UML-діаграм варіантів використання, послідовності, діаграми класів, для графічного відображення зв’язків між об’єктами та компонентами системи;
- опис та документація усіх класів системи, що є її компонентами;
- визначення найбільш вигідної, зрозумілої та комфортної методології розробки програмного забезпечення для створення гри соло-розробником;
- опис спринта на певній стадії розробки програмного забезпечення, проведення тестування функціоналу розробленого під час його проведення.

1. АНАЛІЗ ВИМОГ ТА ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Словник предметної області

Unity - це кросплатформений ігровий двигун та розробницька платформа, яка використовується для створення ігор, симуляцій, віртуальної реальності, анімацій та інших інтерактивних візуальних проектів.

Ігровий двигун (Game Engine) - спеціалізоване програмне забезпечення для створення та розробки відеоігор. Включає в себе інструменти для роботи з графікою, фізикою, звуком, штучним інтелектом та іншими аспектами гри.

Сцена (Scene) - віртуальне середовище в ігровому двигуні, де відбувається гра. Включає об'єкти, персонажів, світло, камери та інші елементи.

Ассет (Asset) - графічні, звукові, або інші ресурси, що використовуються в грі. Може включати моделі персонажів, текстури, звукові ефекти тощо.

Скрипт (Script) - код, написаний мовою програмування C#, який визначає поведінку об'єктів в грі. Використовується для реалізації логіки гри та взаємодії об'єктів.

Шкода (Damage) – кількість одиниць здоров’я котрі певна дія чи об’єкт може нанести іншому об’єктові.

Фрейм (Frame) – кадр у графічному відображенні гри. Метод Update викликається кожен фрейм для обновлення логіки гри.

Компонент (Component) - базова одиницю функціоналу в системі гри. Всі об'єкти в Unity є сукупністю компонентів, які визначають їхню поведінку, властивості та взаємодію з навколишнім середовищем.

Куротина (Coroutine) в Unity - це концепція, що дозволяє запускати асинхронні процеси, які виконуються паралельно з основним потоком виконання коду. Вони часто використовуються для виконання дій, які тривають певний час, але не повинні заблокувати виконання основного коду гри.

Колайдер (Collider) - компонент об’єкта що відображає його тверді фізичні межі та забезпечує виявлення колізій (стикань) між ними.

Штучний Інтелект (AI - Artificial Intelligence) - система, яка реалізує імітацію розуму та прийняття рішень об'єктами або персонажами в грі. У даному проекті штучний інтелект буде використовуватися для пошуку оптимального маршруту для транспорту що забезпечить обминання фізичних об’єктів що розміщені на сцені

Спавн (Spawn) - процес створення нового об'єкта у грі, зазвичай визначеного певними умовами чи подіями.

Префаб (Prefab) - передвизначений об'єкт , який можна використовувати для створення інших об'єктів в грі. Префаб є шаблоном, який містить в собі компоненти, моделі, колайдери, скрипти та інші ресурси, що визначають конкретний об'єкт чи елемент гри.

Гравець (Player) – кінцевий користувач гри, той хто керуватиме ігровим процесом у готовому продукті.

1.2 Аналіз вимог до системи

Можна визначити такі основні типи вимог до при розробці даного проекту:

- функціональні вимоги;
- нефункціональні вимоги;
- вимоги до захисту та доступу;
- вимоги до інтерфейсу системи;
- вимоги до програмного забезпечення.

1.2.1 Постановка задачі

Програмний продукт повинен представляти собою бета-версію комп’ютерної 3D гри «War of Technique» розробленої для платформи Windows.

1.2.2 Функціональні вимоги

У даній версії гри будуть реалізовані наступні основні механіки:

а) активація техніки для її подальшого керування:

1) обрані одиниці техніки додаватимуться у масив об’єктів для їх подальшої експлуатації.

б) пересування техніки по карті:

1) рух активної техніки по карті у вказані мишкою точки обминаючи розміщені на сцені фізичні об’єкти, що було реалізовано за допомогою розширення що використовує штучний інтелект для пошуку оптимального шляху;
2) можливість вказування для техніки декількох точок призначення, котрих транспорт повинен досягнути у певному порядку.

в) атака технікою ворожих будівель та техніки:

1) запуск процесу знищення ворожого об’єкту що завершиться лише коли його здоров’я зрівняється з нулем, або він вийде з зони досягнення;
2) можливість запуску знищення із подальшим переслідуванням ворожого об’єкту.

г) опрацювання здоров’я об’єктів:

1) різні ігрові об’єкти такі як будівлі та техніка мають таку властивість як здоров’я що зменшується при нанесенні їм шкоди. Коли здоров’я об’єкту дорівнює нулю він автоматично видаляється із сцени.

д) автоматичний вибір пріоритетного противника:

1) уся техніка що заходить у певний радіус навколо певного транспорту поміщається у спеціальний масив і буде знищуватися по мірі надходження якщо даний транспорт не зайнятий атакою чогось іншого.

е) розміщення будівель навколо штабу:

1) розміщення на сцені арсеналів що здійснюватимуть спавн техніки для її подальшої експлуатації;
2) розміщення турелей що будуть здійснювати стрільбу по ворожому транспорті поблизу.

є) користування різними бойовими одиницями:

1) можливість експлуатації різних бойових одиниць, кожна з яких має свої переваги та недоліки і є доцільною у використанні у певних ситуаціях. У будуть реалізовані такі одиниці техніки:
   – танк Т90 – бойова одиниця із великою шкодою та відносно невеликою швидкістю пересування та стрільби;
   – бронемашина – бойова одиниця із великою швидкістю пересування та стрільби, але низьким уроном та кількістю одиниць здоров’я.

ж) робота з матеріалами:

1) використання матеріалів у цілях побудови нових будівель, захисних споруд, чи створення нових одиниць транспорту;
2) захоплення ворожих видобувних точок для отримання корисних копалин та грошей. Повинні бути реалізовані такі типи видобувних точок:
   – помпа – точка для видобутку нафти, що буде поповнювати баланс грошей:
   – шахта – видобувна точка для видобутку заліза;
   – лісопильня – видобувна точка для поповнення балансу деревини.

з) реалізація «туману війни»

1) створення, так званого, «туману війни» - спеціальної механіки котра означає, що гравець не має повного огляду всієї карти гри або рухів противника. Гравець може бачити тільки область навколо своїх військ та область навколо його споруд. Територія поза цим областями знаходиться в «тумані війни» і не видима для гравця.

При створенні нових версій продукту планується ввести велику кількість

нововведень та покращень, а саме:

а) збільшення розміру мапи:

1) збільшення масштабів ігрової території;
2) розміщення нових ворожих баз;
3) створення нових мап, наприклад: пустеля, засніжена долина і тд.

б) впровадження нових типів будівель;

в) можливість експлуатації нових типів транспорту:

1) гелікоптер – повітряна бойова одиниця із середніми характеристиками;
2) бронетанк – бойова одиниця із великими показниками здоров’я, відносно високою шкодою, великим інтервалом між пострілами та дуже низькою швидкістю руху;
3) зенітна установка - бойова одиниця із відсутністю можливості пересування, що має великий радіус ураження але неможливість здійснювати постріли на ближніх дистанціях.

г) забезпечення музичного та звукового супроводу, адже на момент бета-версію у грі відсутні будь які звуки чи музика;

д) впровадження анімацій руйнування будівель, транспорту;

е) впровадження нових можливостей інтерфейсу гри:

1) початок нової гри;
2) збереження гри;
3) налаштування графічної складової.

є) здійснення оптимізації гри для зменшення навантаження на систему, зниження кількості обчислювальних ресурсів.

ж) впровадження модифікацій:

1) можливість отримання нового типу ресурсів – спеціальних матеріалів;
2) модифікація характеристик певних типів техніки;
3) модифікація властивостей будівель.

1.2.3 Нефункціональні вимоги

Гра жанру стратегії в реальному часі «War of Technique» повинна відповідати високим стандартам якості та потребам широкого кола гравців. В кінцевому результаті бета-версії продукту повинні бути реалізовані такі нефункціональні вимоги:

- продуктивність - гра повинна запускатися та працювати плавно на пристроях середньої продуктивності;
- оптимізованість – гра вміщатиме в себе лише ассети необхідні для виконання гри. Ресурси в них, котрі не використовуватимуть, будуть видалені із проекту;
- масштабованість - гра повинна бути масштабованою і здатною працювати на різних розширеннях екрану та пристроях з різними характеристиками;
- зручність користування – керування грою повинно бути інтуїтивно зрозумілим та зручним для гравців різного рівня досвіду;
- час завантаження - час завантаження гри повинен бути мінімальним, щоб гравці швидко потрапляли в гру після запуску;
- стабільність - гра повинна бути стабільною і надійною, без системних збоїв чи вилетів;

1.2.4 Вимоги до захисту, доступу, інтерфейсу системи. Вимоги до програмного забезпечення

Враховуючи всі раніше наведені особливості даної гри можна сформувати наступні вимоги для коректної роботи гри:

- доступ до додатку буде здійснюватися після його попереднього встановлення на комп’ютерний пристрій операційної системи Windows 7, або вище;
- оскільки гра ні в якому плані не взаємодіє із інтернет мережею вона не потребує особливого забезпечення кібербезпеки.

Перелік вимог до інтерфейсу даного додатку:

- інтерфейс повинен коректно відображатися на пристроях із різним розширенням екрану;
- інтерфейс повинен мати мінімальну кількість кнопок та вкладок для повноцінного та зручного користування грою;
- усі кнопки повинні чітко відображатись, не мати різких та неприємних кольорів.

1.3 Технології системи

Розробка даної 3D гри «War of Technique» жанру стратегія відбуватиметься за допомогою ігрового двигуна Unity [1].

Unity - це інтегроване середовище розробки (IDE) та ігровий двигун, який використовується для створення різноманітних ігор, інтерактивних додатків та віртуальної реальності. Розроблений компанією Unity Technologies, цей інструмент надає розробникам широкі можливості для втілення їхніх творчих ідей у віртуальних просторах.

Unity використовується як незалежними розробниками, так і великими студіями для створення ігор на різних платформах. Його популярність визначається не лише зручним інструментарієм, але й активною спільнотою розробників, яка надає підтримку та обмін досвідом.

Ігровим двигуном для розробки було обрано саме Unity через великий ряд переваг. Ось кілька основних причин, чому Unity став відмінним вибором [2]:

- доступність: Ігровий двигун Unity знаходиться у відкритому доступі та доступний для завантаження та експлуатації будь ким у власних цілях. Крім того комісії щодо прибутку від проекту створеного за допомогою даного двигуна будуть враховуватися лише коли прибуток за продукт перевищить вказану суму що дорівнює декільком мільйонам в рік.
- кросплатформенність: Unity дозволяє створювати ігри для різних платформ, таких як Windows, macOS, Android, iOS, консолі Xbox і PlayStation. Це означає, що ви можете легко розповсюджувати свою гру для різних аудиторій.
- велика спільнота та документація: Unity має велику і активну спільноту користувачів. Це означає, що ви можете легше знаходити допомогу, ресурси та поради в Інтернеті. Документація Unity також є дуже детальною та доступною.
- легкість вивчення: Unity ставиться як інтуїтивно зрозумілий ігровий двигун, що спрощує процес вивчення для новачків. Є багато онлайн-ресурсів та навчальних матеріалів, які допомагають в освоєнні платформи.
- велика кількість готових ресурсів: Unity має власний офіційний сайт Asset Store [3], де є можливість пошуку готових ресурсів, таких як 3D-моделі, текстури, звуки та інше. Це може значно прискорити процес розробки, особливо при обмеженому бюджеті та часі.
- можливості розширення: Unity підтримує велику кількість розширень і плагінів, що дозволяє розширювати функціональність двигуна за необхідності.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Обраний тип архітектури

Для розробки 3D-гри жанру стратегія на основі ігрового двигуна Unity було обрано компонентний тип архітектури.

Компонентна архітектура - це підхід до проектування та розробки систем, який розбиває їх на окремі, самостійні компоненти з чітко визначеними інтерфейсами. Ці компоненти мають власні методи, події та властивості, що робить їх модульними та легко керованими. Компонентна архітектура пропонує вищий рівень абстракції, ніж об'єктно-орієнтована розробка, не зосереджуючись на деталях протоколів зв'язку чи загального стану системи.

Компонентний тип архітектури ідеально підходить для створення гри на ігровому двигуні Unity, адже при розробці з його допомогою все базується на концепції компонентів, котрі складають певний об’єкт, відображаючи його зовнішній вигляд, функціонал, фізичні властивості, особливості та багато іншого. Скрипти (класи), котрі реалізовують функціонал, в свою чергу теж є компонентами об’єктів, котрі в кінцевому результаті і формують саму гру [6].

2.2 Проектування відношень між акторами і прецедентами

Провівши аналіз необхідного функціоналу гри можна дійти до висновку, що грі даної концепції можна визначити лише одного актора – користувача (або ж гравця), оскільки прототип не матиме взаємодії ні з інтернет мережею, ні з базами даних, ні з іншими користувачами.

Гравець (користувач) – людина що гратиме у гру і використовуватиме увесь функціонал розроблений у грі. Він створюватиме свою військову базу, керуватиме технікою, вестиме бойові дії із ворожим транспортом і так далі:

Можна навести такі приклади дій що користувач здійснюватиме граючи у гру «War of Technique»:

- користувач обирає певну техніку та робить її активною виділивши її спеціальним полем котре з’являється після натискання лівої кнопки миші (далі ЛКМ) та її пересування. Після цього над кожною одиницею транспорту з’являтиметься смуга здоров’я що відображатиме скільки здоров’я залишилося у певної техніки. В такому випадку вся обрана техніка поміщається у масив котрим надалі користувач буде користуватися;
- користувач очищує масив вибраних об’єктів натиснувши ЛКМ на пустому місці на сцені. В такому випадку у всієї раніше обраної техніки зникають смуги здоров’я та вони робляться неактивними та не реагують на жодні команди котрі буде надавати гравець своєму транспорту, допоки вони не будуть повторно активовані;
- користувач наказує техніці пересуватися у певне місце на карті натиснувши правою клавішею миші (далі ПКМ) на потрібному місці. Тоді вся активна техніка розпочне рух до цієї точки обминаючи перешкоди доки не досягне вказаного місця, або не отримає інші вказівки. У ході руху транспорт буде освітлювати територію навколо себе від «туману війни», у випадку потрапляння ворожого транспорту у зону досягання певної одиниці транспорту гравця вони відкриють вогонь один по одному;
- користувач наказує техніці знищити ворожу техніку натиснувши на неї ПКМ. В такому випадку активується куротина що наближуватиме усю техніку що знаходиться у масиві до цілі (якщо вона надто далеко) та запускатиме процес стрільби по цілі допоки значення його здоров’я не буде дорівнювати нулю. В цей час ворожа техніка аналогічно розпочне стрільбу по техніці гравця що наблизилась на достатню для неї відстань;

- користувач наказує техніці атакувати будівлі натиснувши на неї ПКМ. Відбуватиметься аналогічний процес тим, що відбувався при атаці ворожої техніки, тільки без атаки у відповідь;
- користувач будує нові будівлі навколо штабу. Натиснувши на штаб ПКМ над останнім з’явиться вікно, де користувач обере функцію будування. Після цього з’явиться меню вибору бажаного спорудження для розміщення та відображення території дозволеної для будування спорудження, гравець робить вибір та розміщує споруду;
- техніка що знаходиться у стані бездіяльності може самостійно розпочати стрільбу по ворожій техніці що заходить у певний радіус навколо неї. Він буде здійснювати знищення допоки користувач не дасть їй нові вказівки або цей ворожий транспорт не вийде із її зони ураження чи не знищиться;
- користувач виходить з гри. Натиснувши на клавішу “ESC” відкриється головне меню де користувач обирає пункт “Вийти з гри”, після чого завершується виконання програми;
- користувач захоплює ворожу видобувну точку. Маючи активовані одиниці транспорту гравець натискає ПКМ по ворожій видобувній точці наказуючи атакувати її. Після здійснення успішної атаки значення здоров’я ворожої видобувної точки опускається до нуля, після чого на його місці з’являється точка такого ж типу що належить гравцеві та приносить йому прибуток.

Для легшого сприйняття взаємодії між гравцем та можливими сценаріями використання гри було створено діаграму прецедентів. На представленому зображенні подана UML-діаграма, яка ілюструє взаємозв'язки між учасниками та сценаріями використання в системі.

Діаграма прецедентів UML - це візуальне представлення взаємодії користувачів (акторів) з системою. Вона відображає сценарії використання (прецеденти), межі системи, зв'язки між акторами та прецедентами, а також їхні взаємозв'язки. Діаграми прецедентів використовуються для моделювання та

документування функціональних можливостей системи [4].

В основі діаграми прецедентів лежить ідея моделювання системи як набору акторів (користувачів) та їх взаємодії з нею за допомогою чітко визначених сценаріїв використання. Ці сценарії, або "варіанти використання", описують послуги, які система надає кожному конкретному актору. Іншими словами, кожен сценарій чітко визначає набір дій, які система виконує під час взаємодії з певним користувачем.

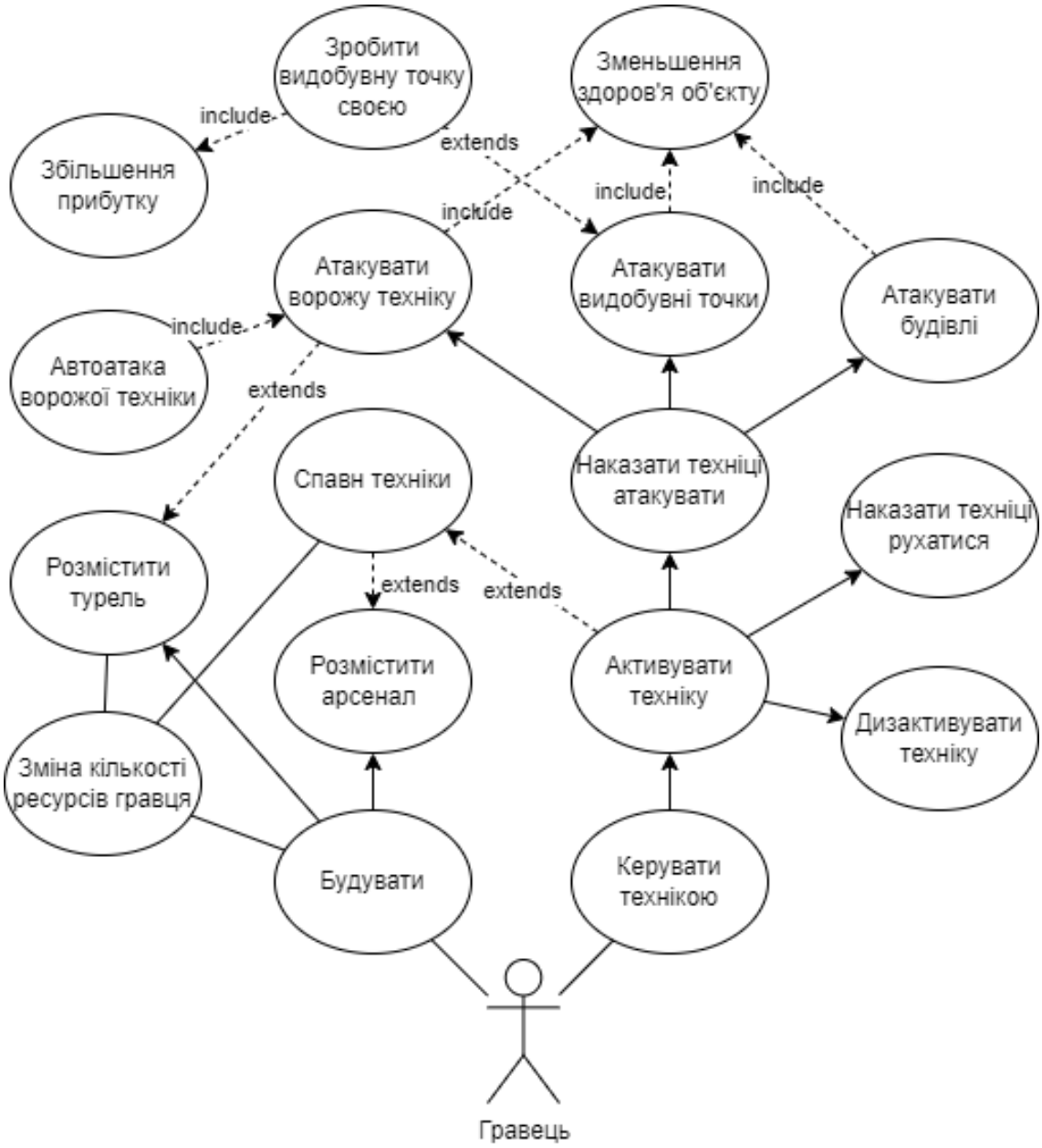


Рисунок 2.1 – Діаграма варіантів використання

2.3 Аналіз взаємодії об’єктів за часом

Для відображення взаємодії об'єктів з урахуванням впорядкованості за часом було створено діаграми послідовностей що відображатимуть два типові зразки користування грою «War of Technique».

Діаграма послідовності (Sequence diagram) - це вид UML діаграм, що візуалізує взаємодію між об'єктами системи в часовому порядку. Вона чітко показує, які об'єкти беруть участь у взаємодії, які повідомлення вони надсилають один одному та в якій послідовності це відбувається [4].

Діаграми послідовностей доповнять діаграми прецедентів, надаючи детальний опис логіки сценаріїв використання. Ці діаграми стануть цінним інструментом для документування проекту, особливо в частині сценаріїв використання. На них будуть представлені об'єкти, які взаємодіють в рамках конкретного сценарію, повідомлення, що надсилаються між ними, та результати, пов'язані з цими повідомленнями.

Розглянемо два типові випадки користування грою:

1. Атака ворожої техніки:
   - гравець активує певні одиниці техніки виділяючи їх полем що створюється та керується натиснувши ЛКМ та перетягнувши мишку, після чого вони стають придатними для їх подальшої експлуатації;
   - користувач обирає ворожу ціль для атаки натиснувши на неї ПКМ, після чого у активної техніки запускається процес знищення ворожого транспорту;
   - якщо відстань між атакуючим та жертвою недостатня для здійснення стрільби, одиниці техніки гравця почнуть наближатися до ворожих доки дистанція не буде задовільною;
   - коли відстань є достатньою активні одиниці транспорту розпочинають стрільбу по ворожих цілях і здійснюватимуть це доки значення їх здоров’я ну буде дорівнювати нулю.

2. Розміщення арсеналу:
   - гравець натискає на штаб, щоб відкрити опції військової бази що побудована навколо даного штабу та обирає функцію будівництва;
   - з’являється меню вибору споруд, відображається територія у межах якої можна розміщувати будівлі, після чого гравець обирає бажаний вид споруди, у даному випадку – арсенал;
   - тоді, будівля буде слідкувати за положенням миші користувача для пошуку вигідного місця. Користувач матиме змогу крутити будівлю натискаючи клавіші “Z” та “X”;
   - користувач натискає на ЛКМ на дозволеному для будівництва місці, вільному від інших об’єктів після чого будівля розміщується у цій точці та перестає слідкувати за мишею гравця;
   - коли арсенал встановлений, через певні проміжки часу він буде створювати біля себе нові одиниці техніки, придатні для експлуатації користувачем;
   - гравець активує новий транспорт для подальшого користування.

Розглянемо діаграму послідовностей першого варіанту користування грою що зображена на наступному рисунку.

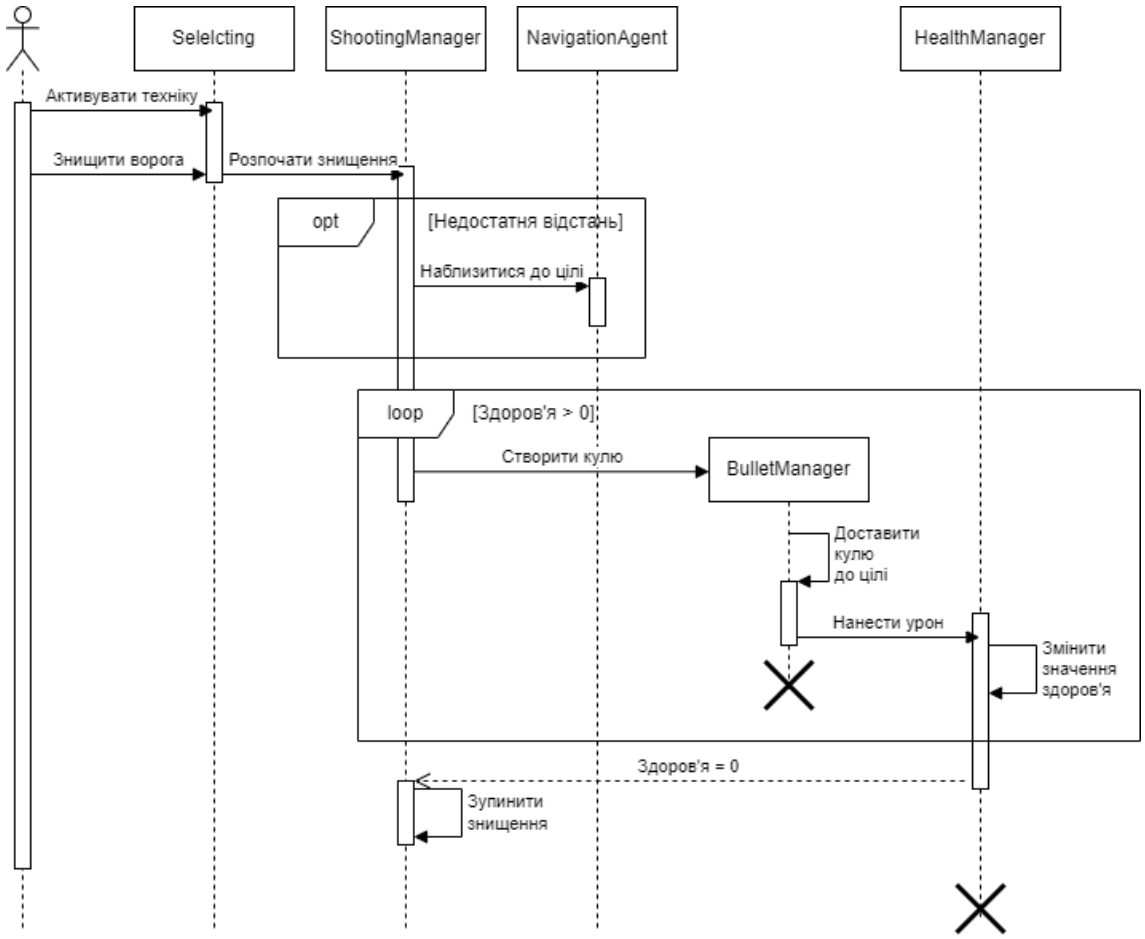


Рисунок 2.2 – Діаграма послідовностей першого випадку

Тепер проведемо огляд діаграми послідовностей, котра зображена на рисунку 2.3, що відображає поведінку гри у другому випадку користування нею.

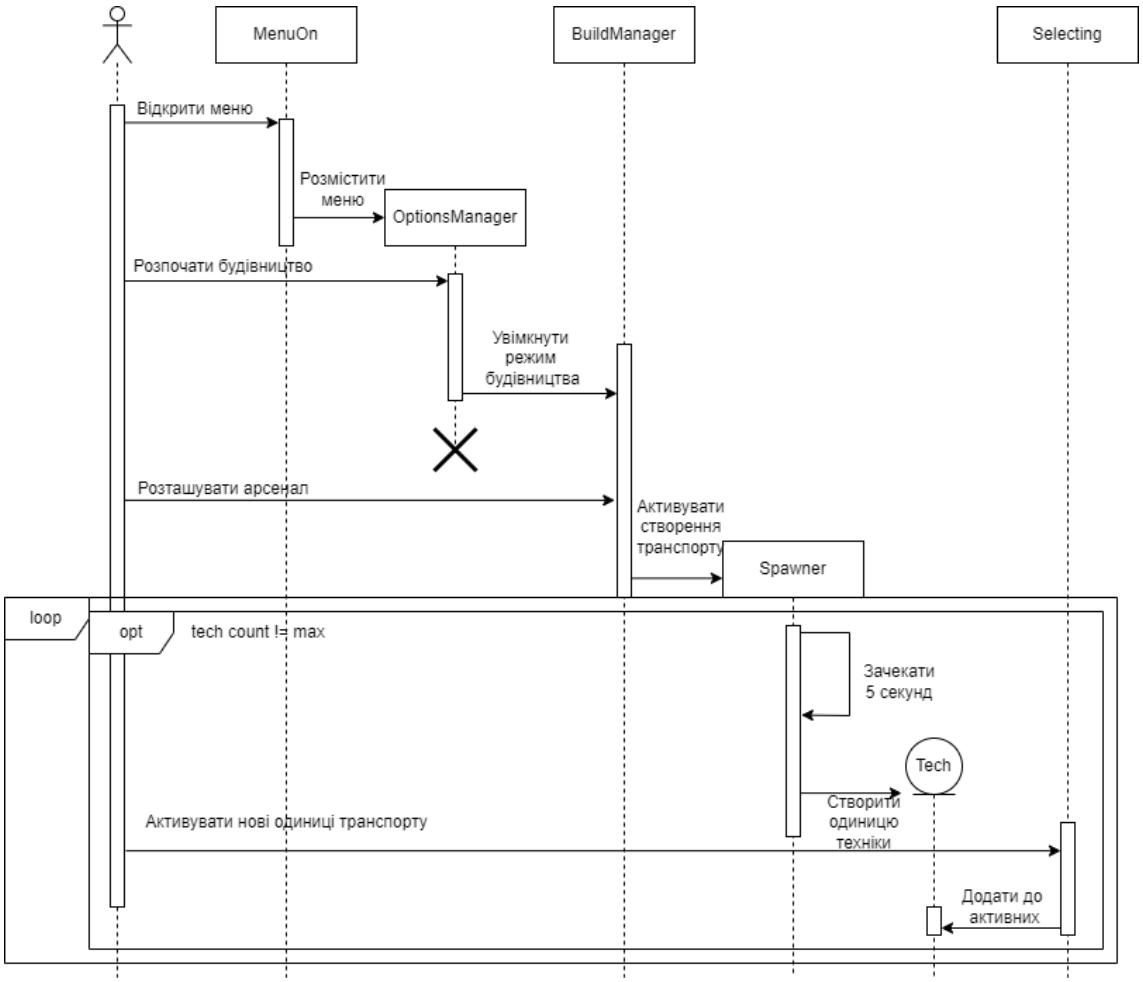


Рисунок 2.3 – Діаграма послідовностей другого випадку

2.4 Визначення класів системи

Після завершення попереднього аналізу системи та ідентифікації всіх потрібних класів і сутностей, які необхідні для повноцінної роботи додатку, можна приступити до розробки архітектури системи.

Було створено діаграми класів для візуального представлення структури системи за допомогою зображення класів, їх властивостей, методів та взаємозв'язків між ними [5].

Діаграми класів є важливим інструментом у процесі проектування програмного забезпечення, створення яких забезпечило досягнення наступних цілей:

- візуалізація структури - діаграми класів надали чітке та візуальне подання структури системи;
- аналіз взаємозв'язків - діаграми класів вказали на зв'язки між класами, що дозволило розуміти, як дані та функціонал системи взаємодіють та інтегруються між собою;
- планування розробки - за допомогою діаграм класів було створено конкретний план розробки, визначаючи, які класи повинні бути реалізовані та як вони будуть взаємодіяти;
- аналіз та рефакторинг - діаграми класів стали корисним інструментом для аналізу та подальшого вдосконалення коду. Вони дозволили виявити можливості для рефакторингу та оптимізації системи.

У випадку розробки гри за допомогою ігрового двигуна Unity дані класи доцільно називати «скриптами» - компонентами певних об’єктів, що реалізують їх поведінку та функціонал.

На одному об’єкті може бути декілька скриптів. Так, наприклад, у об’єкта що відображає певну бойову одиницю гравця є такі класи:

- Shooting – скрипт що реалізовує стрільбу техніки та описує бойові характеристики транспорту;
- HealthManager – скрипт що проводить моніторинг рівня здоров’я об’єкту;
- Triggering – скрипт що перевіряє чи не знаходиться у радіусі стрільби ворожа техніка;
- Moving – скрипт що відповідає за організацію пересування одиниці транспорту, розставлення пріоритетів у русі.

На рисунку 2.4 зображена діаграма класів що реалізують функціонал гри на даному етапі бета-версії:

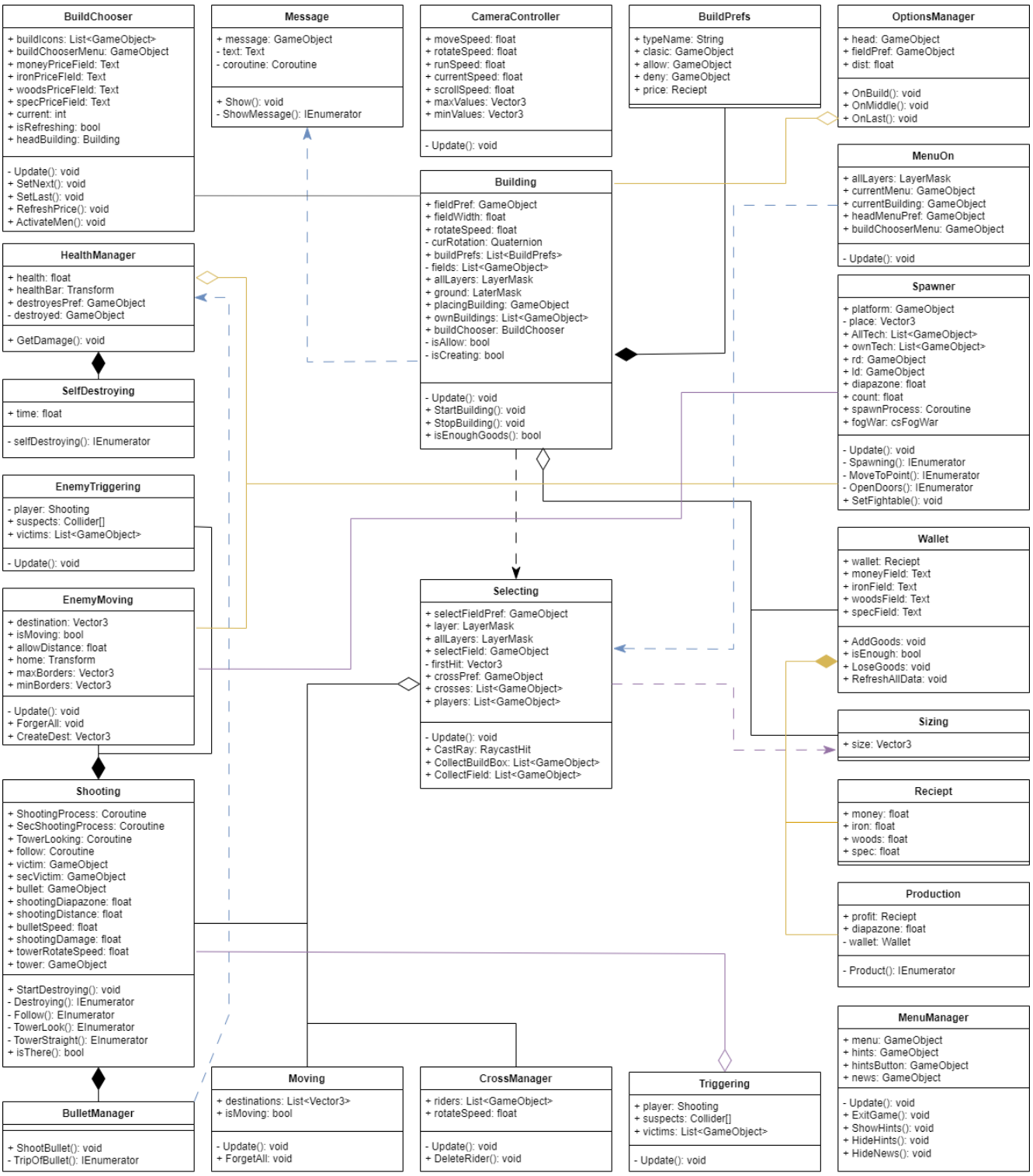


Рисунок 2.4 – Діаграма класів

Здійснимо детальний огляд найскладніших та наймасивніших класів, що відображені на даній діаграмі та здійснимо опис кожного із них:

1. Selecting – клас, котрий здійснює експлуатацію транспорту користувачем, реалізовує активацію транспорту, дає йому команди для

руху, атаки, тощо:

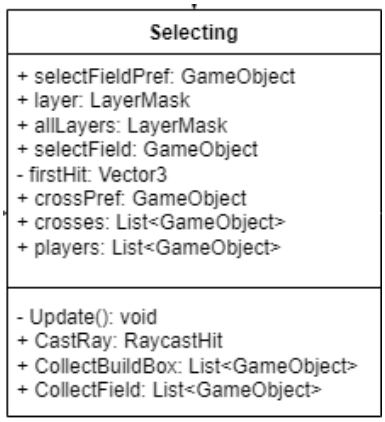


Рисунок 2.5 – Клас Selecting

- selectFieldPref – префаб поля, котрим здійснюється активація (виділення) одиниць транспорту;
- layer – значення масок шарів, взаємодія з якими відслідковується;
- allLayers – значення масок всіх шарів;
- selectField – поле для виділення створене з префабу;
- firstHit – кординати точки з якої користувач розпочав створення поля для виділення;
- crossPref – префаб хреста що розташується у точці до котрої користувач наказав рухатися техніці;
- crosses – список хрестів, що є зараз розташованими на сцені;
- players – список транспорту, на даний момент є обрані (активовані) та готові до експлуатації;
- Update() – метод, що виконується кожен фрейм та відслідковує взаємодію користувача із транспортом
- CastRay() – статичний метод, що здійснює запуск променю у точку, на котру у даний момент вказує курсор миші, та збір інформації про неї;
- CollectBuildBox() – здійснює пошук об’єктів всередині будівлі використовуючи клас Sizing, що є компонентом будівлі;

– CollectField() – здійснює пошук об’єктів всередині поля для виділення.

2. Shooting – клас що реалізовує ведення бою кожною одиницею транспорту, дає можливість змінити його бойові властивості: швидкість польоту кулі, шкода та швидкість стрільби:

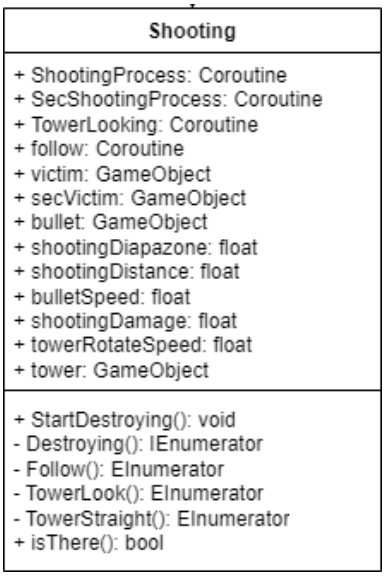


Рисунок 2.5 – Клас Building

– ShootingProcess – активна куротина що здійснює процес ведення бою;
– SecShootingProcess – активна куротина що здійснює другорядний процес ведення бою;
– TowerLooking – активна куротина що здійснює процес слідкування дула одиниці транспорту за ворожим об’єктом;
– follow – активна куротина що здійснює процес руху за ворогом;
– victim – основна ворожа ціль котру необхідно знищити;
– secVictim - другорядна ворожа ціль котру необхідно знищити;
– bullet – префаб кулі даного типу транспорту;
– shootingDiapazone – значення відстані на котрі даний тип транспорту може вести бій;

- bulletSpeed – значення швидкості польоту кулі;
- shootingDamage – значення шкоди, котру даний тип транспорту може нанести ворогові;
- towerRotateSpeed – значення швидкості обертання вежі із дулом;
- tower – вежа із дулом котра слідкує за ворогом;
- StartDestroying() – функція що запускає процес знищення ворога;
- Destroying() – куротина, що здійснює процес знищення ворога;
- Follow() – куротина, що здійснює процес переслідування ворога;
- TowerLook() – куротина що здійснює процес слідкування дула за ворожим об’єктом;
- TowerStraight() – куротина, що здійснює процес повернення дула у початковий стан;
- isThere() – функція що перевіряє чи дана одиниця транспорту вже досягнула вказаного місця призначення.

3. Building – клас що реалізовує будування нових споруд для розширення військової бази, відкриває режим будування, відповідає за списання коштів за будівлі:

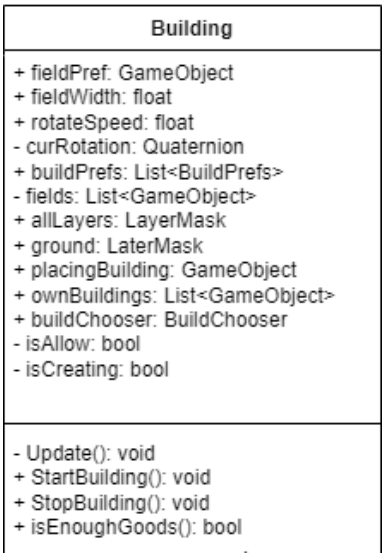


Рисунок 2.6 – Клас Building

- fieldPref – префаб поля для розміщення (поля навколо усіх будівель

бази);

- fieldWidth – значення розміру полей для розміщення;
- rotateSpeed – значення швидкості обертання макетів будівель;
- curRotation – останнє зафіксоване значення обертання макету;
- buildPrefs – список префабів будівель та їх макетів;
- fields – список полів для розміщення, котрі вже є на сцені;
- allLayers – значення усіх масок шарів;
- ground – значення маски шару землі, на котрій будують будівлі;
- placingBuilding – макет, котрий на даний момент розміщується і є на сцені;
- ownBuildings – список усіх будівель що є частиною даної військової бази;
- buildChooser – компонент, що є класом BuildChooser, котрий відповідає за вибір типу будівлі що буде розташовуватися;
- isAllow – значення що відображає чи може будівля розташуватися на місці макету;
- isCreating – значення що відображає, чи відбувається процес розташування будівлі на даний момент;
- Update() – влаштований метод, що виконується кожен фрейм. Відслідковує дії користувача.
- StartBuilding() – метод, що активує режим будівництва: розташовує на сцені поля для розміщення, активує інтерфейс для вибору будівель.
- StopBuilding() – метод, котрий закриває режим будівництва;
- isEnoughGoods() – метод, що перевіряє чи достатньо матеріалів для розміщення будівлі даного типу.

4. Spawner – клас що здійснює створення нових одиниць транспорту арсеналами, забезпечує їх боєздатність, додає їх до списку об’єктів що можуть розсіювати туман:

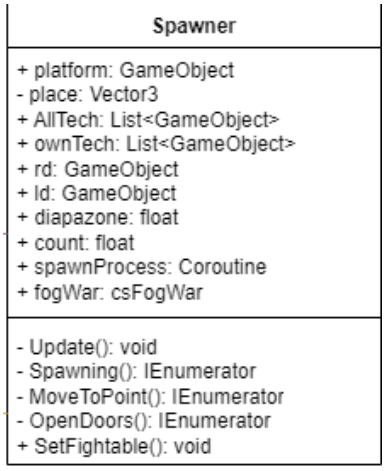


Рисунок 2.7 – Клас Spawner

- platform – платформа перед арсеналом, на котру виїжджатиме техніка;
- place – випадкове місце для виїзду із арсеналу знайдене на платформі;
- AllTech – список префабів усієї техніки що може виїхати із арсеналу;
- ownTech – список техніки котра уже виїхала із арсеналу;
- rd – права частина воріт арсеналу;
- ld – ліва частина воріт арсеналу;
- diapazone – період через який може створитися та виїхати нова одиниця транспорту;
- count – значення кількості одиниць транспорту із арсеналу, що одночасно можуть бути на сцені;
- spawnProcess – значення активною куротини процесу створення транспорту;
- fogWar – туман війни;
- Update() - влаштований метод, що виконується кожен фрейм. Відслідковує кількість транспорту із даного амбару що є на сцені;
- Spawning() – куротина що реалізовує процес створення техніки;
- MoveToPoint() – куротина, що здійснює процес виїзду одиниці транспорту із арсеналу;

- OpenDoors() – куротина, що здійснює процес відкривання дверей арсеналу;
- SetFightable() – функція, що вмикає, або вимикає значення боєздатності одиниці транспорту. Уся техніка повинна бути небоєздатною, допоки повністю не виїде із арсеналу.

Здійснимо короткий опис решти класів, котрі також є невід'ємною частиною реалізації функціоналу гри:

1. BuildChooser – клас, котрий надає можливість обирання типу споруди, котру користувач бажає розмістити на сцені. Даний клас реалізовує інтерфейс що з'являється у режимі будування та відображає ціну будівлі обраного типу.
2. HealthManager – клас, що здійснює маніпуляції над значенням здоров'я певного об'єкту. Крім того він реалізовує унікальну поведінку дуяких об'єктів, коли значення їх здоров'я опускається до нуля.
3. SelfDesroying – клас, що є компонентом зруйнованих об'єктів та відповідає за їх зникнення із сцени, через певні проміжки часу.
4. EnemyTriggering – клас. що реалізовує функціонал ворожого транспорту. Відповідає за автоматичний пошук об'єктів гравця у певному радіусі навколо себе та віддає команди щодо їх знищення.
5. EnemyMoving – клас, що реалізовує пересування ворожого транспорту. Шукає для них певні точки на мапі, котрі знаходяться недалеко від арсеналу котрий їх випустив.
6. BulletManager – клас, що є компонентом кулі, котра створюється класом Shooting при здійсненні пострілу. Відповідає за доставку кулі до точки призначення та надання команди про зменшення одиниць здоров'я.
7. Moving – клас, що контролює рух одиниць транспорту гравця. Збирає значення усіх координат до котрих повинна дістатись одиниця техніки, та надає команди щодо руху до них.
8. CrossManager – клас що реалізовує функціонал хреста що відображає точку призначення для техніки. Здійснює моніторинг об'єктів, що

рухаються до нього, видаляє хрест, у випадку їх відсутності. Здійснює анімацію обертання хреста.

9. Triggering - клас. що реалізовує автоматичний пошук ворожих об’єктів навколо одиниць транспорту гравця у певному радіусі навколо них та віддає команди щодо їх знищення.
10. MenuManager – клас, що реалізовує функціонал інтерфейсу гри. Відповідає за вихід із гри, користування підказками, відображення інформації про оновлення гри при вході.
11. Production – клас що є компонентом видобувних точок, котрі поповнюють баланс ресурсів гравця на певну кількість, за певний період.
12. Reciept – клас, що відображає набір значень кожного типу ресурсів. Зазвичай використовується для відображення балансу користувача та прибутку від певної видобувної точки.
13. Sizing – клас, що відображає розмір певного об’єкта по всіх трьох координатах. Використовується для розрахунку розмірності будівель та, контрольованого за розмірами, пошуку об’єктів всередині них.
14. Wallet – клас, що реалізовує «гаманець» користувача. Відповідає за відображення кількості кожного типу ресурсів гравця та зміни їх кількості.
15. MenuOn – клас, що відповідає за відкриття опцій військової бази при натисканні на штаб та відображення інтерфейсу над ним.
16. OptionsManager – клас, що реалізовує функціонал опцій військової бази. На даний момент доступною є лише опція будівництва, а в подальшому планується створення функцій покращення транспорту, запуску радарів і тд.
17. BuildPrefs – клас, що являє собою набір моделей будівлі, що має розташовуватися, а саме:
    - класична модель – модель будівлі, що в кінцевому результаті розміститься на сцені;

- прозорий макет – макет, що слідкує за мишею гравця при обиранні місця, якщо його поточне положення задовільне;
- червоний макет – макет, що слідкує за мишею гравця при обиранні місця, якщо його поточне положення незадовільне;

Зберігає значення ціни розташування даного типу будівлі.

18. CameraController – клас, котрий здійснює рух основної камери гравця по карті, її обертання, масштабування, прискорення і тд.
19. Message – клас, що відповідає за відображення повідомлень на екрані. Використовується для повідомлень при нестачі коштів, чи невдало підібраному місці для розташування будівлі.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ГРИ

3.1 Обрана методологія розробки

Методологією розробки даної гри було обрано Scrum.

Scrum — це гнучка методологія розробки, який використовується для управління проектами, особливо в середовищах, що потребують швидкої адаптації до змін. Scrum організовує виконання роботи у короткі, фіксовані цикли, звані спринтами, які зазвичай тривають від одного до чотирьох тижнів. Основні принципи Scrum полягають у постійному вдосконаленні, тісній співпраці, швидкому зворотному зв'язку і адаптивному плануванні [7].

Методологія Scrum підходить для розробки гри на Unity соло розробником завдяки таким його факторам:

а) Гнучкість і адаптивність: Scrum дозволяє швидко адаптуватися до змін вимог або пріоритетів, що можуть виявитись під час процесу розробки, що дуже важливо для розробки гри, де вимоги можуть змінюватися на основі нових ідей;

б) Мотивація і фокус: Робота у коротких спринтах допомагає підтримувати високу мотивацію і зосередженість на конкретних завданнях. Надається можливість бачити результати своєї роботи через короткі проміжки часу, що підтримує відчуття прогресу;

в) Планування і організація: Scrum допомагає структурувати роботу і ставити чіткі цілі на кожен спринт. Це особливо корисно для соло розробника, оскільки допомагає уникнути відчуття перевантаженості і хаосу;

г) Самоаналіз і вдосконалення: Регулярні ретроспективи дозволяють оцінити свою роботу і знайти способи покращення процесу. Це сприяє постійному вдосконаленню навичок і процесів;

д) Пріоритизація завдань: В ролі власника продукту можна визначати, які завдання є найважливішими і фокусуватися на них. Це допомагає ефективно використовувати час і ресурси;

е) Зворотний зв'язок: Огляди спринтів дозволяють отримати зворотний

зв'язок від потенційних користувачів або тестувальників на ранніх етапах, що допомагає виявити проблеми та покращити гру до її випуску.

Scrum підходить для розробки гри на Unity соло розробником завдяки своїй гнучкості, чіткій структурі, можливості самоаналізу та постійному вдосконаленню. Цей фреймворк допомагає ефективно організувати роботу, підтримувати високу мотивацію і адаптуватися до змін, що є ключовими факторами успішної розробки ігор.

3.2 Проведення спринтів

Оскільки обраною методологією є Scrum, то процес розробки гри здійснювався спринтами, із поставленими цілями, що необхідно виконати за визначений період часу, підведенням підсумків у кінці та тестуванням розробленого компоненту.

Далі буде наведено приклад одного спринта за методологією Scrum, але із відсутністю зустрічей та обговорень, оскільки розробка здійснюється соло-розробником.

Спринт призначений для розробки режиму будівництва та його реалізації:

Тривалість: 2 тижні.

Підготовка до спринта: був проведений аналіз системи та поточного прогресу, було визначено пріоритетні функції гри, що найбільше потребують розробки на даний момент, створено список завдань що необхідно виконати за поточний спринт, розраховано час для його виконання.

Завдання спринта:

а) розробити меню для увімкнення режиму будівництва забезпечити його увімкнення та слідкування за вершиною штабу;

б) створити усі необхідні макети будівель макети будівель, а саме:

- класична модель: Ця модель являє собою готове візуальне

представлення будівлі, яке буде розміщене на сцені;

- прозорий макет: Цей макет динамічно змінюється, слідуючи за курсором гравця, якщо його поточне положення вважається задовільним;
- червоний макет: Цей макет також динамічно змінюється, слідуючи за курсором гравця, але візуально позначає місця, які не відповідають певним критеріям (наприклад, недоступні або небезпечні);

в) розробити клас, що вміщатиме усі необхідні моделі, назву та ціну будівлі;

г) реалізувати відображення полів для розміщення навколо будівель бази при увімкненні режиму будівництва;

д) реалізувати функцію зміни моделей відносно того, можна розмістити будівлю в даному місці, чи ні;

е) забезпечити відображення повідомлень у випадку недостатньої кількості матеріалів для будівництва будівлі, або нестачі місця для побудови;

є) створити можливість розміщення кінцевої будівлі та додавання її до про глядачів «туману війни»;

ж) забезпечити списання матеріалів за побудову певної будівлі відповідно до її вартістю;

Щоденний аналіз: кожного дня здійснювався аналіз функцій та частин коду що були реалізовані, виставлялися пріоритетні цілі та проводилися розрахунки для визначення прогресу розробки щоб визначити успішність спринта на даному етапі.

Огляд результатів спринта тестування розробленого функціоналу:

а) було розроблено меню для увімкнення будівництва, намальовано іконку кнопки для його увімкнення;

Рисунок 3.1 – Меню штабу, іконка для режиму будівництва

б) було створено макети будівель, для прикладу на наступних рисунках буде зображено макети арсеналів:

– класичний макет:

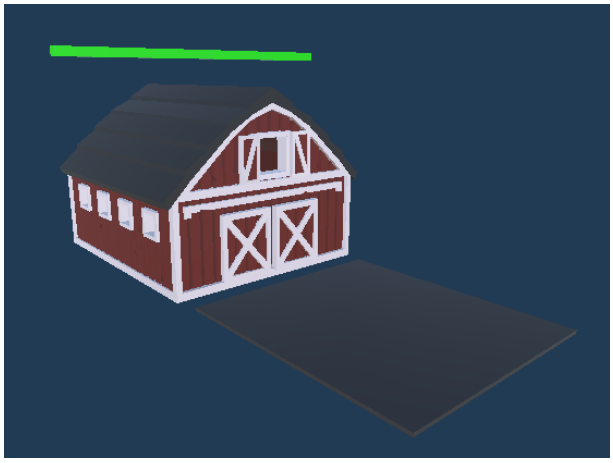


Рисунок 3.2 – Класична модель арсеналу

– прозорий макет:

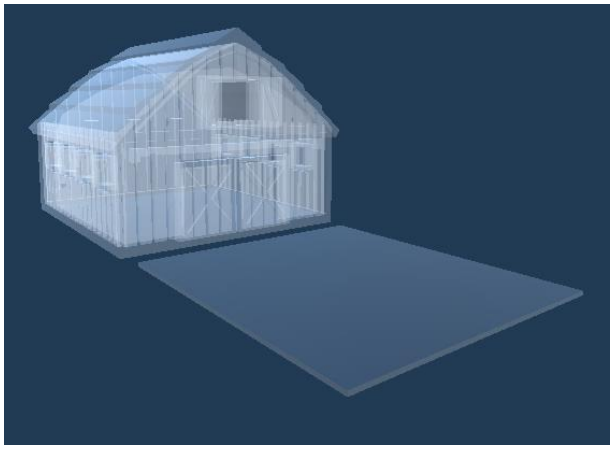


Рисунок 3.3 – Прозора модель арсеналу

– червоний макет:

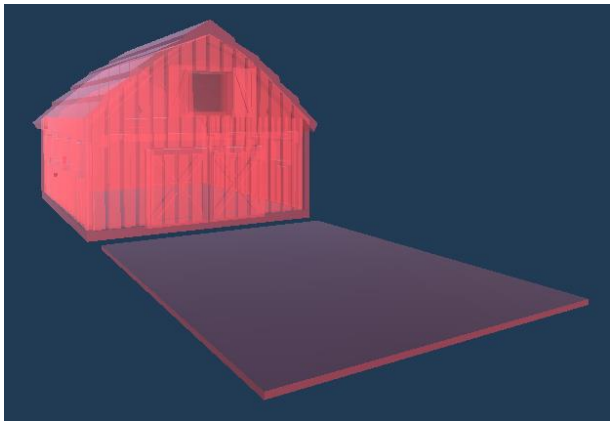


Рисунок 3.4 – Червона модель арсеналу

в) було створено клас, що являє собою набір усіх необхідних моделей будівлі, її назви та ціни:

```
public class BuildPrefs
{
    Ссылок: 1
    public String typeName;
    Ссылок: 2
    public GameObject clasic;
    Ссылок: 3
    public GameObject allow;
    Ссылок: 2
    public GameObject deny;
    Ссылок: 4
    public Wallet.Reciept price;
    Ссылок: 0
    public BuildPrefs(String typeName, GameObject clasic, GameObject allow, GameObject deny, Wallet.Reciept price)
    {
        this.typeName = typeName;
        this.clasic = clasic;
        this.allow = allow;
        this.deny = deny;
        this.price = price;
    }
}
```

```
class UnityEngine.GameObject
Base class for all entities in Unity Scenes.
```

Рисунок 3.5 – Лістинг класу BuildPrefs

г) було реалізовано відображення полів що помічають місця для будування нових споруд:

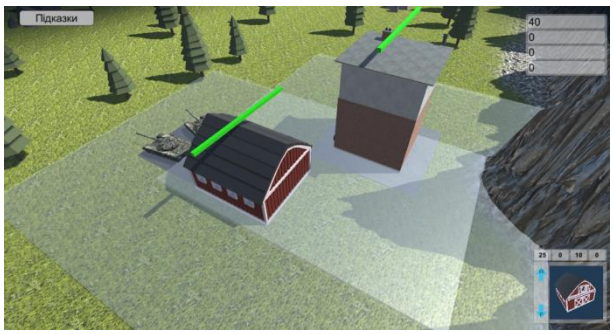


Рисунок 3.6 – Відображення полів для будівництва

д) реалізована динамічна зміна моделей будівлі, відносно із дозволом його розміщення у даному місці:

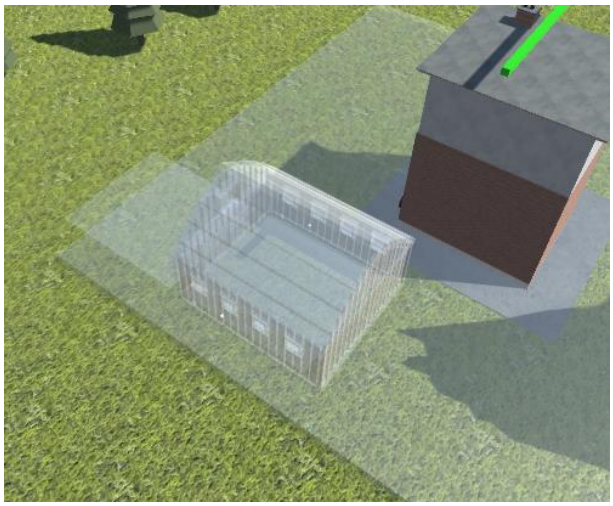


Рисунок 3.7 – Модель арсеналу у задовільному для розміщення місці

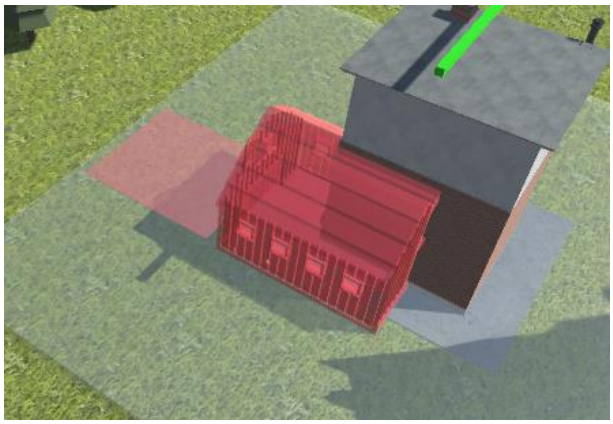


Рисунок 3.8 – Модель арсеналу при нестачі місця для його розміщення

е) здійснене забезпечення відображення повідомлень при відхиленні у

розміщенні будівлі:

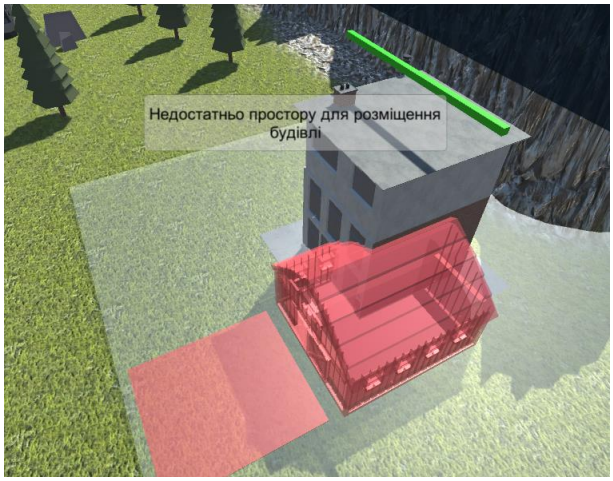


Рисунок 3.9 – Повідомлення про нестачу простору для розміщення будівлі

є) було реалізовано розміщення кінцевої моделі будівлі у задовільному для цього місці:

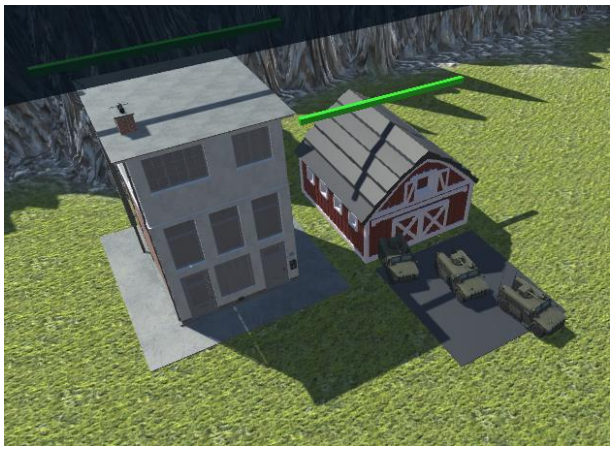


Рисунок 3.10 – Результат розміщення арсеналу біля штабу

ж) було забезпечено списання матеріалу згідно із ціною будівлі

Підведення підсумків: Під час розробки даного функціоналу було створено моделі для розміщень усіх типів будівель. Основні моделі було завантажено із Asset Store – офіційного сайту із заготовками та 3D-моделями від Unity. Прозора та червона моделі були створені у редакторі Unity, встановивши прозорий та прозоро-червоний матеріали для моделей відповідно.

Зовнішній вигляд будівель не завжди відповідає їх сутності оскільки кількість безкоштовних моделей у Asset Store є дуже обмеженою та вибір є дуже малим. У подальшому планується залучитися підтримкою 3D-дизайнерів для створення більш відповідних моделей споруд та транспорту.

Іконка для кнопки режиму будівництва була намальована вручну у програмі Adobe Illustrator у векторному форматі для кращого відображення у малих розмірах.

Поля для розміщення відображаються навколо кожної будівлі що належить до певної бази. При тестуванні було помічено що прозорість полів накладається одна на одну та виглядає більш насиченими у місцях стиків. У подальшому планується покращення матеріалів поля щоб запобігти такому їх відображенню.

Здійснювалось тестування у вигляді спроб розташування будівель в незадовільних місцях та при нестачі ресурсів, розміщення великої кількості будівель та здійснена перевірка продуктивності та кількості обчислювальних ресурсів у таких випадках.

Завершення розробки усіх поставлених цілей спринта, проведення тестування та здійснення аналізу спринта було здійснено вчасно, відповідно до виділеного часу.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Надзвичайні ситуації: визначення причини, класифікація.

Надзвичайні ситуації можуть виникнути у будь який момент, тому потрібно вміти класифікувати їх та вирізняти причини виникнення надзвичайних ситуацій щоб запобігти їх утворенню, якщо це можливо, та знати елементарні правила поведінки в таких ситуаціях.

Надзвичайною ситуацією вважається така, що несе за собою загрозу життю та здоров'ю значної кількості людей, може призвести до суттєвого порушення екологічної рівноваги, зупинити або частково обмежити господарську діяльність, а також завдати значних матеріальних та економічних збитків [8].

Існують різні типи надзвичайних ситуацій, які розрізняються за їх сутністю та причинами виникнення [8]:

- природні надзвичайні ситуації: пов’язані з природними процесами космічного, літосферного, гідросферного, атмосферного, біосферного характеру або кількох процесів одночасно і відбуваються незалежно від участі людини;
- соціальні надзвичайні ситуації: відбуваються в суспільстві: воєнний стан, злочинність, революції, міжнаціональні конфлікти, поширення людиноненависницьких ідеологій та ін;
- техногенні надзвичайні ситуації: пов’язані з матеріальною сферою, що створена людиною.

За масштабом та глибиною НС поділяють на:

- локальна НС – загроза її виникнення та розповсюдження наслідків
- обмежена виробничим приміщенням.
- об’єктова НС – обмежена територією об’єкта.
- місцева НС – обмежена територією населеного пункту, району чи області.
- регіональна НС – обмежена територією декількох областей, краю чи

суміжних країн.

- національна НС – наслідки охоплюють великі території держави, але не виходять за її кордони.
- глобальна НС – загроза її виникнення та поширення наслідків - континент або значна його частина чи планета в цілому.

У кожному конкретному випадку надзвичайні ситуації виникають через ряд причин, які можна узагальнити як [9]:

а) природні надзвичайні ситуації та небезпечні явища:

- закономірні природні процеси;
- негативний антропогенний вплив на розвиток природних процесів;
- випадковість у розвитку природних процесів.

б) соціальні надзвичайні ситуації:

- політичні;
- економічні;
- національні;
- релігійні.

в) техногенні надзвичайні ситуації:

- недодержання правил безпеки та необережність;
- недосконалість у проектуванні;
- кримінальні;
- елементи та тероризм;
- воєнні дії;
- природні явища.

4.2 Загальні вимоги безпеки з охорони праці для користувачів ПК.

Розробка 3D-ігор, зокрема на платформі Unity, вимагає тривалого перебування за комп'ютером. Це включає програмування, тестування,

моделювання та інші етапи створення гри. Така діяльність пов'язана з певними ризиками для здоров'я, які можна мінімізувати, дотримуючись загальних вимог безпеки з охорони праці.

Ефективність та безпека праці значною мірою залежать від вдалого розподілу функцій між людиною та машиною. Конструкція виробничого обладнання має відповідати ергономічним принципам, щоб створити сприятливі умови праці, знизити рівень її важкості та напруженості, а також забезпечити високу продуктивність системи.

Для цього обладнання має відповідати антропометричним, фізіологічним, психологічним та психофізіологічним характеристикам людини. Це означає, що воно має бути зручним у користуванні, відповідати розмірам та силі м'язів людини, не спричиняти надмірного навантаження на опорно-руховий апарат, органи чуття та нервову систему [9].

Ергономічно спроектоване обладнання не лише покращує умови праці, але й сприяє зниженню травматизму, підвищенню якості продукції та загальної працездатності працівників.

Ефективність, безпека та комфорт праці багато в чому залежать від ергономічних характеристик трудового процесу. До ключових показників, які їх визначають, належать [10]:

- гігієнічні показники: характеризують вплив зовнішнього середовища на працівника, включаючи температуру, фізико-хімічний склад повітря, освітленість, рівень шуму та інші фактори.
- антропометричні та біомеханічні показники: оцінюють відповідність знарядь праці розмірам, формі та масі тіла людини, силі та напрямку її рухів.
- фізіологічні та психофізіологічні показники: визначають відповідність темпу, енерговитрат, зорових та інших фізіологічних характеристик людини вимогам технологічного процесу.
- психологічні показники: оцінюють відповідність закріплених та сформованих навичок, можливостей сприйняття, пам'яті та мислення

людини завданням, які вона виконує.

– естетичні показники: визначають відповідність естетичних потреб людини художньо-конструкторським рішенням робочих місць, знарядь праці та виробничого середовища.

Комплексний аналіз цих показників дозволяє створити оптимальні умови праці, які мінімізують ризики травматизму, підвищують продуктивність та покращують загальне самопочуття працівників.

Оптимальна організація робочого місця, що ґрунтується на принципах ергономіки, є запорукою високої працездатності та комфорту працівника. Важливу роль у цьому відіграє конструкція робочого місця, яка має відповідати певним ергономічним вимогам, особливо коли йдеться про роботу з комп'ютером у сидячому положенні (рис. 4.1) [9].

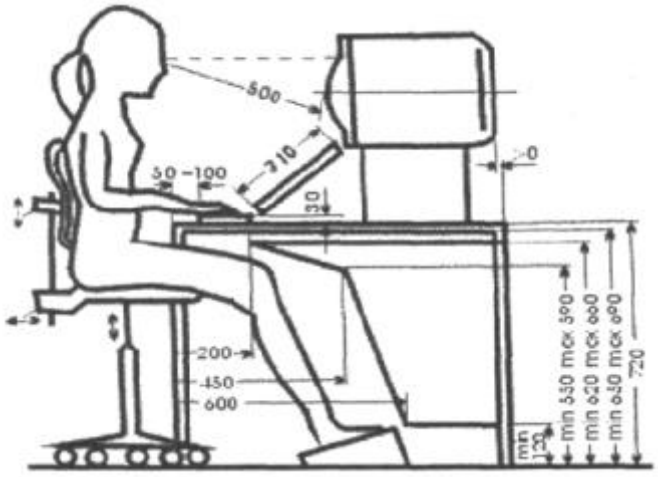


Рисунок 4.1 – Вимоги до конструкції робочого місця користувача ПК

Для операторів, які протягом тривалого часу виконують роботу в сидячому положенні, важливо забезпечити оптимальну робочу позу. Це досягається за допомогою регулювання висоти робочої поверхні та сидіння.

Під час обладнання робочих місць з комп'ютерами (ПК) або відеотерміналами важливо враховувати санітарно-гігієнічні норми, які встановлюють мінімальні розміри робочої зони.

Згідно з цими нормами, площа, виділена на одне таке робоче місце, має становити не менше 6 квадратних метрів, а об'єм - не менше 20 кубічних метрів [8].

Розмір робочого столу має суттєве значення. Його висота повинна становити 725 мм, що дозволяє тримати руки в зручному положенні під час роботи з клавіатурою та мишею. Ширина столу може варіюватися в межах 600-1400 мм, щоб можна було розмістити все необхідне обладнання та документи. Оптимальна глибина столу знаходиться в діапазоні 800-1000 мм, забезпечуючи достатній простір для роботи [9].

Важливим фактором є також вільний простір для ніг. Висота під столом має бути не менше 600 мм, ширина - не менше 500 мм. Глибина на рівні колін повинна становити не менше 450 мм, а на рівні витягнутої ноги - не менше 650 мм. Це дозволить вам сидіти в зручній позі, не відчуваючи дискомфорту [9].

Якщо ноги не дістають до підлоги, при сидінні в оптимальній робочій позі, рекомендується використовувати підставку для ніг. Це допоможе зняти навантаження з хребта та покращити кровообіг.

Важливо також правильно організувати робочу зону. Монітор має бути розташований на відстані 50-70 см від ваших очей, трохи вище рівня очей. Клавіатура та миша повинні знаходитися на такій висоті, щоб зап'ястя були на одній лінії з переднім краєм столу [10].

Дотримання цих простих правил ергономіки дозволить створити комфортне та здорове робоче місце, яке допоможе забезпечити більшу продуктивність та зменшити втомлюваність протягом дня.

ВИСНОВКИ

У результаті написання кваліфікаційної роботи бакалавра на тему «Розробка 3D-гри на Unity» було розроблено бета-версію 3D-гри жанру «стратегія» під назвою «War of Technique».

Гра розроблялася на основі ігрового двигуна Unity, котрий, як і очікувалося, виявився ідеальним варіантом для розробки гри соло-розробником. З його допомогою у відносно короткі терміни було розроблено гарно оптимізовану гру без високих вимог до програмного забезпечення кінцевого користувача. Більшість 3D-моделей об’єктів, що є частиною гри, були завантаженні із Asset Store – офіційного сайту із заготовками та 3D-моделями від Unity, велика кількість яких є абсолютно безкоштовними.

У процесі проектування гри було проведено аналіз вимог до програмного забезпечення, обрано відповідний тип архітектури та створено UML-діаграми що відоражають зв’язок між компонентами системи, а саме: діаграма варіантів використання, діаграма послідовностей, діаграма класів.

Було здійснено детальний опис та документацію чотирьох наймасивніших класів системи та більш загальний опис решти класів, що також є невід’ємною частиною реалізації функціоналу гри.

Здійснено опис проведення спринта по створенню функції побудови нових споруджень навколо штабу та відображено результати виконаної роботи. Було описане проведення тестування функціоналу програмного забезпечення розробленого під час виконання цього спринта.

Задокументовано розділ безпеки життєдіяльності та основ охорони праці про загальні вимоги безпеки з охорони праці для користувачів персонального комп’ютеру, що є невід’ємно пов’язаним із розробниками програмного забезпечення. Було описано категорії та причини виникнення надзвичайних ситуацій, адже вони можуть виникнути у будь який момент, в тому числі під час розробки гри.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Офіційний сайт ігрового двигуна Unity [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com/.

2. Що таке Unity і для чого використовується? [Електронний ресурс] – Режим доступу до ресурсу: https://lemon.school/blog/shho-take-unity.

3. Asset Store [Електронний ресурс] – Режим доступу до ресурсу: https://assetstore.unity.com/.

4. UML-діаграми [Електронний ресурс] – Режим доступу до ресурсу: https://evergreens.com.ua/ua/articles/uml-diagrams.html.

5. Інструкція створення UML-діаграм [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/40575/.

6. Компонентна архітектура [Електронний ресурс] – Режим доступу до ресурсу: https://studfile.net/preview/5200675/page:22/.

7. Scrum [Електронний ресурс] – Режим доступу до ресурсу: https://career.softserveinc.com/uk-ua/stories/what-is-scrum-methodology.

8. Воробієнко П. П. Навчальний посібник із безпеки життєдіяльності [Електронний ресурс] / П. П. Воробієнко, М. В. Захарченко, Л. В. Орел // ОНАЗ ім. О. С. Попова. – 2013. – Режим доступу до ресурсу: https://duikt.edu.ua/uploads/l_13_53225213.pdf.

9. «Безпека життєдіяльності»: навч. посібник [Електронний ресурс] / Т. Є.Стиценко, Г. В. Пронюк, Н. М. Сердюк, Н. М. Хондак // Харків: ХНУРЕ. – 2018. – Режим доступу до ресурсу: https://os.nure.ua/wp-content/uploads/2019/05/posibnik-bgd_2018_p.1.pdf.

10. ДСанПІН 3.3.2.007-98 "Вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин". [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2801-12#Text.

ДОДАТКИ

ДОДАТОК А

Лістинги коду класів системи

Лістинг 1 - Клас Selecting:

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine.AI;
using UnityEngine;
using System;
using UnityEngine.UI;
using UnityEngine.Animations;
using UnityEngine.EventSystems;
//using TreeEditor;
// using UnityEditor.PackageManager.UI;
// using UnityEditor.AI;


public class Selecting : MonoBehaviour
{

    public GameObject selectFieldPref;
    public LayerMask layer;
    public LayerMask allLayers;
    [NonSerialized] public GameObject selectField;
    private UnityEngine.Vector3 firstHit;
    public GameObject crossPref;
    [NonSerialized]   public   List<GameObject>   crosses   =   new
List<GameObject>();
    [NonSerialized]   public   List<GameObject>   players   =   new
List<GameObject>();

    private void Update()
    {
        if(Input.GetKeyUp(KeyCode.K))
        {
            //Перевірити слой
            Debug.Log(CastRay(allLayers).transform.gameObject.layer);
        }
        if(players.Count!=0)
        {
            //Видаляти усі змінні у масиву із значенням null
            players.RemoveAll(item => item == null);
        }

if(Input.GetMouseButtonDown(0)&&!selectField&&CastRay(layer).transform.game
Object.layer==7)
        {
            //Створити поле
            firstHit = CastRay(layer).point;
            selectField    =    Instantiate(selectFieldPref,    new
UnityEngine.Vector3(firstHit.x, 0.5f, firstHit.z), transform.rotation);
```

```
            }
            if(selectField)
            {
                //Змінити розмір поля
                UnityEngine.Vector3                first                =
transform.InverseTransformPoint(firstHit);
                UnityEngine.Vector3              current                =
transform.InverseTransformPoint(CastRay(layer).point);
                UnityEngine.Vector3         newScale         =         new
UnityEngine.Vector3((first.x - current.x)*-1, 1, (first.z - current.z)*-1);
                selectField.transform.localScale = newScale;
            }

if(Input.GetKey(KeyCode.LeftShift)&&Input.GetMouseButtonUp(0)&&selectField)
            {
                //Додати виділений транспорт до активного
                foreach(GameObject el in CollectField(selectField))
                {

if(el.layer==6&&el.CompareTag("Tech")&&!players.Contains(el))
                    {
                        players.Add(el);

el.transform.GetChild(0).gameObject.SetActive(true);
                    }
                }
                Destroy(selectField);
            }
            else
if(Input.GetKey(KeyCode.LeftControl)&&Input.GetMouseButtonUp(0)&&selectFiel
d)
            {
                //Видалити виділений транспорт з активних
                foreach(GameObject el in CollectField(selectField))
                {

if(el.layer==6&&el.CompareTag("Tech")&&players.Contains(el))
                    {
                        players.Remove(el);

el.transform.GetChild(0).gameObject.SetActive(false);
                    }
                }
                Destroy(selectField);
            }
            else
if(Input.GetMouseButton(0)&&Input.GetMouseButtonUp(1)&&selectField)
            {
                //Скасувати виділення
                Destroy(selectField);
            }
            else
if(Input.GetKey(KeyCode.Space)&&Input.GetMouseButtonUp(0)&&selectField)
            {
                //Перевірка які об'єкти є всередині поля
                Debug.Log("There is --------------");
                foreach(GameObject el in CollectField(selectField))
                {
```

```
                    Debug.Log(el.name);
                }
                Destroy(selectField);
            }
            else if(Input.GetMouseButtonUp(0)&&selectField)
            {
                //Очистити масив і додати транспорт
                foreach (GameObject el in players)
                {
                    el.transform.GetChild(0).gameObject.SetActive(false);
                }
                players.Clear();
                foreach(GameObject el in CollectField(selectField))
                {
                    if(el.layer==6&&el.CompareTag("Tech"))
                    {
                        players.Add(el);

el.transform.GetChild(0).gameObject.SetActive(true);
                    }
                }
                Destroy(selectField);
            }

if(Input.GetMouseButtonUp(1)&&!selectField&&CastRay(allLayers).transform.ga
meObject.layer==7&&players.Count!=0)
            {
                //Перемістити гравців
                UnityEngine.Vector3             destination             =
CastRay(allLayers).point;
                if(!Input.GetKey(KeyCode.LeftShift))
                {
                    foreach(GameObject el in crosses)
                    {
                        Destroy(el);
                    }
                }
                crosses.Add(Instantiate(crossPref,            destination,
transform.rotation));
                foreach (GameObject el in players)
                {
                    if(el.GetComponent<Shooting>().victim!=null)
                    {
                        Shooting shooting = el.GetComponent<Shooting>();

if(UnityEngine.Vector3.Distance(el.transform.position,
shooting.victim.transform.position)>shooting.shootingDistance)
                        {
                            //StopCoroutine(shooting.ShootingProcess);
                            shooting.ShootingProcess = null;
                            shooting.victim = null;
                        }
                    }
                    crosses[crosses.Count-
1].GetComponent<CrossManager>().riders.Add(el);
                    if(!Input.GetKey(KeyCode.LeftShift))
                    {
                        el.GetComponent<Moving>().ForgetAll();
```

```
                    }

el.GetComponent<Moving>().destinations.Add(destination);
                }
            }

if(Input.GetKey(KeyCode.LeftShift)&&Input.GetMouseButtonUp(1)&&!selectField
&&CastRay(layer).transform.gameObject.layer==8&&players.Count!=0)
            {
                //Атакувати переслідуючи
                RaycastHit hit = CastRay(layer);
                foreach(GameObject el in crosses)
                {
                    Destroy(el);
                }
                foreach(GameObject el in players)
                {
                    if(hit.transform.gameObject!=null)
                    {
                        el.GetComponent<Moving>().ForgetAll();

el.GetComponent<Shooting>().StartDestroying(hit.transform.gameObject,
true);
                    }
                }
            }
            else
if(Input.GetMouseButtonUp(1)&&!selectField&&CastRay(layer).transform.gameOb
ject.layer==8&&players.Count!=0)
            {
                //Атакувати об'єкт
                RaycastHit hit = CastRay(layer);
                foreach(GameObject el in crosses)
                {
                    Destroy(el);
                }
                foreach(GameObject el in players)
                {
                    if(hit.transform.gameObject!=null)
                    {
                        el.GetComponent<Moving>().ForgetAll();

el.GetComponent<Shooting>().StartDestroying(hit.transform.gameObject,
false);
                    }
                }
            }

        }
        public static RaycastHit CastRay(LayerMask layer)
        {
            //Запустити промінь на певні слої
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            Physics.Raycast(ray, out RaycastHit hit, 3000, layer);
            // if (EventSystem.current.IsPointerOverGameObject())
            // {
            //          return new RaycastHit(); // Повернути пустий
результат, не оброблюючи натискання на UI
```

```
            // }
            return hit;
        }
        public static List<GameObject> CollectBuildBox(GameObject field)
        {
            //Зібрати всі об'єкти що є у колайдері будівлі отримуючи
вручну вказаний розмір зі скрипта Sizing
            UnityEngine.Vector3 size = field.GetComponent<Sizing>().size;
            Collider                selectFieldCollider                 =
field.GetComponent<Collider>();
            Collider[]                  collidersInside                 =
Physics.OverlapBox(selectFieldCollider.bounds.center,                   new
UnityEngine.Vector3(size.x/2,             size.y/2,             size.z/2),
field.transform.rotation);
            List<GameObject> objs = new List<GameObject>();
            foreach(Collider el in collidersInside)
            {
                if(!objs.Contains(el.gameObject))
                    objs.Add(el.gameObject);
            }
            return objs;
        }
        public static List<GameObject> CollectField(GameObject field)
        {
            //Зібрати всі об'єкти що є у колайдері поля
            Collider                selectFieldCollider                 =
field.transform.GetChild(0).GetComponent<Collider>();
            Collider[]                  collidersInside                 =
Physics.OverlapBox(selectFieldCollider.bounds.center,                   new
UnityEngine.Vector3(MathF.Abs(field.transform.localScale.x)/2,
MathF.Abs(field.transform.localScale.y)/2,
MathF.Abs(field.transform.localScale.z)/2), field.transform.rotation);
            List<GameObject> objs = new List<GameObject>();
            foreach(Collider el in collidersInside)
            {
                if(!objs.Contains(el.gameObject))
                    objs.Add(el.gameObject);
            }
            return objs;
        }
    }
```

Лістинг 2 – Клас Shooting

```
using System;
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.PlayerLoop;
using UnityEngine.AI;
using System.Runtime.CompilerServices;

public class Shooting : MonoBehaviour
{
    [NonSerialized] public Coroutine ShootingProcess;
    [NonSerialized] public Coroutine SecShootingProcess;
    [NonSerialized] public Coroutine TowerLooking;
```

```
        [NonSerialized] public GameObject victim;
        [NonSerialized] public GameObject secVictim;
        [NonSerialized] public Coroutine follow;
        public GameObject bullet;
        public float shootingDiapazone;
        public float shootingDistance;
        public float bulletSpeed;
        public float shootingDamage;
        public float towerRotateSpeed;
        public GameObject tower;

        public void StartDestroying(GameObject _victim, bool isFollow)
        {
            if(victim!=_victim)
            {
                victim = _victim;
                if(ShootingProcess!=null)
                {
                    if(gameObject.CompareTag("Tech"))

gameObject.GetComponent<NavMeshAgent>().SetDestination(gameObject.transfor
m.position);
                    StopCoroutine(ShootingProcess);
                }
                if(isFollow)
                    ShootingProcess = StartCoroutine(Destroying(true));
                else
                    ShootingProcess = StartCoroutine(Destroying(false));

                if(gameObject.layer == 8&&gameObject.CompareTag("Tech"))
                {
                    GetComponent<EnemyMoving>().ForgetAll();

gameObject.GetComponent<NavMeshAgent>().SetDestination(gameObject.transfor
m.position);
                }
            }
            else if(isFollow&&follow == null)
            {
                follow = StartCoroutine(Follow());
            }
            else if(!isFollow&&follow != null)
            {
                StopCoroutine(follow);
                follow = null;
            }
        }
        IEnumerator Destroying(bool isFollow)
        {
            if(TowerLooking!=null)
                StopCoroutine(TowerLooking);
            TowerLooking = StartCoroutine(TowerLook(victim));
            String status = "Out";

if(gameObject.CompareTag("Tech")&&UnityEngine.Vector3.Distance(gameObject.
transform.position, victim.transform.position)>shootingDistance)
            {
                UnityEngine.Vector3              direction              =
```

```
(victim.transform.position - gameObject.transform.position).normalized;
                UnityEngine.Vector3             newposition             =
victim.transform.position - direction * (shootingDistance-5);

gameObject.GetComponent<NavMeshAgent>().SetDestination(newposition);
            }
            if(isFollow)
                follow = StartCoroutine(Follow());
            GameObject amo;
            while(victim!=null)
            {
                if(status                    ==                    "Out"&&
UnityEngine.Vector3.Distance(victim.transform.position,
gameObject.transform.position)<shootingDistance+3)
                {
                    if(TowerLooking!=null)
                        StopCoroutine(TowerLooking);
                    TowerLooking = StartCoroutine(TowerLook(victim));
                }
                else
if(secVictim&&status=="Out"&&UnityEngine.Vector3.Distance(secVictim.transf
orm.position, gameObject.transform.position)<shootingDistance+3)
                {
                    if(TowerLooking!=null)
                        StopCoroutine(TowerLooking);
                    TowerLooking = StartCoroutine(TowerLook(secVictim));
                }
                yield return new WaitForSeconds(shootingDiapazone);
                if(victim!=null)
                {
                    if(status           ==           "Out"           &&
UnityEngine.Vector3.Distance(victim.transform.position,
gameObject.transform.position)<shootingDistance)
                        status = "In";
                    if(status           ==           "In"           &&
UnityEngine.Vector3.Distance(victim.transform.position,
gameObject.transform.position)>shootingDistance)
                    {
                        if(follow == null)
                            status = "Gone";
                        else
                            status = "Out";
                    }
                    if(status=="In")
                    {
                        amo             =             Instantiate(bullet,
tower.transform.GetChild(0).position, Quaternion.identity);

amo.GetComponent<BulletManager>().ShootBullet(victim, amo, shootingDamage,
bulletSpeed);
                    }
                    if(status == "Gone")
                        break;
                }

if(secVictim&&status=="Out"&&UnityEngine.Vector3.Distance(secVictim.transf
orm.position, gameObject.transform.position)<shootingDistance)
                {
```

```
                    amo                =                Instantiate(bullet,
tower.transform.GetChild(0).position, Quaternion.identity);

amo.GetComponent<BulletManager>().ShootBullet(secVictim,                amo,
shootingDamage, bulletSpeed);
                }
            }
            ShootingProcess = null;
            victim = null;
            follow = null;
            secVictim = null;
            if(TowerLooking!=null)
                StopCoroutine(TowerLooking);
            TowerLooking = StartCoroutine(TowerStraight());
        }
        IEnumerator Follow()
        {
            while(victim!=null)
            {

if(UnityEngine.Vector3.Distance(victim.transform.position,
gameObject.transform.position)>shootingDistance)
                {
                    UnityEngine.Vector3            direction            =
(victim.transform.position - gameObject.transform.position).normalized;
                    UnityEngine.Vector3          newposition            =
victim.transform.position - direction * (shootingDistance-5);

gameObject.GetComponent<NavMeshAgent>().SetDestination(newposition);
                }
                yield return new WaitForEndOfFrame();
            }
        }
        IEnumerator TowerLook(GameObject _victim)
        {
            while(_victim!=null)
            {
                Quaternion                    rotation                    =
Quaternion.LookRotation(_victim.transform.position                          -
tower.transform.position, UnityEngine.Vector3.up);
                tower.transform.rotation                                    =
Quaternion.RotateTowards(tower.transform.rotation,                rotation,
towerRotateSpeed*Time.deltaTime);
                yield return new WaitForFixedUpdate();
            }
        }
        IEnumerator TowerStraight()
        {
            while(tower.transform.rotation!=transform.rotation)
            {
                tower.transform.rotation                                    =
Quaternion.RotateTowards(tower.transform.rotation,     transform.rotation,
towerRotateSpeed*Time.deltaTime);
                yield return new WaitForFixedUpdate();
            }
        }
        public bool isThere(UnityEngine.Vector3 target)
        {
```

```
            if(UnityEngine.Vector3.Distance(target,   transform.position)
< gameObject.GetComponent<NavMeshAgent>().stoppingDistance)
                return true;
            else
                return false;
        }

    }
```

Лістинг 3 – Клас Building

```
using System;
using System.Collections;
using System.Collections.Generic;
using FischlWorks_FogWar;
using Unity.VisualScripting;
using UnityEngine;

public class Building : MonoBehaviour
{
    public GameObject fieldPref;
    public float fieldWidth;
    public float rotateSpeed;
    private Quaternion curRotation;
    public List<BuildPrefs> buildPrefs = new List<BuildPrefs>();
    private List<GameObject> fields = new List<GameObject>();
    public LayerMask allLayers;
    public LayerMask ground;
    [NonSerialized] public GameObject placingBuilding;
    [NonSerialized]  public  List<GameObject>  ownBuildings  =  new
List<GameObject>();
    [NonSerialized] public BuildChooser buildChooser;
    private bool isAllow;
    private bool isCreating;

    void Start()
    {
        buildChooser                                                    =
GameObject.Find("EventSystem").GetComponent<BuildChooser>();
        buildChooser.headBuilding = GetComponent<Building>();
        isCreating = false;
        if(!ownBuildings.Contains(gameObject))
            ownBuildings.Insert(0, gameObject);
    }
    void Update()
    {
        if(isCreating&&Input.GetMouseButtonUp(1))
        {
            //Зупиняємо будування
            StopBuilding();
        }
        if(buildChooser.isRefreshing)
        {
            //Оновлюємо заготовку на інший тип
            Destroy(placingBuilding);
            buildChooser.isRefreshing = false;
        }
```

```
if(isCreating&&!placingBuilding&&Selecting.CastRay(allLayers).transform.ga
meObject.name == "PlacingField(Clone)")
            {
                //Поставити заготовку якщо вона в полі розташування
                placingBuilding                                          =
Instantiate(buildPrefs[buildChooser.current].allow,
Selecting.CastRay(ground).point, curRotation);
                isAllow = true;
            }

if(isCreating&&placingBuilding&&Selecting.CastRay(allLayers).transform.gam
eObject.name != "PlacingField(Clone)")
            {
                //Видалити заготовку якщо вона поза полем розташування
                Destroy(placingBuilding);
            }
            if(isCreating&&placingBuilding)
            {
                //Переміщати заготовку
                placingBuilding.transform.position                       =
Selecting.CastRay(ground).point;
                //Крутити заготовку
                if(Input.GetKey(KeyCode.Z))
                {
                    placingBuilding.transform.Rotate(new
UnityEngine.Vector3(0f,1f,0f)*Time.deltaTime*rotateSpeed);
                    curRotation = placingBuilding.transform.rotation;
                }
                if(Input.GetKey(KeyCode.X))
                {
                    placingBuilding.transform.Rotate(new
UnityEngine.Vector3(0f,-1f,0f)*Time.deltaTime*rotateSpeed);
                    curRotation = placingBuilding.transform.rotation;
                }
                //Перевірка чи є об'єкти всередині будівлі
                isAllow = true;
                foreach(GameObject                  el                   in
Selecting.CollectBuildBox(placingBuilding))
                {
                    if(el.layer!=11)
                    {
                        isAllow = false;
                        break;
                    }
                }
                //Встановлення правильного кольору
                if(!isAllow&&!placingBuilding.name.Contains("Deny"))
                {
                    //Якщо є, але не червоне
                    UnityEngine.Vector3              pos                 =
placingBuilding.transform.position;
                    Destroy(placingBuilding);
                    placingBuilding                                      =
Instantiate(buildPrefs[buildChooser.current].deny, pos, curRotation);
                }
                else if(isAllow&&placingBuilding.name.Contains("Deny"))
                {
                    //Якщо немає, але не прозоре
```

```
                    UnityEngine.Vector3              pos              =
placingBuilding.transform.position;
                    Destroy(placingBuilding);
                    placingBuilding                                   =
Instantiate(buildPrefs[buildChooser.current].allow, pos, curRotation);
                }
            }

if(isCreating&&placingBuilding&&Input.GetMouseButtonUp(0)&&!isAllow)
            {

GameObject.Find("EventSystem").GetComponent<Message>().Show("Недостатньо
простору для розміщення будівлі", 3);
            }

if(isCreating&&placingBuilding&&Input.GetMouseButtonUp(0)&&isAllow)
            {
                //Підтвердити розміщення
                if(!isEnoughGoods())
                {
                    //Відхилення при нестачі ресурсів

GameObject.Find("EventSystem").GetComponent<Message>().Show("Недостатньо
матеріалів для розміщення будівлі", 3);
                    return;
                }

GameObject.Find("EventSystem").GetComponent<Wallet>().LoseGoods(buildPrefs
[buildChooser.current].price);
                GameObject                    building                  =
Instantiate(buildPrefs[buildChooser.current].clasic,
placingBuilding.transform.position, placingBuilding.transform.rotation);

GameObject.Find("FogWar").GetComponent<csFogWar>().AddFogRevealer(building
.transform, 50, true);
                Destroy(placingBuilding);
                ownBuildings.Add(building);
                //Розмістити поле біля нової будівлі
                if(building.CompareTag("Base"))
                {
                    fields.Add(Instantiate(fieldPref,
building.transform.position, building.transform.rotation));
                    Sizing sizing = building.GetComponent<Sizing>();
                    fields[^1].transform.localScale         =         new
UnityEngine.Vector3(sizing.size.x+fieldWidth,                        0.5f,
sizing.size.z+fieldWidth);
                }
            }
        }
        public void StartBuilding()
        {
            // Увімкнення режиму будівництва
            for(int i = 0; i<ownBuildings.Count;i++)
            {

if(ownBuildings[i].CompareTag("Base")||ownBuildings[i].CompareTag("Head"))
                {
                    fields.Add(Instantiate(fieldPref,
```

```
ownBuildings[i].transform.position, ownBuildings[i].transform.rotation));
                    Sizing                    sizing                    =
ownBuildings[i].GetComponent<Sizing>();
                    fields[^1].transform.localScale         =         new
UnityEngine.Vector3(sizing.size.x+fieldWidth,                          0.5f,
sizing.size.z+fieldWidth);
                }
            }
            isCreating = true;
            buildChooser.ActivateMenu(true);
        }
        public void StopBuilding()
        {
            // Вимкнення режиму будівництва
            isCreating = false;
            foreach(GameObject el in fields)
            {
                Destroy(el);
            }
            fields.Clear();
            Destroy(placingBuilding);
            curRotation = Quaternion.identity;
            buildChooser.ActivateMenu(false);
        }

        [System.Serializable]
        public class BuildPrefs
        {
            public String typeName;
            public GameObject clasic;
            public GameObject allow;
            public GameObject deny;
            public Wallet.Reciept price;
            public   BuildPrefs(String   typeName,   GameObject   clasic,
GameObject allow, GameObject deny, Wallet.Reciept price)
            {
                this.typeName = typeName;
                this.clasic = clasic;
                this.allow = allow;
                this.deny = deny;
                this.price = price;
            }
        }
        public bool isEnoughGoods()
        {
            return
GameObject.Find("EventSystem").GetComponent<Wallet>().isEnough(buildPrefs[
buildChooser.current].price);
        }
    }
```

Лістинг 4 – Клас Spawner

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

```
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.AI;

public class Spawner : MonoBehaviour
{
    public GameObject platform;
    private UnityEngine.Vector3 place;
    public List<GameObject> AllTech;
    [NonSerialized] public List<GameObject> ownTech = new
List<GameObject>();
    public GameObject rd, ld;
    public float diapazone;
    public float count;
    [NonSerialized] public Coroutine spawnProcess;
    public csFogWar fogWar;

    void Start()
    {
        place = platform.transform.position;
        fogWar = GameObject.Find("FogWar").GetComponent<csFogWar>();
    }

    void Update()
    {
        if(ownTech.Count!=0)
        {
            ownTech.RemoveAll(item => item == null);
        }
        if(ownTech.Count<count&&spawnProcess==null)
        {
            spawnProcess = StartCoroutine(Spawning());
        }
    }

    IEnumerator Spawning()
    {
        //Заспавнити транспорт
        yield return new WaitForSeconds(diapazone);
        GameObject tech =
Instantiate(AllTech[UnityEngine.Random.Range(0,AllTech.Count)],
transform.position, transform.rotation);
        ownTech.Add(tech);
        if(fogWar&&gameObject.layer == 6)
            fogWar.AddFogRevealer(tech.transform,
Mathf.RoundToInt(tech.GetComponent<Shooting>().shootingDistance+5), true);
        if(gameObject.layer == 8)
            tech.GetComponent<EnemyMoving>().home = transform;
        SetFightable(tech,false);
        StartCoroutine(MoveToPoint(tech, new
UnityEngine.Vector3(place.x+UnityEngine.Random.Range(-2.9f,3f),0,
place.z+UnityEngine.Random.Range(-2.9f,3f)), 20));
        spawnProcess = null;
    }

    IEnumerator MoveToPoint(GameObject obj, UnityEngine.Vector3
point, float speed)
    {
```

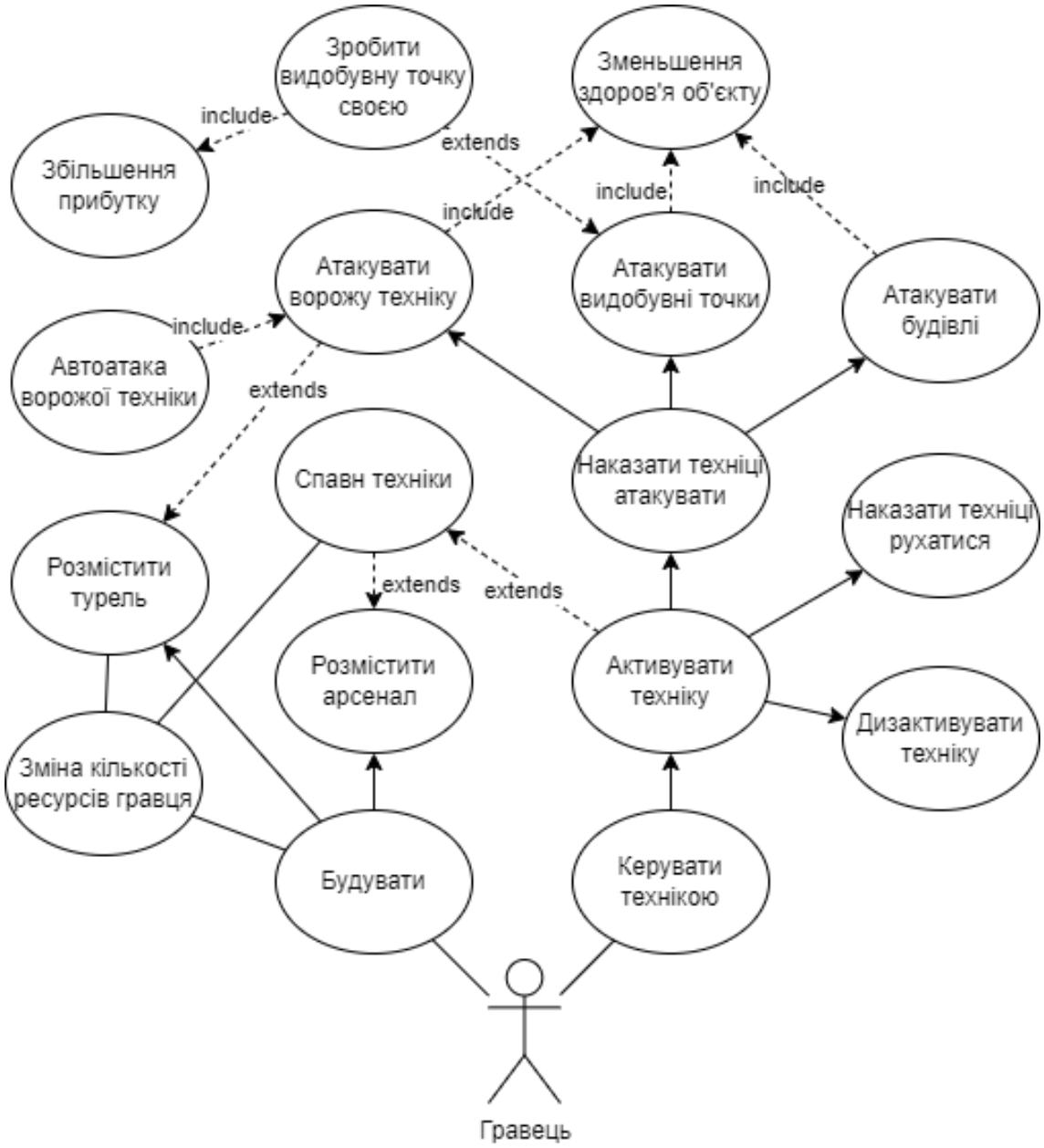
```
            //Вивезти од.тр. з амбару
            UnityEngine.Vector3 relativePos = point -
obj.transform.position;
            Quaternion rotation = Quaternion.LookRotation(relativePos,
Vector3.up);
            obj.transform.rotation = rotation;
            StartCoroutine(OpenDoors(rd,ld,true));
            while(obj.transform.position!=point)
            {
                obj.transform.position =
UnityEngine.Vector3.MoveTowards(obj.transform.position, point,
speed*Time.deltaTime);
                yield return new WaitForSeconds(0.01f);
            }
            SetFightable(obj,true);
            StartCoroutine(OpenDoors(rd,ld,false));
        }
        IEnumerator OpenDoors(GameObject rd, GameObject ld, bool status)
        {
            //Відкрити/Закрити двері
            if(status)
            {

while(transform.InverseTransformPoint(rd.transform.position).x > -2.5)
                {
                    rd.transform.Translate(new UnityEngine.Vector3(-
1f,0f,0f)*20*Time.deltaTime);
                    ld.transform.Translate(new
UnityEngine.Vector3(1f,0f,0f)*20*Time.deltaTime);
                    yield return new WaitForSeconds(0.01f);
                }
            }
            else
            {

while(transform.InverseTransformPoint(rd.transform.position).x < -1)
                {
                    rd.transform.Translate(new
UnityEngine.Vector3(1f,0f,0f)*20*Time.deltaTime);
                    ld.transform.Translate(new UnityEngine.Vector3(-
1f,0f,0f)*20*Time.deltaTime);
                    yield return new WaitForSeconds(0.01f);
                }
            }
        }
        public static void SetFightable(GameObject tech, bool status)
        {
            //Ввімкнути/Вимкнути боєздатність од.тр.
            if(!status)
            {
                tech.GetComponent<NavMeshAgent>().enabled = false;
                tech.GetComponent<Collider>().enabled = false;
                tech.GetComponent<Shooting>().enabled = false;
                if(tech.GetComponent<Triggering>())
                    tech.GetComponent<Triggering>().enabled = false;
                if(tech.GetComponent<EnemyTriggering>())
                    tech.GetComponent<EnemyTriggering>().enabled =
false;
```

```
                tech.GetComponent<HealthManager>().enabled = false;
            }
            else{
                tech.GetComponent<Collider>().enabled = true;
                tech.GetComponent<Shooting>().enabled = true;
                if(tech.GetComponent<Triggering>())
                    tech.GetComponent<Triggering>().enabled = true;
                if(tech.GetComponent<EnemyTriggering>())
                    tech.GetComponent<EnemyTriggering>().enabled = true;
                tech.GetComponent<HealthManager>().enabled = true;
                tech.GetComponent<NavMeshAgent>().enabled = true;
            }
        }
    }
```

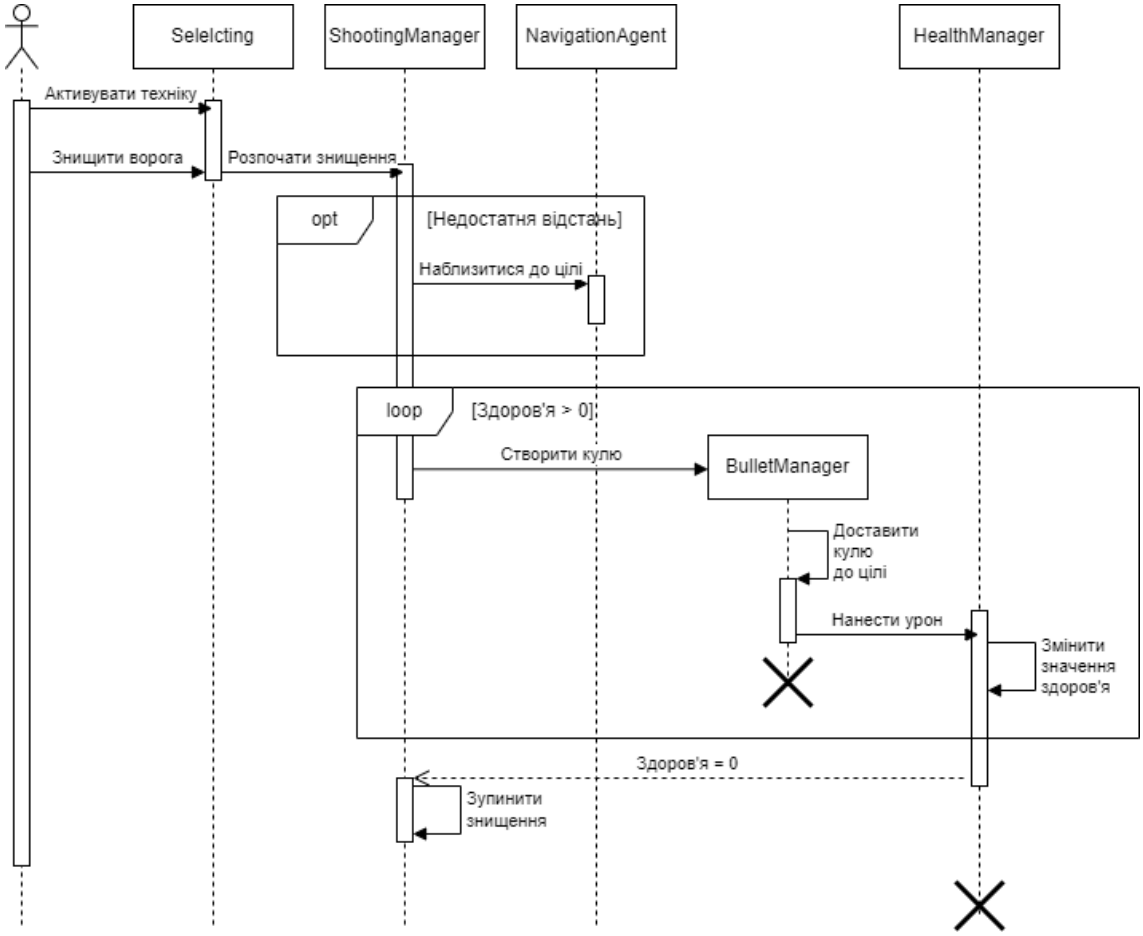
ДОДАТОК Б

Діаграми UML

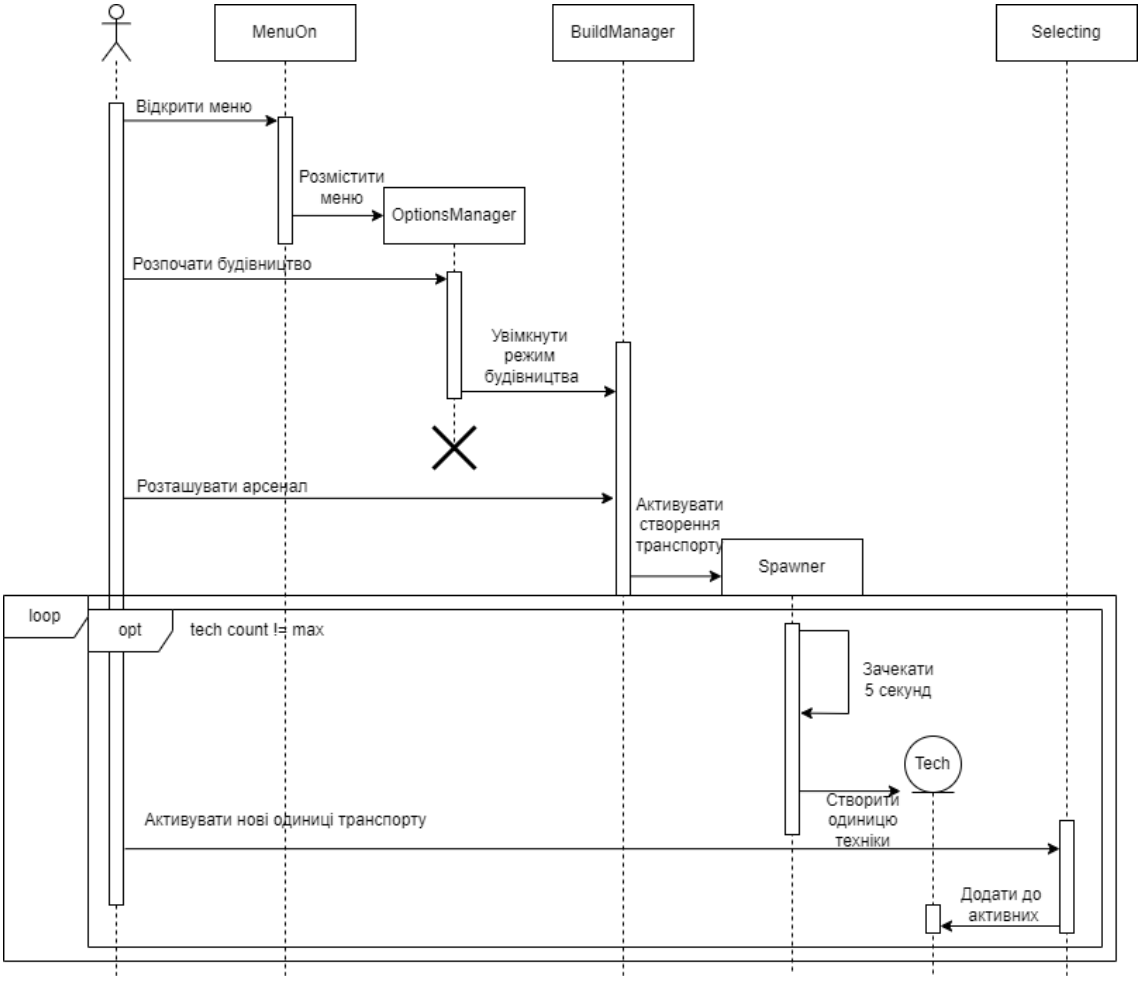
Діаграма 1 - Діаграма варіантів використання



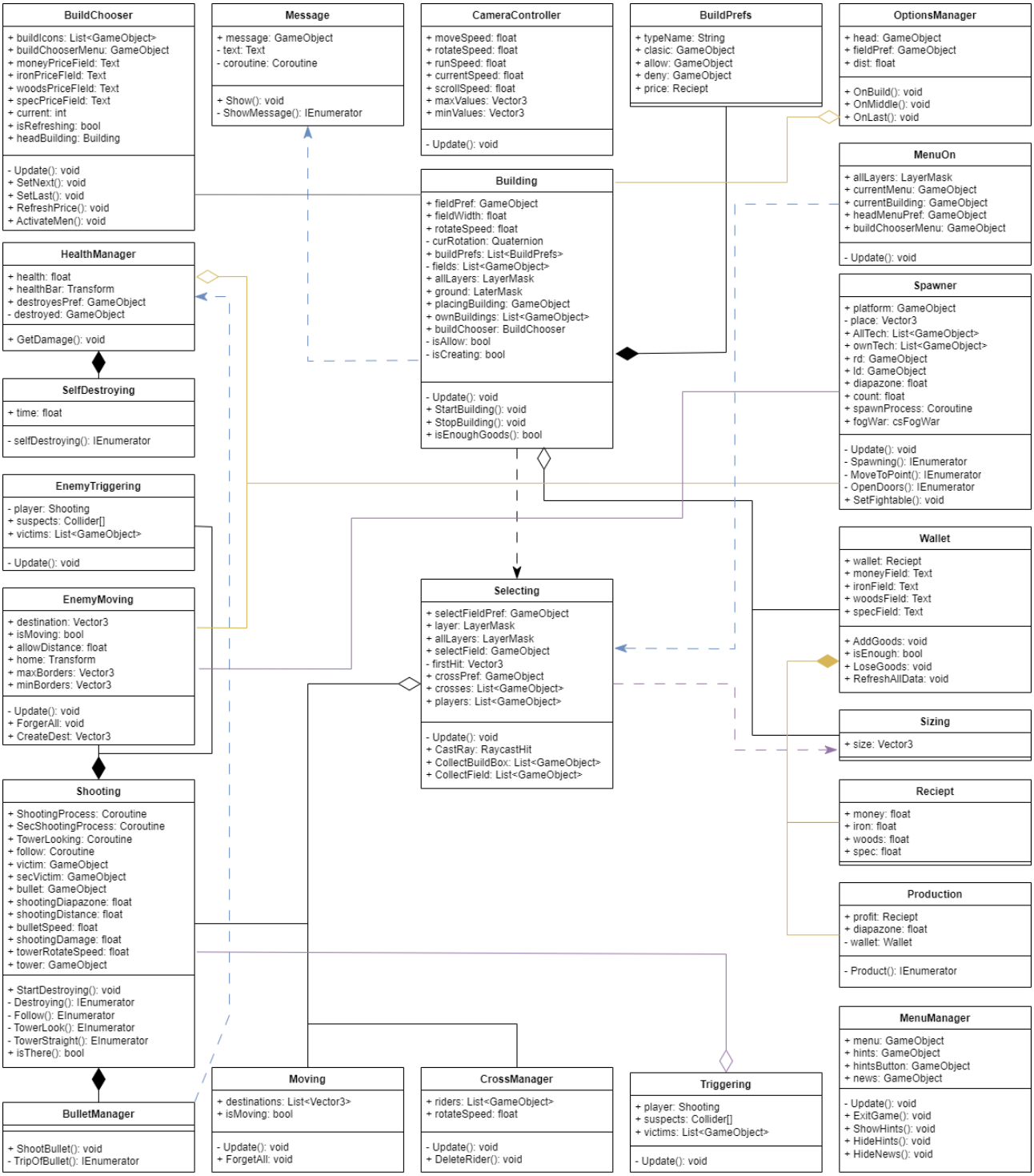
Діаграма 2 – Діаграма послідовностей 1



Діаграма 3 – Діаграма послідовностей 2



Діаграма 4 – Діаграма класів



ДОДАТОК В

Знімки екрану при процесі гри

Рисунок 1 – Пересування транспорту по мапі

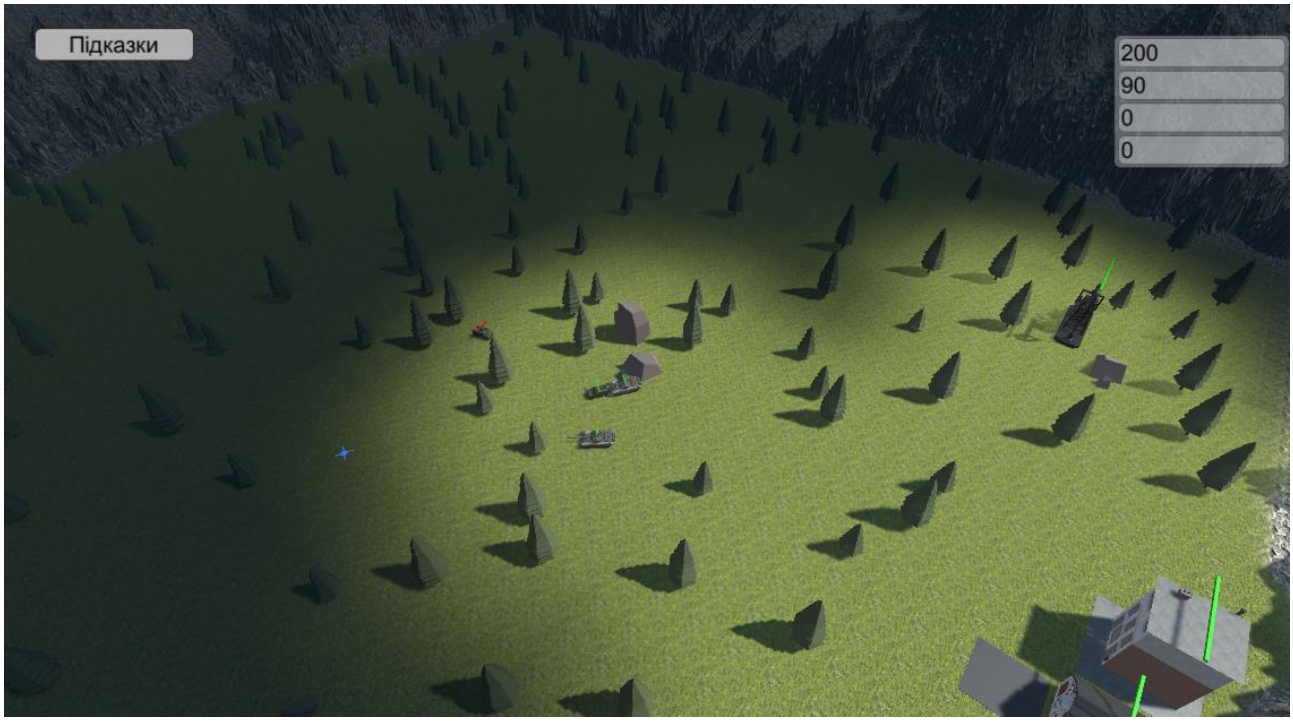


Рисунок 2 – Режим будівництва споруд

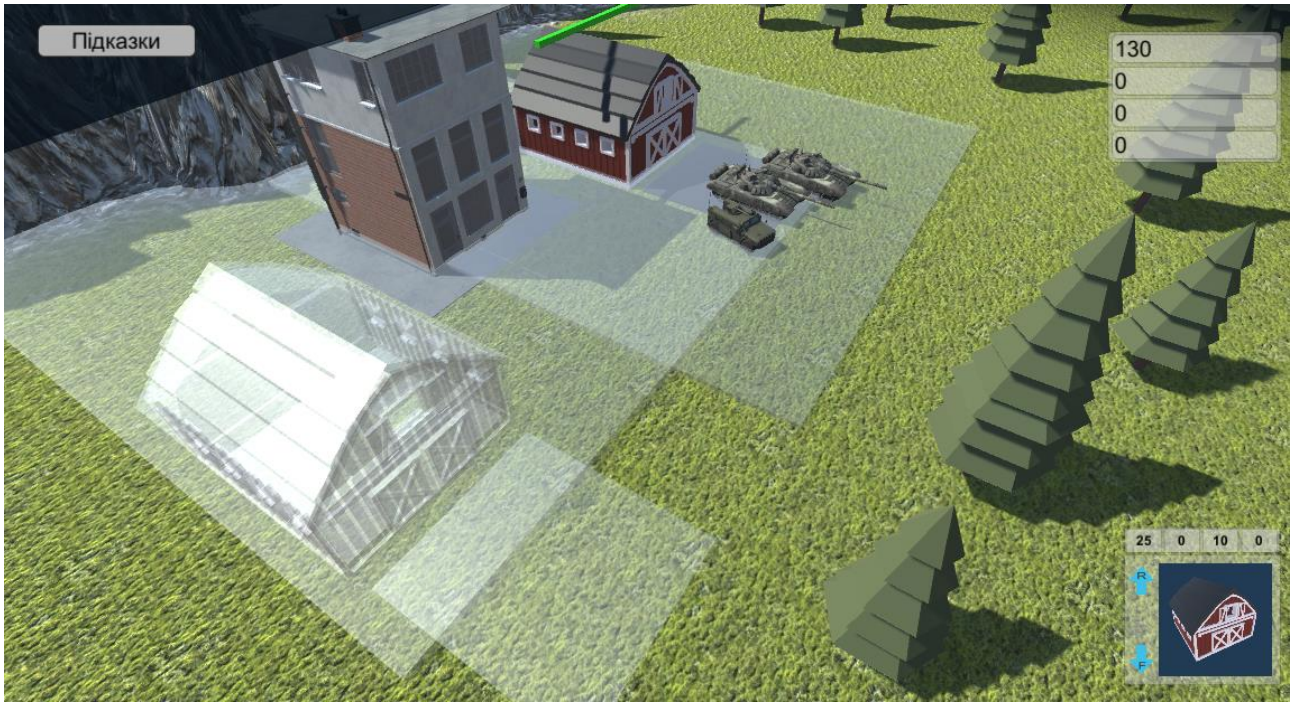


Рисунок 3 – Процес стрільби по ворожих видобувних точках

Рисунок 4 – Процес перестрілки із ворожою технікою

ДОДАТОК Г

Диск із кваліфікаційною роботою бакалавра