

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Гра стратегія з елементами шутера з використанням
нейромережі для симуляції поведінки персонажів»**

Виконав студент групи ІПЗм-22-2
БОРАЧУК Анатолій Григорович

Керівник роботи:
ПАНАРІНА Ірина Володимирівна

Консультант:
ЛЕВКІВСЬКИЙ Віталій Леонідович

Рецензент:
ВАКАЛЮК Тетяна Анатоліївна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БОРАЧУК** Анатолій Григорович

Керівник роботи: **ПАНАРІНА** Ірина Володимирівна

Тема роботи: **«Гра стратегія з елементами шутера з використанням нейромережі для симуляції поведінки персонажів»**,

затверджена Наказом закладу вищої освіти від **«18» вересня 2023 р. 479/с**

Вихідні дані для роботи: об'єктом дослідження є історія жанру стратегія, шутер; алгоритми та технології для розробки комп'ютерних ігор; моделі нейромереж. Предметом дослідження є розробка нейромережі, розробка 3D моделей, їх подальша анімація, реалізація власного програмного забезпечення.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	ЛЕВКІВСЬКИЙ Віталій Леонідович	22.09.2023р.	22.09.2023р.
2	ЛЕВКІВСЬКИЙ Віталій Леонідович	22.09.2023р.	22.09.2023р.
3	ЛЕВКІВСЬКИЙ Віталій Леонідович	22.09.2023р.	22.09.2023р.

Календарний план

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналізування вимог до програмного продукту	до 25.09.2023	виконано
2	Пошук, огляд та аналіз аналогічних розробок	до 01.10.2023	виконано
3	Планування структури програми	до 15.10.2023	виконано
4	Дизайн документ	до 25.10.2023	виконано
5	Реалізація алгоритмів роботи гри	до 10.11.2023	виконано
6	Написання програмного коду	до 25.11.2023	виконано
7	Тестування роботи програми	до 01.12.2023	виконано
8	Оформлення пояснювальної записки	до 10.12.2023	виконано
9	Захист	20.12.2023	

Студент Анатолій БОРАЧУК

Керівник Ірина ПАНАРІНА

РЕФЕРАТ

Кваліфікаційна робота складається з пояснювальної записки та програмного продукту - гри. Пояснювальна записка до кваліфікаційної роботи містить 82 сторінки, 61 ілюстрацію та 1 таблицю.

Метою роботи є розробка гри стратегії з елементами шутера від першої особи за допомогою рушія Unity3D.

Під час розробки проекту розглянуто та проаналізовано інші ігрові розробки в жанрі Action та обґрунтовано вибір кожного рішення реалізації. Спроектowana структура програмної частини, наведені сценарії роботи, створений весь графічний контент для гри, проведено програмування ігрових елементів, проведено тестування ігрової розробки.

КЛЮЧОВІ СЛОВА: ГРА, ШУТЕР, СТРАТЕГІЯ, ІГРОВИЙ РУШІЙ, UNITY3D, НЕЙРОМЕРЕЖА.

ABSTRACT

The graduation work consists of an explanatory note and the software complex of the game. The explanatory note to the graduation thesis contains 82 pages, 61 pages and 1 table.

The method of operation is the development of a strategy game with elements of a first-person shooter using the Unity3D engine.

During the development of the project, other game developments in the Action genre are considered and analyzed, and the choice of each implementation solution is justified. The structure of the software part was designed, the work scenarios were given, all the graphic content for the game was created, and the game development was tested.

KEYWORDS: GAME, SHOOTER, STRATEGY, GAME ENGINE, UNITY3D, NEURAL NETWORK.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ІГРОВИХ ДОДАТКІВ	9
1.1. Використання нейромереж при створенні ігор	9
1.2. Існуючі аналоги програмного продукту	11
1.3. Постановка задачі	20
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.	21
Висновки до першого розділу	22
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ДОДАТКУ	23
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.	23
2.1.1. Діаграма варіантів використання	23
2.1.2. Діаграма класів	26
2.2. Дизайн документ	31
2.2.1. Геймплей	31
2.2.4. Локації / сцени	32
2.2.5. Текстури, персонажі та різні гейм-об'єкти	33
2.2.6. Керування	39
2.3. Реалізація алгоритмів роботи гри	40
Висновки до другого розділу	45
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ	46
3.1. Інтерфейс та порядок роботи з програмним додатком.	46
3.2. Тестування роботи програмного комплексу.	52
Висновки до третього розділу	56
ВИСНОВКИ	57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	58
ДОДАТКИ	60
	6

ВСТУП

Актуальність теми. Актуальність теми розробки ігор обумовлена тим, що ігри є одним з найбільш популярних видів розваг у світі. Вони приносять задоволення людям різного віку та з різними інтересами. Ігри можуть бути навчальними, розвиваючими, а також просто розважальними.

Розвиток технологій також сприяє розробці ігор. Завдяки новим технологіям розробники можуть створювати все більш реалістичні та захоплюючі ігри. На сьогодні є продукти, які використовують, наприклад, нейромережі. Таке нововведення дозволяє зробити гру ще цікавішою та непередбачуваною, адже немає конкретної послідовності дій, до якої звикає гравець. Це стимулює зростання попиту на ігри та створює нові можливості для розробників.

Економічна ефективність розробки ігор також є важливим фактором, що сприяє її актуальності. При відносно невеликих початкових інвестиціях розробники можуть отримувати значні прибутки від успішних проєктів. Це приваблює до сфери розробки ігор талановитих людей, які прагнуть досягти успіху та забезпечити собі фінансову стабільність.

Мета дослідження. Створення гри стратегії з елементами шутера та використанням нейромережі, здобуття та закріплення знань з ігрової розробки, 3D-анімації, 3D-моделювання, графіки, побудови архітектури коду, реалізація та навчання нейромережі.

Для досягнення поставленої мети необхідно вирішити такі основні завдання:

- Аналіз інформації по реалізації нейромережі та методів її навчання;
- Аналіз існуючих часткових аналогів продукту, що розробляється;
- Вибір та дослідження середовища розробки програмного продукту;
- Визначення основних аспектів геймплею;
- Підготовка візуальної складової гри (3D-об'єкти, текстури, анімації, ландшафт);

- Реалізація логіки застосунку;
- Тестування продукту.

Об'єктом дослідження є історія жанрів стратегія, шутер; алгоритми та технології для розробки комп'ютерних ігор; моделі нейромереж.

Предметом дослідження є розробка нейромережі, розробка 3D моделей, їх подальша анімація, розробка інтерфейсу, реалізація власного програмного забезпечення та тестування застосунку.

Методи аналізу. Під час аналізу інформаційних джерел в процесі дослідження було використано абстрагування, аналіз та формалізацію отриманих результатів.

Інформаційна база дослідження складається з: літературних джерел з питань розробки ігор, нейромереж та ШІ; статті та доповіді з результатами експериментальних досліджень використання нейромереж для симуляції поведінки.

Наукова новизна дослідження полягає в розробці нового підходу до розробки, який базується на використанні нейромереж та поєднанні декількох жанрів відеоігр.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ІГРОВИХ ДОДАТКІВ

1.1. Використання нейромереж при створенні ігор

Нейромережі - це моделі машинного навчання, які імітують роботу людського мозку. Вони складаються з вузлів, які пов'язані між собою та називаються **нейронами**. Кожний нейрон отримує вхідні сигнали від інших нейронів і обробляє їх, використовуючи певну функцію активації. Результат обробки надсилається на інші нейрони.

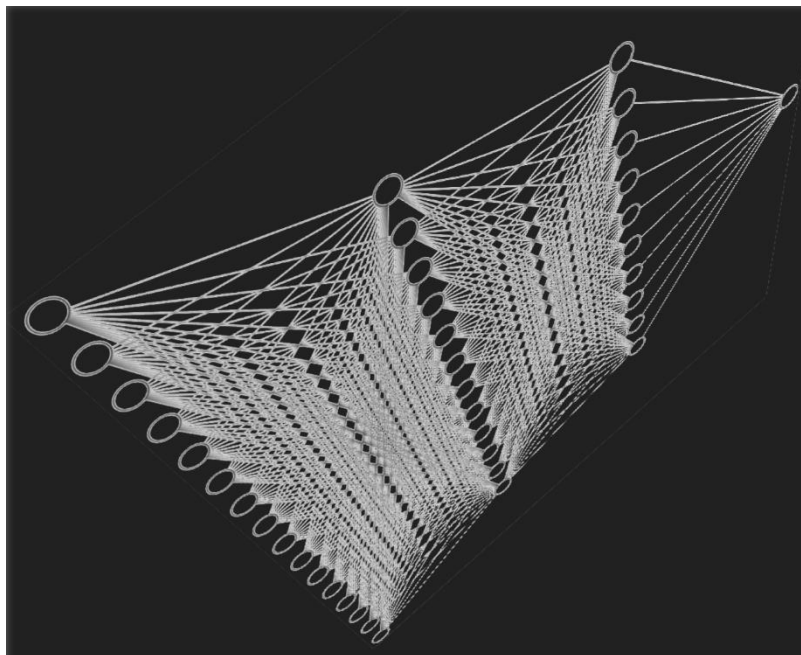


Рис. 1.1 – Структура зв'язків нейромережі

Існує два основних типи навчання нейронних мереж:

- 1. Навчання з учителем** - це тип навчання, при якому нейронна мережа отримує набір навчальних даних, який містить правильні відповіді. Нейронна мережа використовує ці дані, щоб навчитися відтворювати правильні відповіді на нові дані.
- 2. Навчання без учителя** - це тип навчання, при якому нейронна мережа не отримує набір навчальних даних з правильними відповідями. Нейронна мережа вчиться, аналізуючи дані та знаходячи закономірності.

в них. Оскільки гра буде динамічна і не завжди один і той же набір даних буде правильним, тому цей варіант підходить найкраще.

Існує багато різних типів нейронних мереж, які розрізняються за своєю структурою, функцією активації та типом навчання. Тому після аналізу декількох з них було обрано мережу прямого поширення.

Нейромережа прямого поширення (НПП) - це тип нейронної мережі, яка використовується для навчання з учителем. Вона складається з трьох шарів нейронів: вхідного, прихованого та вихідного. Є хорошим вибором для реалізації гри, оскільки вона є відносно простою в реалізації і забезпечить нас необхідним функціоналом. Наприклад, нейромережу можна навчити відтворювати правильні ходи в шахах або стрілялці.

Вхідний шар нейронів отримує вхідні дані. На нього ми будемо відправляти дані, отримані з «сенсорів» персонажу, оточуючого світу, кількості очків здоров'я і т.п. Прихований шар нейронів обробляє вхідні дані, використовуючи функцію активації. Вихідний шар нейронів генерує вихідні дані, які будуть керувати певним об'єктом.

В іграх нейромережі можуть використовуватися для створення штучного інтелекту (ШІ). Він на основі нейромереж може приймати більш розумні рішення, ніж традиційний ШІ, що може призвести до більш захоплюючого та конкурентоспроможного геймплею. Ось кілька конкретних прикладів використання нейромереж у стратегіях:

Розробка тактики. Нейромережі можуть використовуватися для навчання ШІ розробляти ефективні тактики в бою. Наприклад, нейромережа може навчитися визначати найкраще розташування військ, вибирати найкращі атаки та уникати небезпек.

Розподіл ресурсів. Нейромережі можуть використовуватися для навчання ШІ ефективно розподіляти ресурси. Наприклад, нейромережа може навчитися визначати, які ресурси найважливіші для перемоги, і як найкраще їх використовувати.

Створення рівнів. Нейромережі можуть використовуватися для створення процедурних рівнів, які будуть більш складними та цікавими для гравців. Наприклад, нейромережа може навчитися створювати рівні з різноманітними ландшафтами, ворожими силами та завданнями.

1.2. Існуючі аналоги програмного продукту

1.2.1. Total Conflict: Resistance

Деякі з запланованих стратегічних механік можна побачити в грі “Total Conflict: Resistance”. Це гра, в якій змішані жанри глобальної стратегії, тактики та екшена від першої особи. Вибір ігрових режимів невеликий (2 кампанії, та можливість зіграти «швидку» битву; гра ще в розробці):

- Глобальна кампанія
- Тактична кампанія
- Швидка битва

Тактична кампанія

Цей режим дає змогу сфокусуватися лише на бойовому процесі: немає держав, економіки, логістики і т.д. Доступне дерево розвитку, де одразу відкриваються цілі підрозділи різного призначення та бронетехніка.

Генерується випадкова послідовність сценаріїв (атака, оборона, дуель), яку гравець має пройти. Перед кожною битвою можна налаштувати послідовність куплених підрозділів та оновити озброєння, особовий склад. Після бою гравець отримує трофеї, які можна використовувати.

Глобальна кампанія

Гравець обирає одну з запропонованих держав, якою буде керувати (див. рис. 1.2). Спочатку між віртуальними країнами немає конфліктів. Це відображається на екрані дипломатії (див. рис. 1.3). Потрібно дбати про свої землі та армію, але не забувати що деякі дії можуть вплинути на відносини з іншими країнами. Наприклад, якщо нарощувати виробництво лише військової техніки – сусідам це може не сподобатись.



Рис. 1.2 – Екран вибору країни

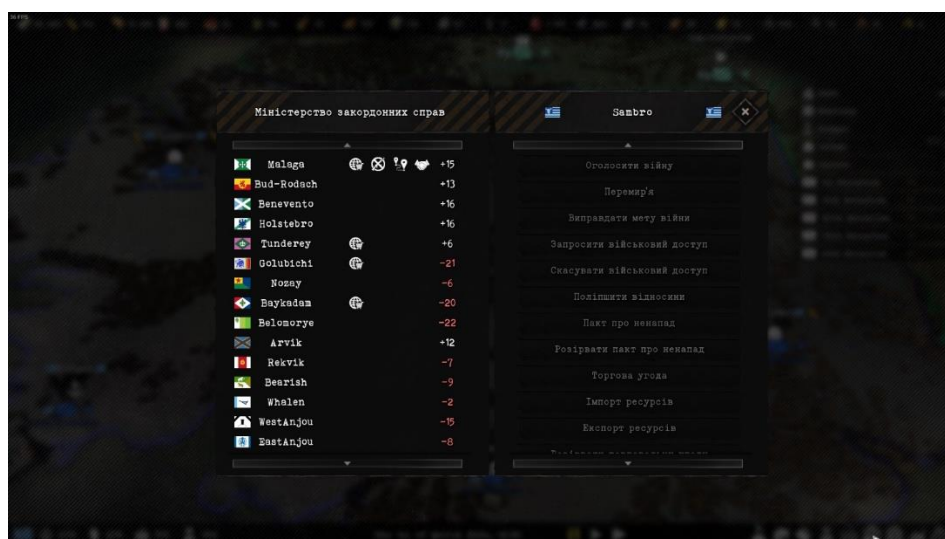


Рис. 1.3 – Екран дипломатії

В будь-якому із своїх міст можна щось побудувати, якщо населений пункт задовольняє вимоги для цього. Наприклад, для побудови шахти треба якась руда поблизу. Також місця для забудови обмежені, і в кожному місті їх може бути різна кількість. Тому потрібно раціонально використовувати ці місця, щоб мати максимальну вигоду. Після початку будівництва має пройти певний час, перш ніж будівля почне функціонувати.

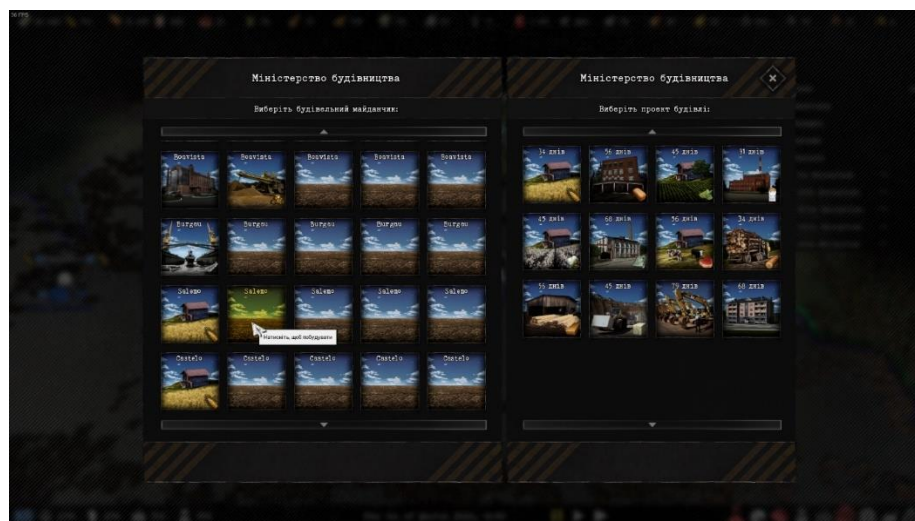


Рис. 1.4 – Побудова

Для створення нових батальйонів потрібно в містах, де є цивільне населення, і яке придатне до служби, обрати тип: поліція, регулярна армія, спецназ і т.п.. Населення - це окремий ресурс, який можна отримати будуючи житлові будинки в місцях забудови. Звичайних резервістів недостатньо для повноцінного загону: потрібна спеціальна уніформа (залежить від типу загону) і відповідне озброєння у вигляді бронетехніки, яка є в межах міста, снарядів та топлива до неї; зброї, набоїв, модифікацій, їжі (консерви, хліб) і аптечки – для піхоти.



Рис. 1.5 – Створення поліцейського підрозділу

Озброєння можна добувати декількома шляхами:

- Відкрити технологію (див рис. 1.6) та зібрати на військовому заводі, який треба будувати. Зазвичай, він доступний одразу при старті гри, але якщо немає - для цього потрібні ресурси (дерево, камінь і гроші). На відкриття чогось нового потрібен час: від 15 до 170 ігрових днів. Найдовше розробляються танки, літаки та уніформа 3-го ступеню: від 120 днів.
- Купити на ринку. Товари на ринку з'являються після заключення товарної угоди з союзною державою. З часом на ринку стає дедалі більше пропозиції, хоч на початку гри вибір дуже обмежений.



Рис. 1.6 – Екран технологій

Як тільки гравець вважає, що військо готове та надійно екіпіроване, можна, наприклад, в межах держави боротися з озброєними терористами. Бій можна пропустити або ж керувати в режимі реального часу.

Бійці, які знаходяться під контролем гравця можна переміщати на різні позиції (див. рис. 1.11), віддавати різного типу накази (зміна положення, дозволити/заборонити вогонь) або ж взяти під пряме керування одного із них (див. рис. 1.12). Бій триває доки одна із сторін не почне здаватись або ж зовсім

втратить боєздатність. Однак щоб змусити противника здатись потрібна велика перевага в особовому складі та техніці.

Складнощі стає більше, коли бій в самому розпалі: потрібно звертати увагу на загальний стан справ, швидко реагувати на зміну ситуації по всій ширині сутечки, також важливо не забувати про моральний стан взводів та про підкріплення, яке з'являється на відповідному краю ігрової зони, при умові що ще є боєздатні підрозділи в батальйоні.



Рис. 1.7 – Стратегічне керування

Гра демонструє хорошу картинку і від першої особи. Гравець може керувати не тільки персонажем, а й всією наявною в грі, військовою технікою окрім літаків та кораблів.

Фізика польоту куль наближена до реальної і потребує врахування дистанції до цілі та тип зброї при веденні вогню. Важливу роль відіграють модифікації зброї: калібраторні, голографічні та оптичні приціли, оскільки таким чином змінюється максимальна дальність обзору.

Також потрібно слідкувати за боєкомплектom підрозділів, тому що вся вогнепальна зброя витрачає кулі, бронетехніка – снаряди, системи залпового вогню – ракети і т.д. Якщо про це не подбати техніка виходить в бій з порожнім боєкомплектom, і тому користі від неї не багато: лише транспортування.

Конкретних цілей в грі немає. Можна розвиватись дружелюбно або ж агресивно, що моментально псує відносини з іншими країнами. Для новачків гра

складніша, тому що багато механік, які потребують постійної уваги. Особливо під час налаштування шляхів логістики, щоб всі ігрові персонажі отримали необхідне озброєння, а підприємства – ресурси.

1.2.2. Mount and Blade : Warband

Mount and Blade - комп'ютерна гра в жанрі Action/RPG з відкритим світом. Всі ігрові події відбуваються на вигаданому континенті Кальрадія (рис. 1. 8), що нагадує середньовічну Європу. Є декілька королівств, які розпочнуть війну, або ж навпаки - створять союз. Все залежить від стосунків між ними.

Як такого сюжету немає: гравець отримує завдання від інших ігрових персонажів (за наявності бажання). Велика частина геймплею побудована на: вміннях та здібностях героя; чисельності та майстерності його армії; земельних володіннях; стосунках між персонажами/країнами. На відміну від більшості ігор схожої тематики, тут немає магії, вигаданих створінь чи будь-яких інших фентезійних елементів.



Рис. 1.8 – «Кальрадія»

Розпочинається ігровий процес з редактору лиця героя. Набір параметрів великий, тому це перетворюється на цікавий процес. Одразу як картинка відповідає задуму можна перейти до наступного етапу: базовий розподіл балів покращень. Гравець одразу отримує на кожну категорію (атрибути, здібності,

володіння зброєю) по незначній кількості балів. Під час гри їх можна отримати за: виконані завдання, тренування в спеціальних лагерах, підвищення рівня героя.

Кожен із атрибутів відкриває певні можливості: сила, інтелект, харизма впливають на максимальний рівень здібностей (не вище 10-го). Здібності, в свою чергу впливають на ведення справ та бою в цілому. Наприклад, "лідерство" збільшує максимальний розмір армії, "торгівля" - допомагає швидше визначати вигідну торгову угоду, "хірургія" - швидше лікує солдат і т.д. Володіння зброєю відповідає за швидкість, силу, точність удару/пострілу. Кожну категорію зброї потрібно покращувати окремо. Всього їх є 6: одноручна, дворучна, деревкова, лук, арбалет, метальна.



Рис. 1.9 – Характеристики персонажу

Важливу роль відіграють подорожі та різних масштабів битви, в яких гравець теж приймає участь, як командир війська. Спочатку під контролем немає бійців, тому доведеться збирати армію: в селах – звичайні рекрути, а в тавернах, з певною ймовірністю, будуть найманці з різним озброєнням, які за гроші приєднуються. На шляху до населених пунктів часто можна зустріти грабіжників чи піратів. Їхня реакція залежить від слави та армії гравця: втікатимуть якщо точно програють або ж нападуть, якщо перевага в них.

Рекрути мають різне походження і відповідні гілки покращень. На перших рівнях покращень вони не найкращі воїни, але з кожним боєм та тренуванням

отримують досвід, який дозволяє перейти на наступний рівень. Таким чином покращується не все військо, а лише один персонаж, який накопичив достатньо досвіду. В кожного королівства свої унікальні війська, які спеціалізуються на різних здібностях. Наприклад, Норди – сильна піхота, але немає лучників; Сवादія – арбалетчики та кіннота; Хергітське ханство – лучники верхом на конях і т.д.

Важливо слідкувати за моральним станом війська. Чим він нижче – тим більше людей втрачатиме гравець. Цей параметр залежить від:

1. Кількості їжі в інвентарі армії, при чому кожен тип продукту по-різному впливає: капуста – 2, мед – 8, хліб – 4, м'ясо – 4, вино – 6, сир – 7.
2. Розмір армії: чим більша армія – тим тяжче підтримувати дух бійців
3. Останній події: перемога, поразка, підлий поступок гравця чи навпаки – благодійність.



Рис. 1.10 – Свадійські рекрути, покращені до кінноти

В бою можна керувати своєю армією: переміщення, вибір типу атаки, відкрити/припинити вогонь. Бойова система дозволяє гравцю самому вибирати напрямок атаки або блоку. Доступна велика кількість холодної зброї: від книжалів до дворучних мечів, списи, луки та арбалети, щити, металеві ножі, сокири та інша характерна зброя для Середньовіччя. Функції командування можна викликати на гарячі клавіші F1-F8. Однак, якщо в бою гравець не сам, а, наприклад, з союзним лордом, то командувати його військами не вийде. За виключенням, якщо гравець досягнув певних успіхів на службі у будь-якого короля і отримав статус головнокомандувача армії.

Чи вступати на службу чи ні – вирішує тільки гравець. Але щоб отримати офіційне запрошення від короля треба виграти декілька десятків битви та турнірів. Таким чином параметр слави зростає і допоки гравець не стане вассалом – будуть пропозиції від різних королівств.

Турніри – цікава механіка для заробітку ігрової валюти. Вони проводяться в великих містах, під час святкування. Вступає певна кількість NPC та гравець. Вся кількість учасників ділиться так, щоб було 6 етапів. Перед кожним етапом можна поставити ставку на себе і в разі перемоги виграти хороший приз, який дуже корисний на початкових етапах гри. Також перемогу можна присвятити комусь, що покращує стосунки.



Рис. 1.11 – Турнір

Якщо не вступати на службу і завоювати собі землі, тоді утвориться нове королівство, яке ще ніхто не визнає. В такому випадку потрібно збирати з таверн різних міст людей, які готові бути вассалами. Їм можна передати будь-яку екіпіровку, війська чи навіть покращити характеристики (див. рис. 1.9), звісно за наявності балів покращення. Окремою фігурою на карті вони стають лише після передачі їм земельних володінь. Кожен такий подарунок впливає на відносини з усіма підлеглими, що може привести до заколоту, особливо якщо не радитись з жодним вассалом. Чітко вказаних цілей немає. Завдання можна отримати всюди: від супроводу каравану або худоби до доставки листа від короля певному персонажу чи розвідки у військовому поході. Гру завершити можна в будь-який

час, перейшовши в меню лагера та обрати пункт «Завершити історію». Тоді буде відображена статистика гравця та невеличка історія як герой живе далі.

1.3. Постановка задачі

Гра буде розроблена в жанрі стратегія та здебільшого матиме відповідні механіки: елементи менеджмента бази, власного війська та його тренування; керування армією під час бою, де також будуть присутні механіки шутера. Гравець матиме можливість власноруч впливати на хід битви, обравши одного із персонажів своєї армії.

Буде лише один режим гри: компанія. Вона матиме глобальну карту, на якій буде декілька різних поселень і дві країни. Гравець має окреме невеличке стартове поселення та групу війська до 10 персонажів. У власному населеному пункті доступні декілька варіантів подій:

1. Тренування (симуляція бою для тренування нейромережі). Відбувається перехід на сцену «Лагер», де маємо меню, в якому налаштовуємо деталі битви. Таким чином тренуємо та оновлюємо дані нейромережі.
2. Створити або додати в існуючий (має бути в межах міста) батальйон підрозділ. Можна активувати один раз в декілька ігрових діб. Повний цикл ігрової доби (24 години) триватиме реальних 12 хвилин. При наймі віднімається відповідна кількість цивільних з населення та необхідні ресурси/екіпірування
3. Побудова. Це розділ меню, де є вибір доступних місць для забудови. В залежності від рівня поселення кількість цих місць може збільшуватись. Наприклад, село матиме 3 місця, місто – 5, столиця – 7 (може бути лише одна). Рівень населеного пункту визначає кількість цивільних в місті. Для цього потрібно будувати їм житлові будинки.
4. Торгівля. Доступно якщо побудований ринок. Можна обмінювати ресурси конкретно з цього поселення на валюту, або ж навпаки – купити. Також

мають бути каравани, які будуть перевозити ресурси (при потребі) та гроші в межах однієї держави.

5. Гарнізон. Список солдатів, які стоять на охороні цього населеного пункту. Можна додавати чи забирати юнітів з гарнізону. Це дає змогу підготуватися до можливих атак ворога.

Мета гри полягає у завоюванні всієї карти. У гравця два опонента, у яких теж є свої населені пункти, армія, ресурси та фінанси.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.

Для розробки гри необхідний ігровий рушій. На сьогодні існують багато відомих рушіїв, таких як: Unity3D, Unreal Engine, CryEngine та багато інших. Більшість з них мають можливість створення гри одразу для декількох платформ (Android, Windows/Linux, iOS і т.д.). Зрозуміло, що кожен з рушіїв має свої переваги та недоліки. Не для всіх рушіїв є детальна документація та хоча б якісь уроки по створенню гри, тому вибір будемо робити між Unity3D та Unreal Engine, які мають величезну кількість матеріалів по розробці ігор (на офіційних сайтах та YouTube)

Unreal Engine чудово опрацьовує картинку, яку бачить користувач. Звісно, це більше реусрозатратно, ніж в Unity, але ця функція вбудована в рушій, на відміну від Unity, де шейдери треба писати власноруч або використовувати AssetStore. Цей рушій підтримує мову програмування C++, або, якщо людина не впевнена в своїх можливостях щодо програмування, присутній візуальний редактор ігрової логіки – Blueprints. Інтерфейс програми сильно відрізняється від Unity і в ньому не так просто розібратись без документації чи туторіалів.

Оскільки Unity3D простіший в багатьох планах, тому будемо використовувати його.

Реалізувати нейромережу можна за допомогою власного коду або ж використовуючи бібліотеки, наприклад: Unity ML-Agents, що використовує PyTorch, Noedify (AssetStore). Спочатку потрібно визначитись з типом

нейромережі, тому що їх існує багато. В даному випадку можна застосувати мережу прямого поширення, оскільки алгоритм не потребує дуже багато ресурсів. Оптимізація – важливе питання, тому що на сцені одночасно може бути велика кількість екземплярів нейромереж. Тренування буде відбуватись епохами - певний відрізок часу. Правильними та не правильними діями мережа змінюватиме ефективність. На основі тих даних генерується наступна епоха.

Тепер необхідно обрати програмне забезпечення для створення графіки 3D та 2D. Найкращим растровим редактором на сьогоднішній день – Adobe Photoshop. Зручний інтерфейс, підтримка графічних планшетів, велика кількість плагінів та вбудованих інструментів тому підтвердження.

Що стосується 3D-графіки, то тут є декілька варіантів: 3Ds Max, Cinema 4D, Belnder3D. Оскільки 3Ds Max та Cinema 4D не безкоштовні тому, оберемо Blender3D, тим паче, що він має теж досить непогані можливості. На результат впливає лише вміння працювати з будь-яким з цих інструментів.

Висновки до першого розділу

Було проаналізовано ігри Total Conflict: Resistance та Mount and Blade: Warband, які частково є аналогами програмного продукту, що розробляється. Також отримано теоретичний фундамент для подальшої роботи і повністю сформовано уявлення про гру, що створюється та нейромережу, яка стане базовою частиною геймплею. Для реалізації штучного інтелекту в грі була обрана нейромережа прямого поширення. НПП є відносно простою в реалізації і забезпечує необхідний функціонал для гри. НПП можна використовувати для навчання штучного інтелекту гравця, агента, який керує грою, або системи, яка оцінює гру.

Також було визначено необхідні програмні оболонки та інструменти для реалізації поставленої задачі, вказані їх переваги. Огляд дозволив виділити платформу Unity для проекту, що створюється та мову програмування C#.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ДОДАТКУ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.

2.1.1. Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) відображає залежність між актором або групою акторів та прецедентами. У ній можна відстежити, які саме дії актора призведуть до виконання одних чи інших процесів. Для розробки Use Case діаграми було визначено одного актора - гравець. На рисунку 2.1 зображено діаграму варіантів використання головного меню.

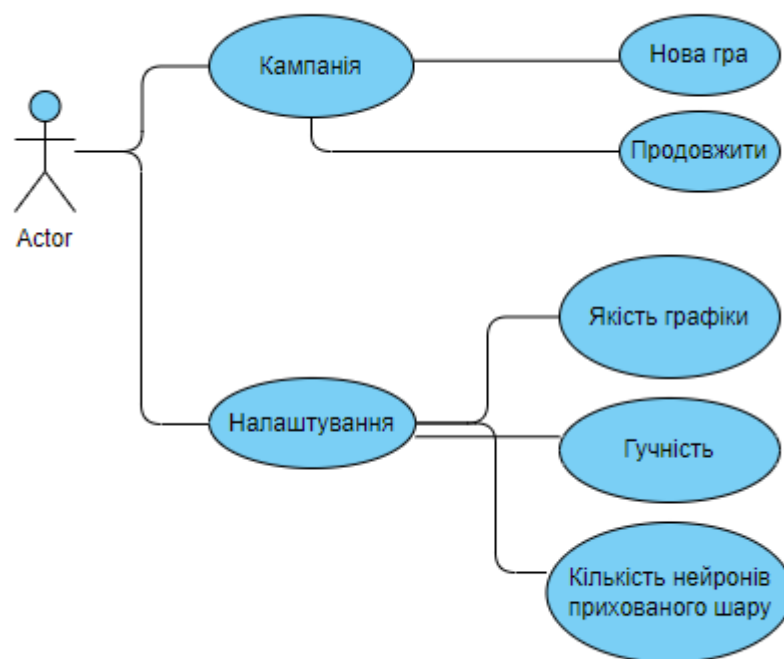


Рис. 2.1 – Діаграма варіантів використання головного меню

На основі сформованих даних про акторів та всіх можливих варіантів використання ігрового додатку була розроблена діаграма варіантів використання глобальної мапи (див. рис. 2.1), бій (рис. 2.2), тренування (рис. 2.3).

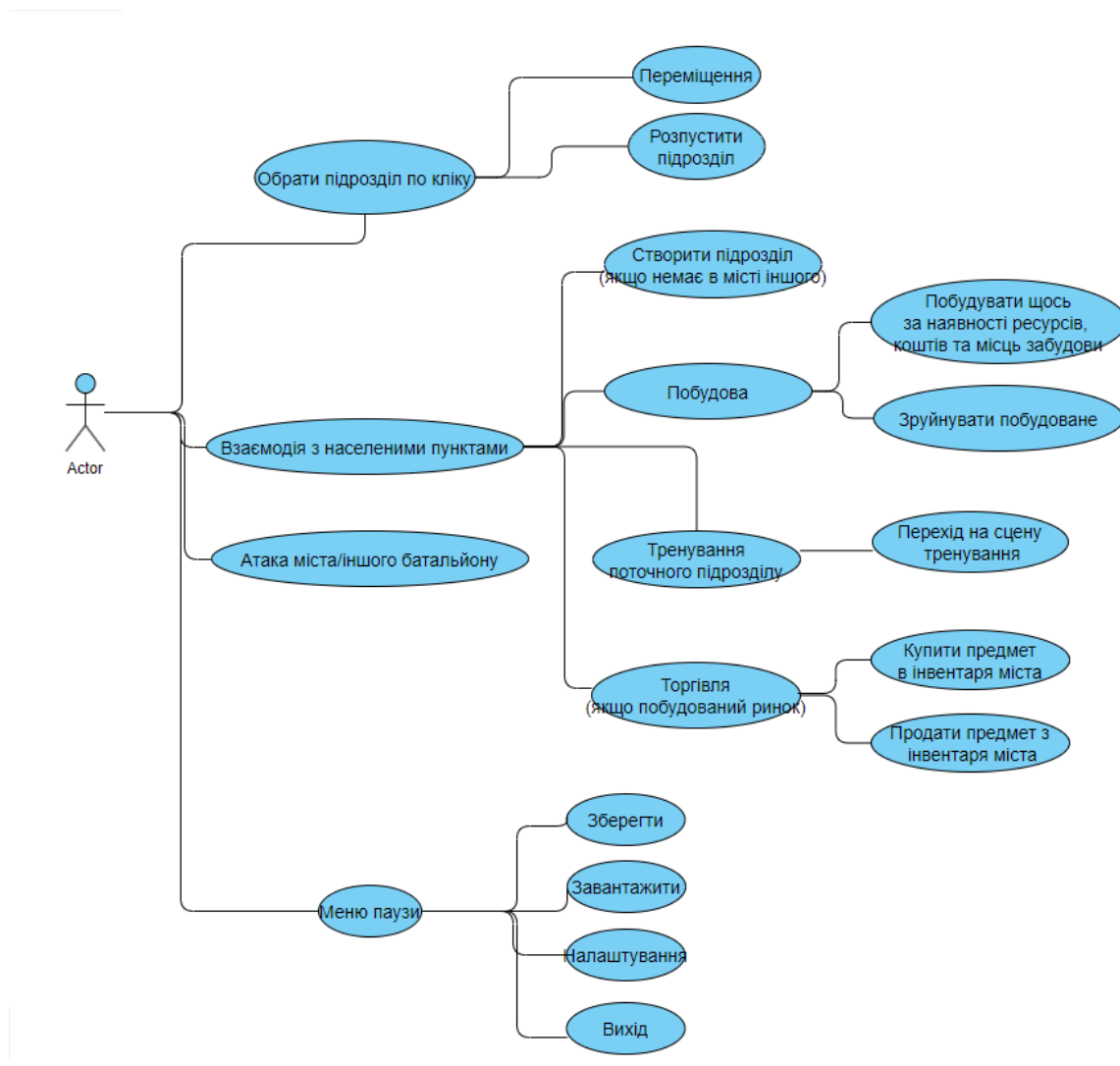


Рис. 2.2 – Діаграма варіантів використання глобальної мапи

На рисунку 2.2 зображена діаграма варіантів використання гри на глобальній мапі. Це окрема сцена, на якій будуть розміщені фігури, підконтрольні гравцю або комп'ютеру.

Функціональні вимоги:

- Менеджмент власних населених пунктів: гравець може будувати чи руйнувати будівлі, які відкривають різні можливості; тренувати військо (нейромережу), яке знаходиться зараз в місті; торгівля.
- Менеджмент батальйонів: можливість створити та керувати окремою фігурою на мапі, що уособлює загін, та екіпірувати відповідно до рангу бійців в ньому.

- Батальйони противника атакуватимуть гравця, потрібна можливість керувати боєм під час нападу.
- меню паузи: користувач має змогу в меню паузи вийти з гри, відкрити налаштування та змінити налаштування графіки, звуку та деяких елементів нейромережі для кращих результатів або ж просто продовжити гру.

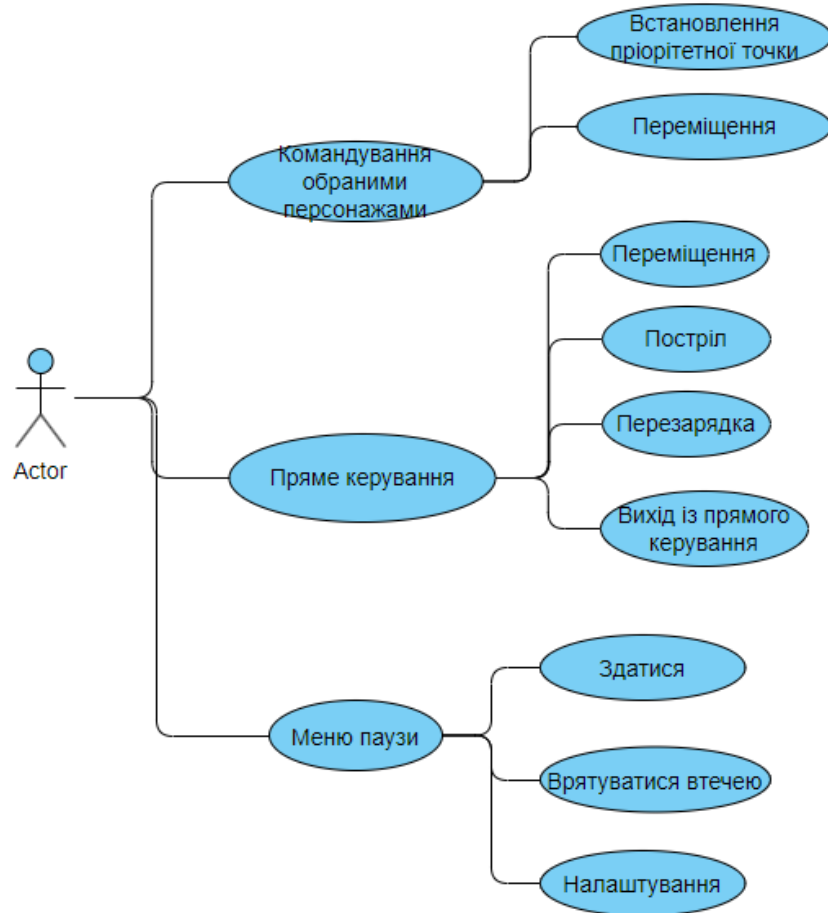


Рис. 2.3 – Діаграма використання під час віртуального бою

Командування – важливий процес на шляху до перемоги. Гравець повинен мати можливість вибирати різних бійців під час бою на віддавати накази щодо пріоритетної точки, а також взяти під прямий контроль одного із персонажів.

2.1.2. Діаграма класів

Загальними класами є `AppInstaller`, `StorageDatabase`, `UIService`, `AssetProvider` та `IOHelper`. `AppInstaller` – клас, який ініціює основні компоненти гри та реєструє їх в контейнері залежностей `DI`.

`UIService` – клас, який відповідає лише за інтерфейс, який бачить користувач. В цілях оптимізації цей клас додається в `DontDestroyOnLoad` та має лише один канвас на відображення усіх елементів.

`AssetProvider` – невеликий клас, який завантажує ресурси по адресі із `Addressable Groups`. Підтримує асинхронне завантаження

`IOHelper` – основні функції запису та завантаження даних з локальних файлів за допомогою серіалізації. Також має додаткові метод для створення ієрархії папок та коректне опрацювання деяких виключень.

Для реалізації `Dependency Injection` було використано `Zenject`. Він має необхідний набір функцій, детальну документацію та декілька корисних відео з прикладом реалізації на `YouTube`. Ця бібліотека вимагає створення префабу `ProjectContext` в папці `Resources` проекту. Цей префаб уособлює контейнер залежностей, які поширюються на всі контейнери проекту. Для цього чудово підходять, наприклад, сервіси локалізації, керування сховищем, аудіо-менеджери і т.д. Це зручно і надійно, тому що не потрібно вручну встановлювати залежності між компонентами префабів, оскільки все буде в контейнері. Нижче наведений рисунок на якому продемонстровано реєстрацію компонентів в `ProjectContext`.

`Addressables` – одна із дуже корисних технологій рушія `Unity3D`, яка дозволяє працювати з ресурсами гри через текстову адресу. Основні переваги: підтримує асинхронне завантаження, зручно сортувати ресурси гри, безпечне кодування ресурсів в бінарні файли. Дуже часто доводиться звертатись до `Addressables`, тому хорошим виходом із ситуації буде створення «провідника», який буде забезпечувати зв'язок.

```

1  using Diploma.Application.Providers;
2  using UnityEngine;
3  using Zenject;
4
5  namespace Diploma.Application
6  {
7      public class AppInstaller : MonoInstaller
8      {
9          [SerializeField] MonoBehaviour[] autoBind;
10
11          public override void InstallBindings()
12          {
13              Container.
14                  Bind<AssetProvider>().
15                  FromNew().
16                  AsSingle().
17                  CopyIntoAllSubContainers().
18                  NonLazy();
19
20              for(var x = 0; x < autoBind.Length; x++)
21              {
22                  Container.
23                      Bind(autoBind[x].GetType()).
24                      FromInstance(autoBind[x]).
25                      AsSingle().
26                      //CopyIntoAllSubContainers().
27                      NonLazy();
28              }
29          }
30      }
31  }

```

Рис. 2.5 – AppInstaller

Ядро нейромережі реалізовано в відповідному класі `NeuralNetwork`, який в конструкторі приймає масив `int`, що відподає за кількість нейронів на кожному шарі. На основі цього класу реалізовані базові механіки в `NeuralAgentBehaviour`. Він будує мережу, або ж завантажує дані, збирає дані вхідного шару та викликає опрацювання даних вихідного шару. Одна із найважливіших функцій цього класу – нагорода. Наступний по ієрархії – клас `NeuralUnit`: включає функції навчання нейромережі, має інформацію про стан здоров'я персонажа та номер команди, а також інформація про видимі об'єкти (ворог, друг, сцена). Саме від цього класу наслідуються всі ігрові персонажі, які мають навчатись. `TacticalNeuralUnit` – це теж наслідник `NeuralUnit`, тільки він враховує ще розташування «пріоритетної точки», яку під час битви ставить реальний гравець або ж штучний інтелект.

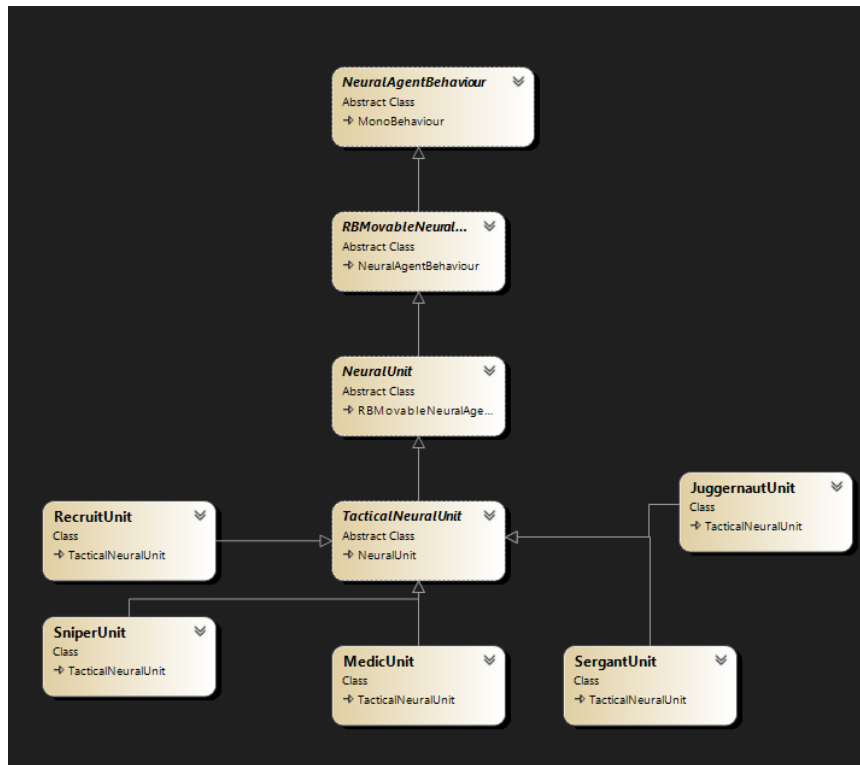


Рис. 2.6 – Діаграма класів персонажів

Клас GeneticOptimisation потрібен для керування еволюцією нейромережі. Еволюція – це метод, який обирає пари з найкращим fitness та створює нові екземпляри мережі з модифікованими даними.

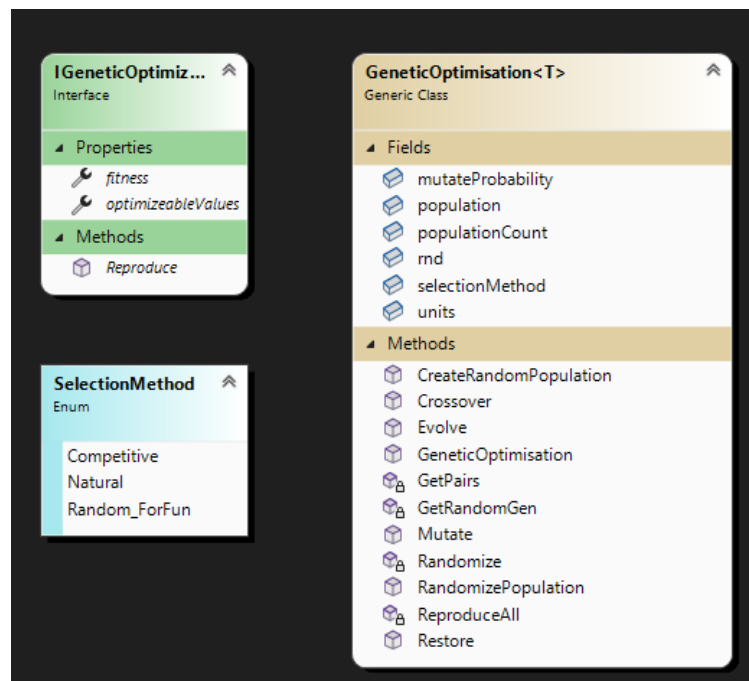


Рис. 2.7 – Загальний вигляд класу GeneticOptimisation

Усіма процесами на тренуванні або в бою керує базовий клас `ModeBaseBehaviour`. Він розміщує війська команд, завантажує або створює екземпляри нейромережі і підключає їх до персонажа, налаштовує сцену відповідно до сценарію. На сцені може бути лише один екземпляр цього класу, тому є сенс унаслідувати його від `MonoInstaller`, що дозволить зареєструвати цей компонент в контейнері сцени (`SceneContext`).

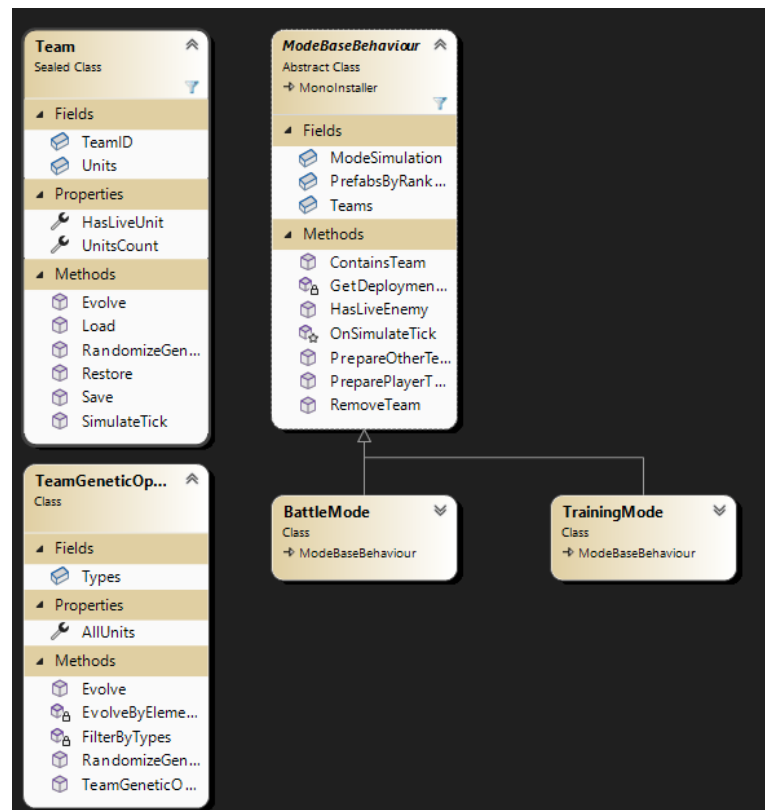


Рис. 2.8. — Загальний вигляд `ModeBaseBehaviour`

`BattleMode` розташовує бійців відповідно до вхідних даних із глобальної мапи та запускає бій. Після його завершення записує дані (втрати, поранені, хто переміг) та рахує досвід, який отримала армія за цей час. Для зручності збереження даних вони записуються в класи-моделі, і потім серіалізуються у файл.

Кожен гравець на глобальній мапі має свій контроллер. Оскільки у всіх (штучний інтелект, реальний гравець) можливості мають бути однакові, то вся базова логіка потрапила до класу `GlobalMapPlayerBase`. Різниця між `GlobalPlayer` та `GlobalAIPlayer` полягає в механіках керування та у відображенні UI.

Усі об'єкти, з якими можна взаємодіяти на мапі (населені пункти, підрозділи) називаються – ігровими фігурами. Вони містять інформацію про власника (штучний інтелект, реальний гравець), розмір та ранги особового складу.

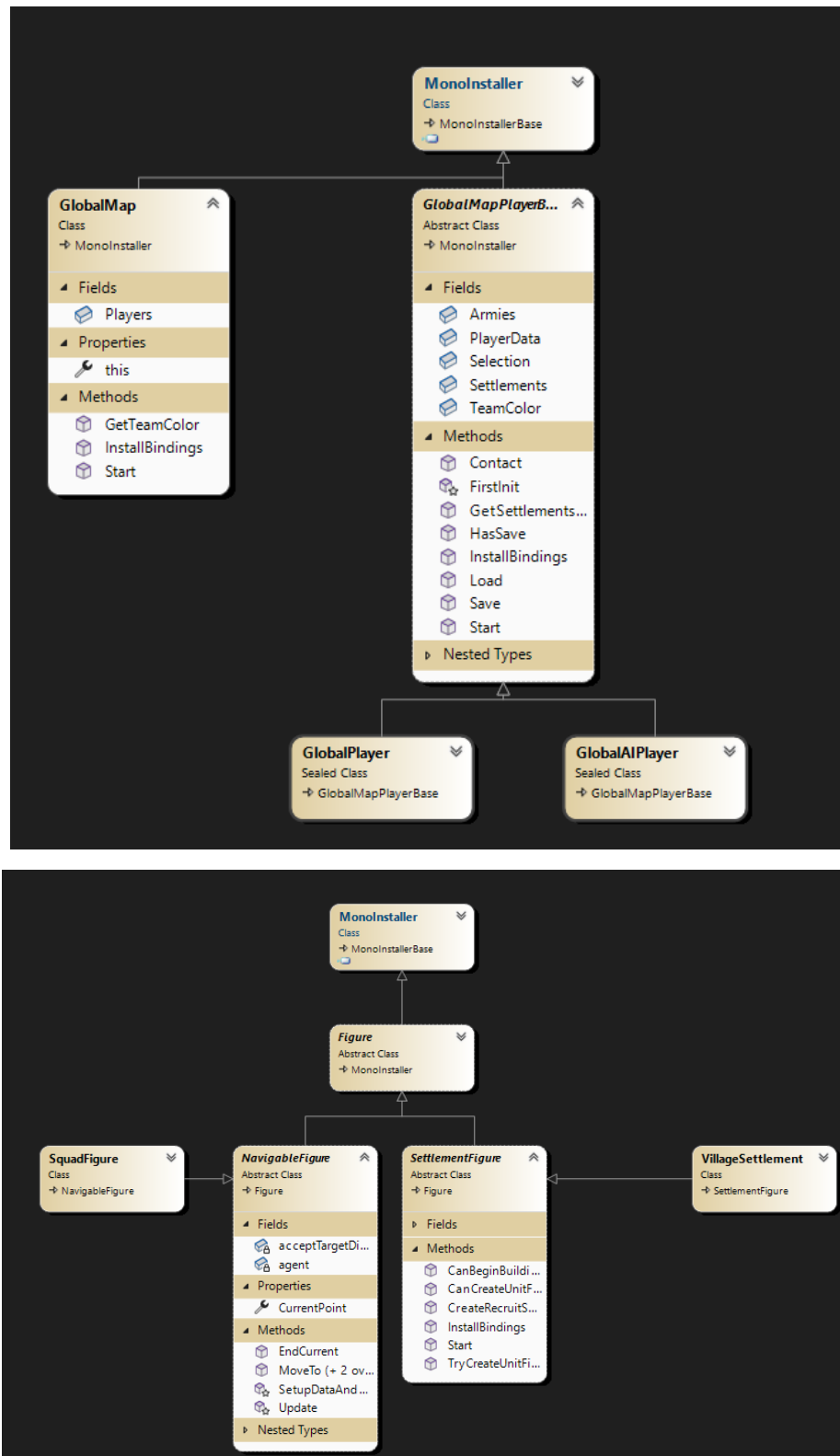


Рис. 2.9. – Структура контролерів на глобальній мапі

За поведінку гравця відповідає клас `PlayerController`. Він передає керування конкретним персонажем на клавіатуру та мишку.

2.2. Дизайн документ

2.2.1. Геймплей

Гравець отримує на старті два населених пункти та один батальйон рекрутів, яких необхідно натренувати. Противники, які керуються комп'ютером, мають по 3 населених пункти та 2 батальйони, але значно слабших по рангу. Події відбуваються на вигаданому континенті, де гравець повинен оборонятись від двох противників. Завершити гру можна два шляхами:

1. Захопити всі населені пункти суперників
2. Знищити всі війська суперників

Для цього необхідно будувати в населених пунктах заводи, житлові будинки, підприємства. Кожне місто чи село має обмежену кількість місць для забудови. Звісно, що на будівництво потрібен час, гроші та ресурси. Одночасно може будуватись не більше одного об'єкта в межах населеного пункту.

Важливо підкреслити, що для збільшення чисельності особового складу потрібні люди. Для цього потрібно будувати житлові будинки, які з часом покращують лічильник населення в містах. В селах це працюватиме не так ефективно.

Після налаштування економіки можна тренувати війська. Для цього потрібно обрати підрозділ та відправити його в будь-який свій населений пункт. При вході бійців на територію міста буде меню, в якому доступно «тренування». В цьому режимі ми можемо керувати швидкістю часу, положенням військ своїх та ворогів. Механіки менеджменту юнітів точно такі ж як і в бою: переміщення, вказання пріоритетних точок для груп. Особливість полягає в тому, що кожен ігровий персонаж – нейромережа, якій потрібен час на навчання. Процес відбувається наступним чином: запускається таймер на певний час N (час епохи) по завершенню якого буде обрана мережа з найкращим показником ефективності

для «схрещування» з іншою. Після цього всі стають на своє початкове місце, відновлюється лічильник здоров'я, боєкомплекту і починається цикл заново.

Якщо здається, що бійці чомусь навчилися, можна випробувати їх в бою проти одного із двох противників. Напасти можна на війська або одразу на місто. Різниця в чисельності особового складу. Можна створювати декілька батальйонів, щоб раціональніше використовувати ресурси. При зіткненні фігурок на мапі відкривається невеличке меню, в якому треба обрати режим бою.

1. Авто – обраховує потенційну силу обох сторін та показує результат.
2. Керований – перехід на відповідну сцену, вмикається стратегічна камера, розташовуються бійці.

Дані, які були отримані на тренуваннях завантажуються в нейромережі. У випадку з опонентом інша ситуація: він копіює дані з персонажів реального гравця та випадковим чином змінює значення. Бій триває доки одна із сторін повністю не втратить боєздатність. Кожен персонаж отримує досвід, який дорівнює найвищому значенні ефективності за весь час сутечки. Коли вони накопичили достатньо досвіду – вмикається кнопка покращення.

Під час бою можна натиснути на персонажа та перейти до прямого керування. Це режим, в якому ми можемо самі контролювати бійця та використовувати наявне в нього обладнання чи екіпірування. Щоб вийти з нього достатньо натиснути клавішу «Esc». Всі бали ефективності, які отримує гравець за цей період, будуть передані в якості досвіду.

2.2.4. Локації / сцени

Події гри відображені на декількох локаціях: глобальна мапа, два види міст, один тип села, тренування. Також ще є сцена головного меню, в якій лише один скрипт, що відповідає за відображення потрібного екрану.

На глобальній мапі є симуляція погоди та зміна дня/ночі. Час можна прискорити в декільках раз, якщо потрібно, наприклад, дочекатися завершення будівництва.

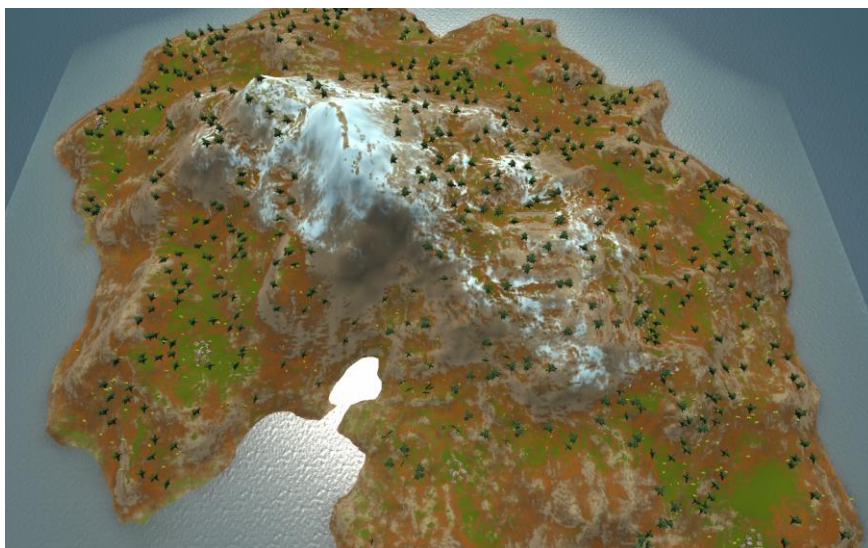


Рис. 2.10 – Глобальна мапа



Рис. 2.11 – Місто першого типу

На кожній із фігурок населених пунктів можна налаштувати такий параметр, як «SceneType». Це дає можливість швидко змінювати ігрову сцену, яка буде завантажуватись при переході в бій.

2.2.5 Текстури, персонажі та різні гейм-об'єкти

Для реалізації рельєфної поверхні можна використати Terrain. Це вбудований в рушій інструмент, за допомогою якого можна змінювати форму,

текстури ландшафту. На рисунку 2.12 дано приклади текстур, що застосуються в грі для міста першого типу. [20]

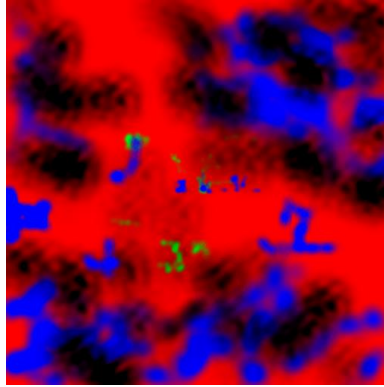


Рис. 2.12 – Текстури ландшафту міста першого типу

Оскільки візуально персонажі мають відрізнитись один від одного на різних рангах – був створений загальний префаб, який включає в себе різні частини тіла та варіанти екіпірування. Важливо підкреслити, що бійців може бути на сцені одночасно до 50, тому моделі не повинні містити більше 50 тисяч трикутників.



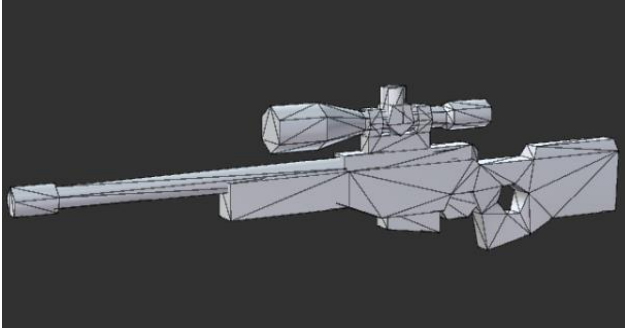
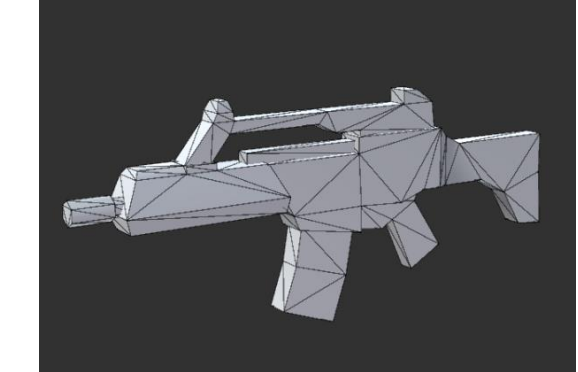
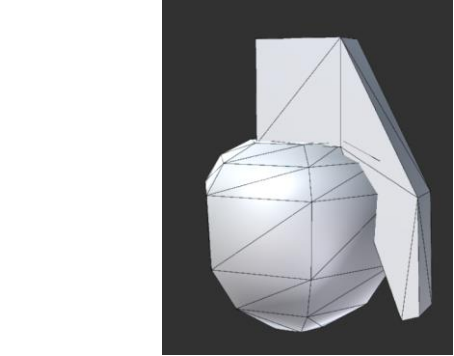
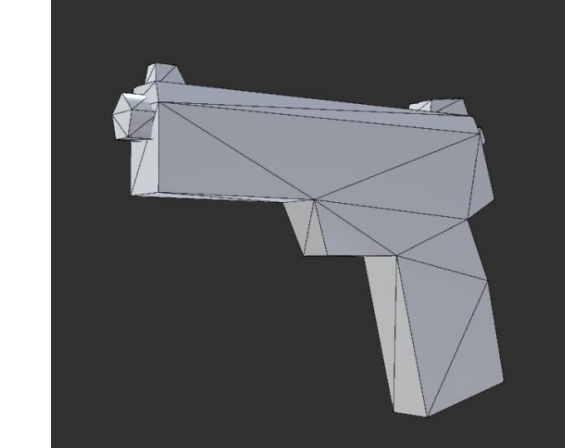
Рис. 2.13 – Візуальний вигляд персонажів різних рангів

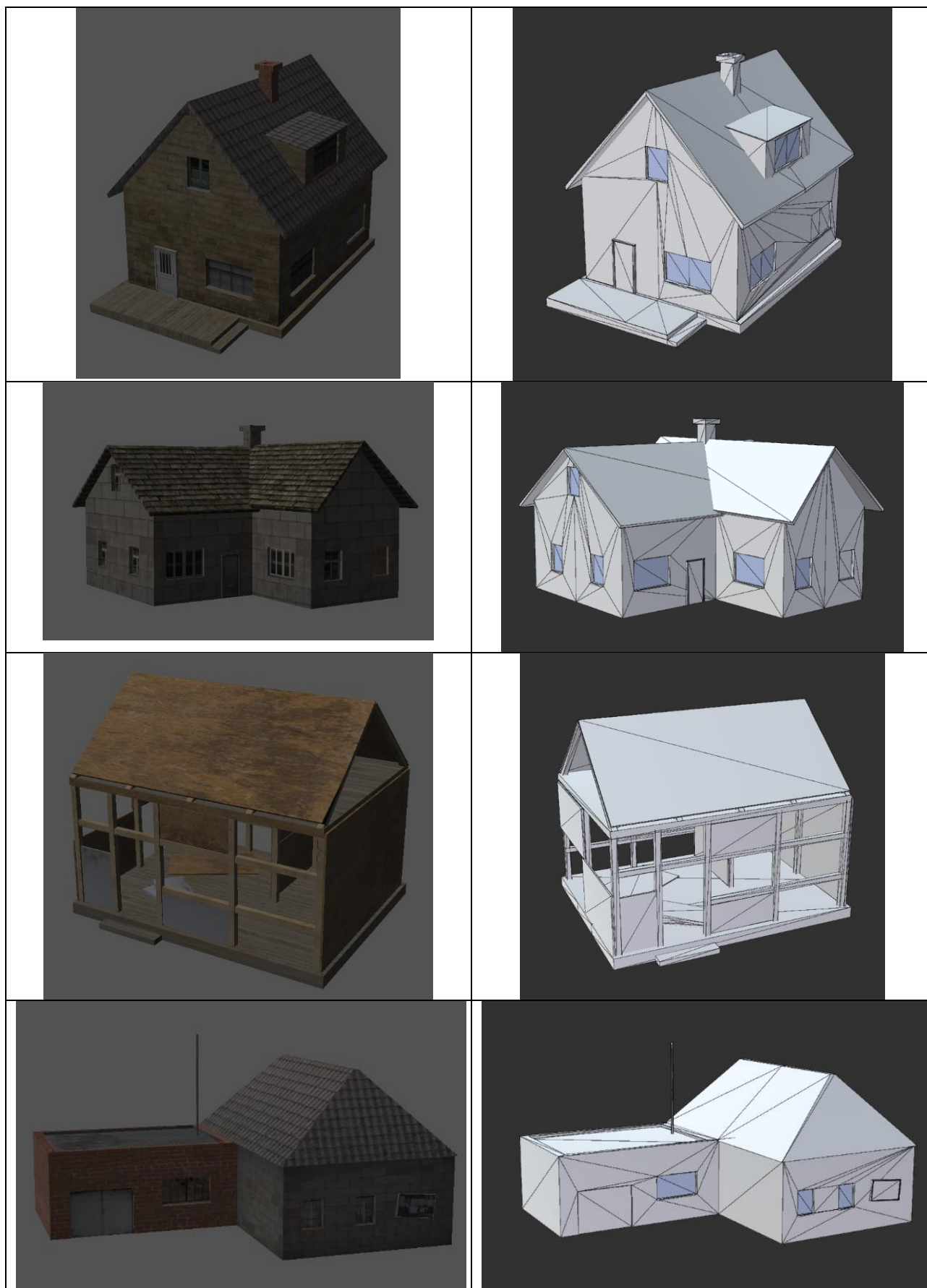
Варіантів зовнішнього вигляду набагато більше, ніж показано на попередньому скріншоті, але для досягнення мети вистачить і цієї кількості. Використання «конструктора» покращує враження під час гри, але варто пам'ятати про оптимізацію та кількість об'єктів, що знаходяться одночасно на сцені.

В таблиці 2.1 наведені деякі 3D-моделі, що використовуються у грі. [20]

Таблиця 2.1

3D-моделі

Сітка 3D-моделі	3D-модель з текстурами
	
	
	
	





Для інтерфейсу зазвичай потрібна велика кількість текстур, тому хорошим рішенням буде створення атласу текстур. Це може значно пришвидшити завантаження префабів. На рисунку 2.14 відображений атлас, що був згенерований за допомогою вбудованих можливостей рушія.

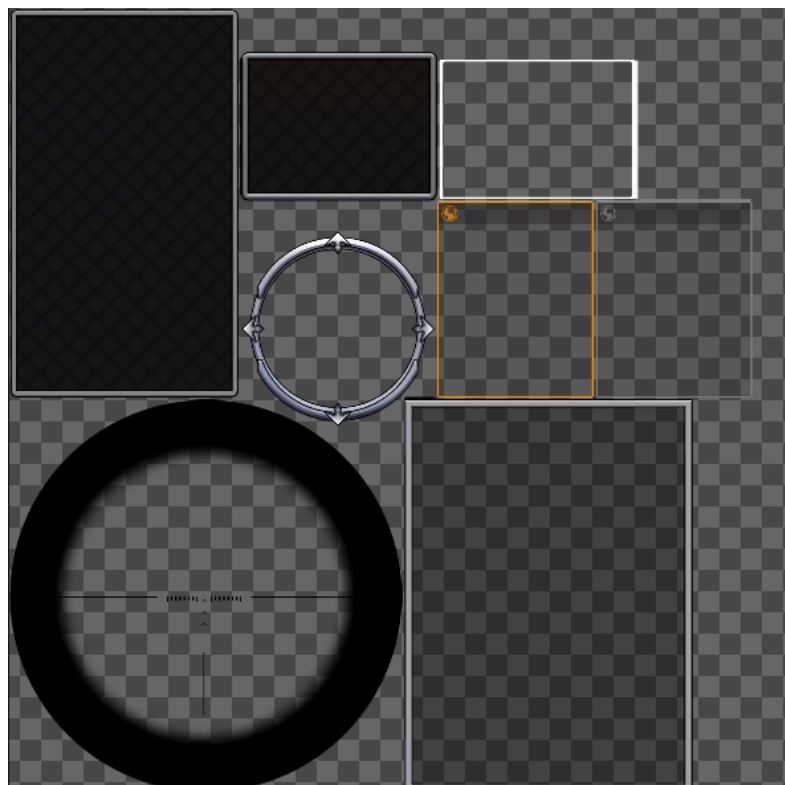


Рис. 2.14 – Атлас UI текстур

Анімації персонажів можна реалізувати, використовуючи Animator. Він простий у використанні і повністю відповідає вимогам. Щоб анімації виглядали гарно буде застосований сервіс Міхато, який на готову 3D-модель накладає ріг та пропонує десятки цікавих анімацій. Особливістю є те, що вони записані за технологією Motion Capture.

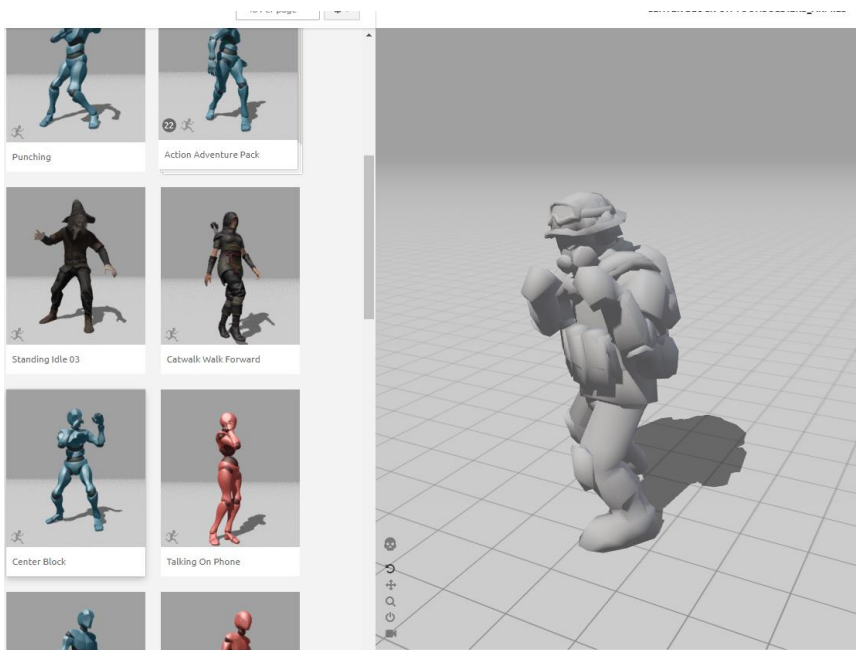


Рис. 2.15 – Підготування анімацій персонажів

2.2.6. Керування

На глобальній мапі гравець може виділяти лівим кліком миші свої фігурки. На правий клік, якщо вдалось виділити батальйон, переміщення військ у вказану точку. Коли виділення активне (див. рис. 2.16) – доступна панель взаємодії. В ній можна переглянути: особовий склад, ранг та рівень кожного; інвентар та його вміст; і якщо виділено населений пункт, то можна побудувати щось.

Зверху розташована панель ресурсів, яка відображає загальну кількість ресурсів певного типу. На рисунку 2.16 видно, що на той момент гри у гравця було 28 рекрутів, 52 одиниці дерева та 3 150 ігрової валюти. Трохи нижче відображений віртуальний час та дата.



Рис. 2.16 – Інтерфейс додатку на глобальній мапі

Керування камерою на глобальній мапі відбувається за допомогою клавіш W, A, S, D – для переміщення, Q, E – для зміни повороту камери.

Під час бою можна виділити два режима керування: стратегічний та пряме. Перший працює за аналогією з глобальною мапою, тільки ще додана можливість розміщувати пріоритетну точку на клавішу «1» та обирати персонажа для наступного типу керування (клавіша «4»). Для тренування доступні ще «2-3», що

відповідають за спавн-поінти кожної з сторін. Це для того, щоб можна було тренувати мережу в різних ситуаціях.

Пряме керування включає в себе: переміщення гравця на стандартні клавіші W, A, S, D; використати поточне екіпірування персонажу – лівий клік, перезарядка – R. Щоб повернутися до стратегічного керування можна натиснути Esc. Також пряме керування буде зупинене, коли юніт повністю втрачає боєздатність.

2.3. Реалізація алгоритмів роботи гри

У даному розділі розглянуто частину створеного коду, що дасть основні уявлення про процес розробки. Детальніше можна ознайомитися з важливою частиною коду в додатку А.

Далі наведено реалізацію AppInstaller, який одразу реєструє в контейнері Dependency Injection найнеобхідніші класи. Наприклад, провідники (Providers), адже вони використовуються майже всюди [14].

Лістинг коду:

```
namespace Diploma.Application
{
    public class AppInstaller : MonoInstaller
    {
        [SerializeField] MonoBehaviour[] autoBind;

        public override void InstallBindings() { }
    }
}
```

Далі наведений код AssetProvider. Забезпечує зв'язок гри з її файлами. В ньому лише дві асинхронних функції: LoadAssetAsync – для завантаження файлу за адресою. ReleaseAsset – для звільнення пам'яті девайсу від раніше завантаженого об'єкту. Лістинг коду:

```
namespace Diploma.Application.Providers
{
    public sealed class AssetProvider
    {
        public async Task<T> LoadAsset<T>(string address) where T : Object
        {
            return await Addressables.LoadAssetAsync<T>(address);
        }

        public void ReleaseAsset<T> (T target) where T : Object
    }
}
```

```

        {
            Addressables.Release(target);
        }
    }
}

```

Далі розглянемо код UIService. Клас працює лише з елементами UI. Розміщується в ProjectContext для доступу з будь-якої точки проекту. Префаб цього сервісу містить один Canvas. Він і буде відображати всі елементи[6]

Лістинг коду:

```

namespace Diploma.Application.Services.UI
{
    public sealed class UIService : MonoInstaller
    {
        public ElementsContainer<UIScreen> Screens;
        public ElementsContainer<UIPopup> Popups;
        public Camera UICamera;

        [Inject] AssetProvider assets;

        [SerializeField] RectTransform screensParent;
        [SerializeField] RectTransform popupsParent;

        public override void InstallBindings() { }
    }
}

```

Клас ElementsContainer<T> ініціалізує нові об'єкти та зберігає посилання на них в пам'яті, щоб можна було звернутись до них в майбутньому. Асинхронна функція CreateElement завантажить префаб по, вказаному в атрибуті AssetPath, шляху. Параметр clear відповідає за очистку всіх інших елементів цього контейнера. Лістинг коду:

```

namespace Diploma.Application.Services.UI
{
    public sealed class ElementsContainer<T> where T : UIElement
    {
        private readonly AssetProvider Assets;
        private readonly RectTransform Root;

        private List<T> Items;

        public bool IsActiveShowRequest { get; private set; }

        public ElementsContainer(AssetProvider assets, RectTransform root){ }

        public async UniTask<U> CreateElement<U>(bool clear = true, bool findInCreated = false) where U : T { }

        public void RemoveElement<U>() where U : T { }
        public void RemoveElement (Type targetType) { }

        public bool Contains <U>(out U screen) where U : T { }
        public bool Contains (Type targetType, out T element) { }

        public U GetElement<U>(bool exception = true) where U : T { }
    }
}

```

```
}  
}
```

Гра починається на сцені GlobalMap, де є відповідний контролер. Він «слідкує» за гравцями, запускає битви або ж тренування, зберігає та коректно підставляє завантажені дані. Зберігає лише значення Ticks, отримане із віртуальної дати. Лістинг коду:

```
namespace Diploma.Game.Global  
{  
    public class GlobalMap : MonoInstaller  
    {  
        public const int MAX_TIMESCALE = 15;  
        public GlobalMapData Data;  
        public GlobalMapPlayerBase[] Players;  
  
        public float Timescale  
        {  
            get => Time.timeScale;  
            set => Time.timeScale = Mathf.Clamp(value, 0, MAX_TIMESCALE);  
        }  
  
        public GlobalMapPlayerBase this[int teamID] { }  
        public Color GetTeamColor (int teamID) { }  
  
        public override void InstallBindings() { }  
        public override void Start() { }  
        private void Update() { }  
  
        #region IO  
        public void Save () { }  
        public void Load () { }  
        public bool HasSave() =>  
            File.Exists(WorldSavePath());  
  
        private string WorldSavePath() =>  
            UnityEngine.Application.persistentDataPath + "/World" +  
            NeuralNetwork.NN_FILE_EXTENSION;  
        #endregion  
  
        public async void InitializeBattle (Figure attacker, Figure defender) { }  
    }  
  
    [Serializable]  
    public class GlobalMapData  
    {  
        public long Ticks;  
    }  
}
```

Наступний клас – GlobalMapPlayerBase. Це базовий клас, який одразу включає в себе всі необхідні функції для роботи з гравцем. Від нього є два похідних класи GlobalPlayer та GlobalAIPlayer. Перший – відповідає за вчасний показ інтерфейсу, а другий – за алгоритм комп’ютера на глобальній мапі.

Комп'ютер-гравець має певний алгоритм роботи: спочатку розвиток армії, потім будівництво необхідних підприємств та напад на противника, але результат не завжди буде онадковий із-за псевдовипадкових значень Random.Range(). Лістинг коду:

```
namespace Diploma.Game.Global.Players
{
    public sealed class GlobalAIPlayer : GlobalMapPlayerBase
    {
        [Inject] UnitProductionLine units;
        [Inject] BuildingsData buildingsData;
        [Inject] GlobalMap global;

        public int TargetSquadsAmount = 3;
        public int DefaultSquadSize = 5;
        public int PatrolRadius = 15;
        public float MinTrainRateInSec = 10;
        public float MaxStayTime = 5;
        public float BuildingRate = 15;
        public float MinAssaultRate = 120;

        private SquadFigure targetFigure;
        private SquadFigure assaultFigure;

        protected override void Update() { }

        private void CheckNeedStartAssault () { }

        private float lastAssaultTime;
        private bool canBeginAssault =>
            Time.time - lastAssaultTime > MinAssaultRate;

        private Figure GetRandomEnemy (bool squadOrSett) { }

        private float lastTrainTime;
        private bool canTrain =>
            Time.time - lastTrainTime > MinTrainRateInSec;

        private float lastIdleTime;
        private float genStayTime;
        private bool canPatrolNext =>
            Time.time - lastIdleTime > genStayTime;

        private void ComplectCurrentTarget (bool achivedCity) { }
        private void FindAndFocusOnTarget () { }
        private bool TryCreateNewSquad () { }
        #endregion

        private SettlementFigure GetRandomCity () { }

        private SettlementFigure GetNearestCity (SquadFigure targetFigure, bool
includeRecruitPrice) { }

        #region Settlements Controller
        private float lastBuildingTime;
        private bool canBuildByTimer =>
            Time.time - lastBuildingTime > BuildingRate;

        private void UpdateBuildings () { }

        private bool TryBuild (SettlementFigure settlement, string buildingID) { }
```

```

        #endregion
    }
}

```

Як тільки гравець увійде в своє місто і запустить тренування, або потрібно відбиватись від нападу, то контроль над грою передається до іншого скрипта та на іншу сцену, де і приймає участь нейромережа. ModeBaseBehaviour розставляє персонажів, завантажує нейромережі в префаби солдатів, налаштовує параметри команд. Ядро нейромережі виглядає наступним чином. Функцію активації можна змінити на кожному шарі окремо або ж на всіх одразу. Лістинг коду:

```

namespace Diploma.Game.AI
{
    [System.Serializable]
    public class NeuralNetwork
    {
        public int[] layers;
        public double[][][] weights;
        public double[][] bias;
        public double[][] neuronsOutputs;

        [System.NonSerialized]
        public IActivationFunction[][] activateFunctions;

        public NeuralNetwork(NeuralNetwork copy) : this(copy.layers) { }
        public NeuralNetwork(params int[] layers) { }

        public void SetActivateFunction(IActivationFunction func) { }
        public void SetActivationFunctionForLayer(int layerNum, IActivationFunction
func) { }

        public double[] Compute(double[] input) { }
        public List<double> Serialize() { }
        public void Deserialize(List<double> data) { }
        public NeuralNetwork Copy() { }

        public const string NN_FILE_EXTENSION = ".nd";

        public static void ClearDirectory (string directory) { }
        public static void PrepareDirectory (string directory) { }
        public static bool ContainsSave(string fileName) { }
        public static string PathByFolderName (string directory) { }
        public static string PathByFileName(string fileName) { }
        public static void Save(GeneticOptimizeableNerualNetwork network, string
fileName) { }

        public static GeneticOptimizeableNerualNetwork Load(string fileName) { }
        public static IActivationFunction ActivationFuncByType(ActivationFuncType
funcType) { }
    }
}

```

Висновки до другого розділу

Діаграма класів дає загальне уявлення про те, як має відбуватись процес гри з технічної точки зору. Також проведена підготовка необхідного візуального контенту, а саме: 3D-моделі персонажів та оточення; сформовано ландшафти для кожної із ігрових сцен. В цілях оптимізації згенеровані спрайт-атласи, які містять набір текстур певного призначення (інтерфейс, UV-розгортка).

Проведено розробку програмного забезпечення, яке створене на основі дизайн документу та технічного завдання: реалізована базова логіка гри на глобальній карті та її гравців, які в свою чергу керують своїми фігурками. Всі статичні фігури дозволяють будувати в них щось, а динамічні можуть вступати в бій з іншою фігуркою будь-якого типу. Під час бою завантажуються натреновані дані нейромережі. В результаті розроблено готовий програмний продукт, що повністю виконує умови поставлених задач.

Одним з важливих аспектів розробки гри є баланс. У грі важливо, щоб всі типи фігурок були рівносильними та щоб гравці мали можливість вибрати стратегію, яка буде для них найбільш ефективною. Для досягнення балансу необхідно ретельно проаналізувати всі аспекти гри, такі як характеристики фігурок, їхні можливості та взаємодії між ними.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ

3.1. Інтерфейс та порядок роботи з програмним додатком.

Опис роботи з додатком включає в себе розробку текстової та ілюстраційної частини, які демонструють зовнішній вигляд закінченого продукту. Зустрічає гравця головне меню гри. В ньому є заплановані функції, а саме: створення або продовження гри, налаштування та вихід. При натисканні на кнопку «Play» відбудеться перехід на сцену глобальної мапи.

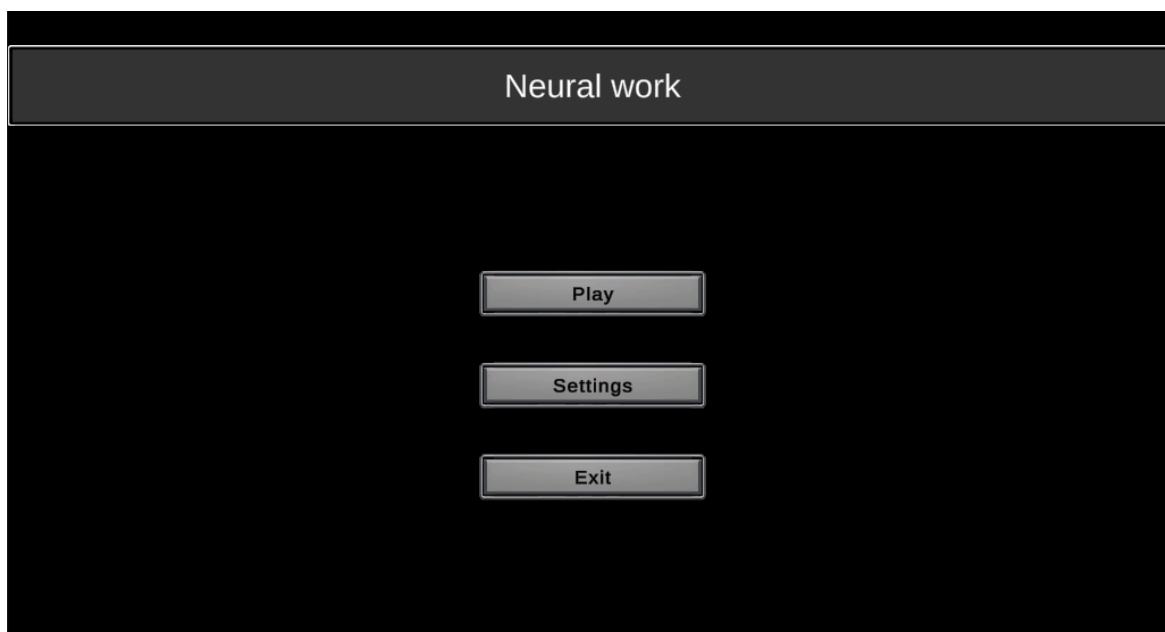


Рис. 3.1 – Головне меню гри

Після декількох секунд завантаження потпапляємо на острів, де можна знайти населені пункти свої та противників (див. рис. 3.2). Кожен з гравців має певний колір (бірюзовий – ми; червоний, зелений - противники).

Найперше в чому виникає потреба під час гри – це гроші та люди. Варто попіклуватися про побудову житлових будинків (час від часу збільшують населення на 1 одиницю) та лісопилки. Щоб це зробити потрібно виділити місто та побудувати житловий комплекс. На панелі інформації (див рис. 3.3) переходимо в вкладку «Building» та натискаємо на будь-який вільний слот. В новому вікні обираємо те, що відповідає потребам. Потрібно також звернути увагу що на будівництво потрібен час та певна кількість ресурсів.



Рис. 3.2 – Початкове положення гравця

Тепер можна підготувати армію. Для цього обираємо місто, де є рекрути та створемо батальйон. Коли загін в межах міста – всі нові рекрути будуть в його складі, інакше залишаться на захисті міста (гарнізон). На даному етапі військо може величезним, але не ефективним. Потрібно спершу натренувати мережу, інакше вона може, наприклад, відкривати вогонь по своїм, що, звісно, недопустимо в бою. Тому, виділивши будь-який батальйон, відправляємо його в місто на тренування. По прибуттю переходимо в відповідний режим (див. рис. 3.4)

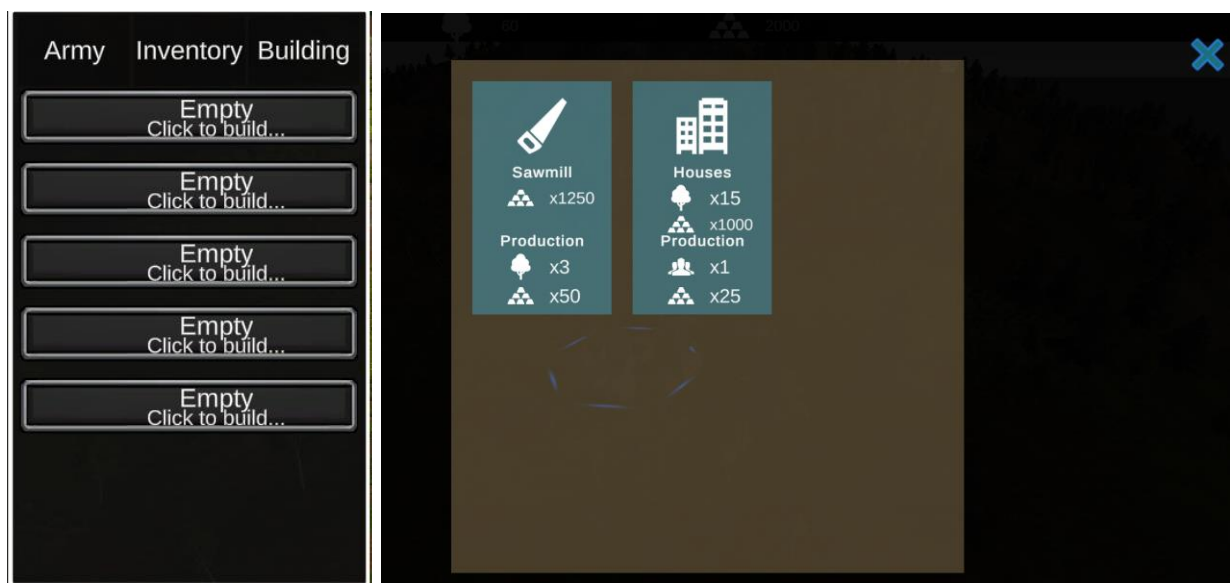


Рис. 3.3 – Панель будівництва та меню вибору будівлі

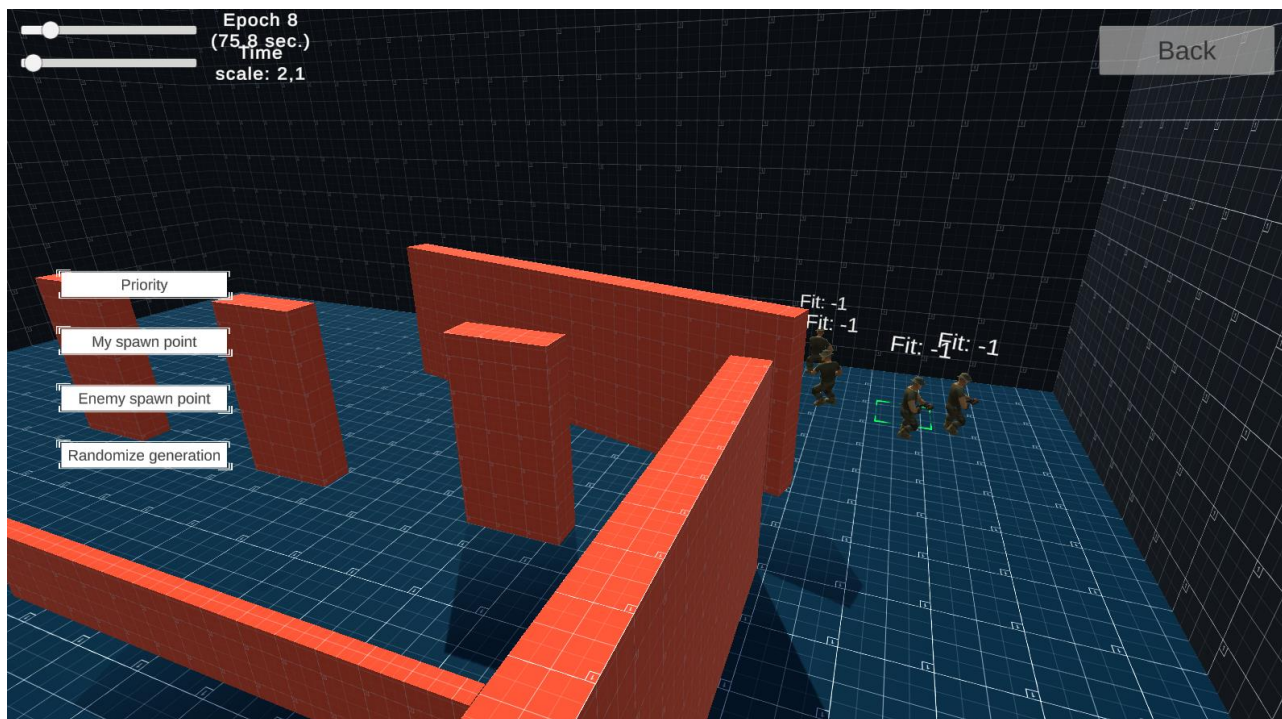


Рис. 3.4 – Режим тренування

В цьому режимі ми можемо керувати декількома параметрами:

1. Кількість часу на одну епоху
2. Швидкість часу
3. Випадкові значення мережі

Також можна змінювати положення спавн-поінтів будь-якої з сторін та проіоритетною точкою. Кожен персонаж над собою має показник ефективності. Цей параметр ключевий для визначення найкращих екземплярів мережі. Всі значення зберігаються (в цьому і сенс цього режиму) та використовуються в грі. При всіх наступних входженнях в тренування дані завантажуються, а не генеруються спочатку.

Після декількох десятків ітерацій по 70 секунд всі зрозуміли що стреляти по союзникам не можна, а по сцені – небажано, але все ж не зовсім розуміють що робити. Цього рівня, на перший час, вистачить. Натискаємо кнопку “Back” та повертаємось до глобальної мапи. Адже щоб покращити озброєння та ранг бійців таких тренувань не достатньо: потрібні ресурси та досвід.

Почнімо з ресурсів. Для покращення рекретів батальйону чи гарнізону до сержанта потрібна легка броня та зброя. І те, і те виробляє простий військовий завод, який можна будувати і в селах, і в містах. Однак, для виробництва цих речей потрібне залізо та дерево (уже отримуємо з лісопилки). Залізо можна купувати в вкладці торгівлі, де є ринок, а можна побудувати в місті металургійний завод, який раз в декілька хвилин приносить 1 залізний злиток без витрат інших ресурсів.

Потрібно не забувати, що комп'ютер теж розвивається і буде нападати на інших гравців. На рисунку 3.5 помітно, що населений пункт Froud перейшов до зеленого гравця. У випадку, якби він атакував бірюзового, тобто реального, гравця, то камера сфокусувалась би на місці події та був показаний відповідний UI (див. рис. 3.6)



Рис. 3.5 – Завоювання міст противником

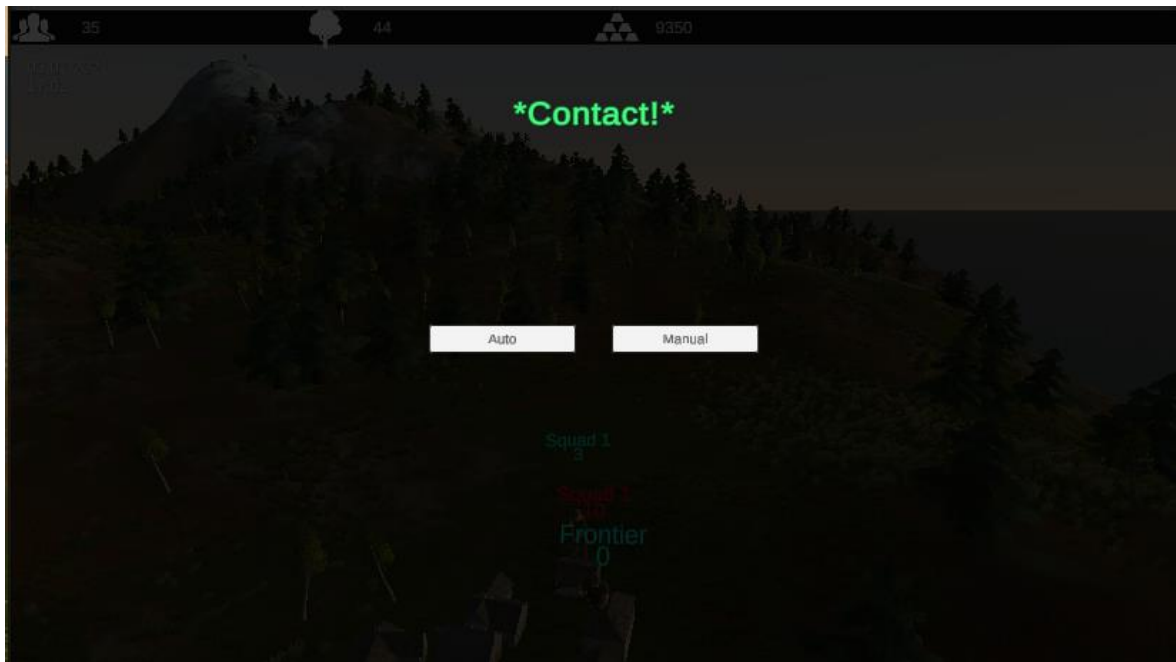


Рис. 3.6 – Напад противника на поселення гравця

За наявності бажання можна перейти в ручний режим керування битвою. У випадку, який відображений на попередньому рисунку це означає автоматичну поразку, адже в гарнізоні немає війська. Було б там хоч декілька солдатів – відбулось би завантаження мапи населеного пункту. Керування там майже таке ж, як і на тренуванні. На наступних рисунках розглянемо ситуацію, коли гарнізон не пустий. Всі дані завантажені та приступля помітна схожість послідовності дій нейромереж.



Рис. 3.7 – Інтерфейс під час бою

Час в бою відображає час, який був на глобальній мапі. Тобто, якщо на мапі ніч – в бою теж буде нічний час. Технічно це ніяк не впливає на поведінку персонажів.



Рис. 3.8 – Пряме керування

Через те, що мережа може бути погано натренована, пряме керування – дуже корисна механіка, яка дозволяє змінити ситуацію в свою користь. Звісно, що чим краще натренована мережа – тим важче буде обіграти її. Але для цього потрібно дуже багато часу, адже тренування відбувається в режимі симуляції бою і всі цілі – динамічні, а оскільки карт може бути декілька, то положення перешкод різного типу – теж.

Після завершення бою з'являється відповідний екран, який інформує про втрати та накопичений досвід (див. рис. 3.9) Помітно, що деякі персонажі готові до перекваліфікації (тому що мережу потрібно буде навчати спочатку) на інший ранг. Можемо обрати декілька варіантів розвитку. Далі наведено список можливих рангів та список ресурсів необхідних для «підвищення»:

1. Снайпер (зброя - 1)
2. Сержант (легка броня – 1, зброя - 1)

3. Медик (легка броня – 1)
4. Інженер (легка броня – 1, міни - 1)
5. Джагернаут (тяжка броня - 1, зброя - 2)

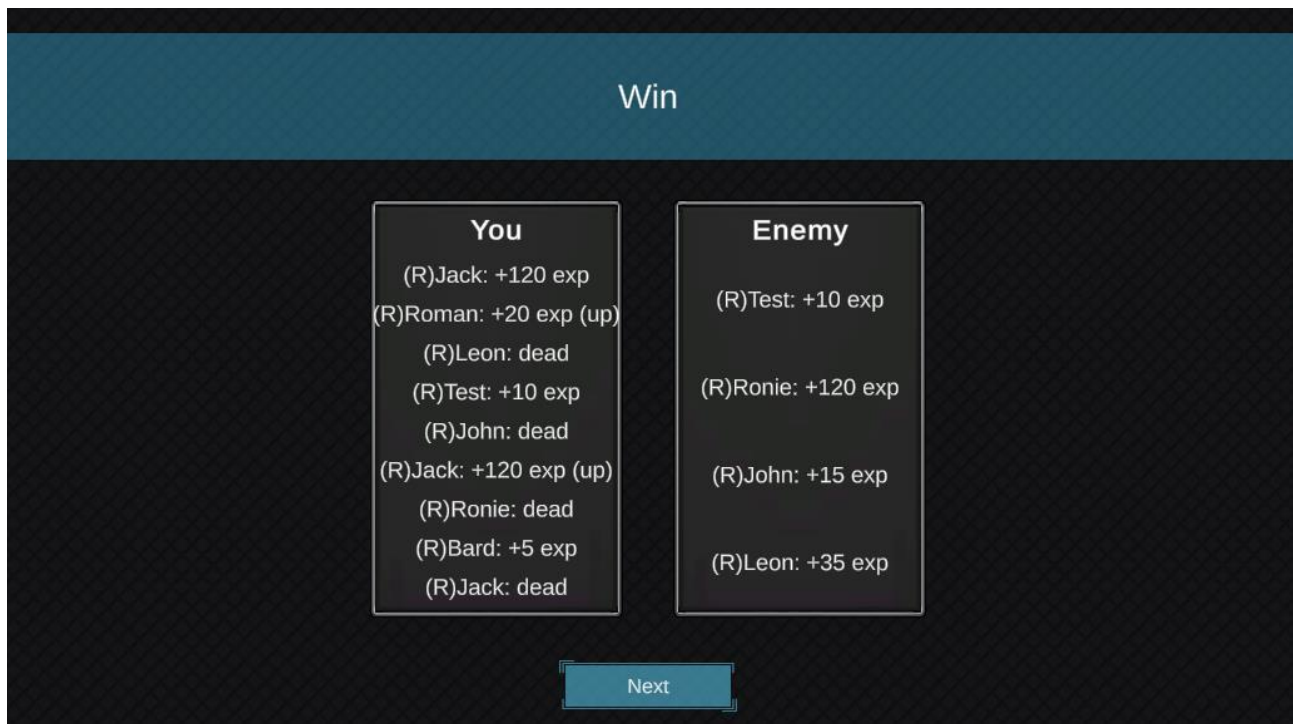


Рис. 3.9 – Екран завершення бою

3.2. Тестування роботи програмного комплексу.

Тестування програмного забезпечення - процес дослідження, випробування програмного продукту, що має дві різні цілі:

- переконатися, що програма відповідає вимогам;
- виявити ситуації, в яких поведінка програми є неправильним, небажаним.

Тест-кейс – це опис перевірки роботи системи, він описує параметри, кроки та умови, які потрібні для перевірки. Проведемо функціональне тестування для перевірки гри на її роботу здатність та відповідність основним функціональним потребам.

Тестування було проведене модульне.

Спочатку був перевірений інтерфейс гри. Гра має екрани: головне меню, глобальний режим, стратегічний режим, режим від третього лиця.

Першим було протестована перша сцена головного меню.

- Тест № 1. Ініціалізація DI контейнера:

Передумови: Гравець повинен запустити гру;

Кроки: Запуск гри;

Очікуваний результат: Відображення головного меню;

Фактичний результат: Відображення головного меню;

Тест пройдено успішно.

- Тест № 2. Кнопка «Play» на екрані меню:

Передумови: Відображене головне меню

Кроки: Натискаємо на кнопку «Play»

Очікуваний результат: перехід на етап вибору (завантажити/створити нову)

Фактичний результат: перехід на етап вибору (завантажити/створити нову)

Тест пройдено успішно.

- Тест № 3. Вибір своїх фігур на глобальній мапі:

Передумови: Активна сцена глобальної мапи

Кроки: Лівий клік миші на будь-яку фігурку з бірюзовим кольором

Очікуваний результат: Поява панелі взаємодії/інформації

Фактичний результат: Поява панелі взаємодії/інформації

Тест пройдено успішно.

- Тест № 4. Відправка обраної фігурки в певну точку

Передумови: Має бути обраний підрозділ

Кроки: Правий клік миші по ландшафту

Очікуваний результат: Фігурка починає рух у відповідному напрямку

Фактичний результат: Фігурка починає рух у відповідному напрямку

Тест пройдено успішно.

- Тест № 5. Початок тренувань

Передумови: Має бути обраний підрозділ

Кроки: Правий клік миші по союзному поселенню

Очікуваний результат: Фігурка починає рух у відповідному напрямку та перехід на сцену тренування після прибуття

Фактичний результат: Фігурка починає рух у відповідному напрямку та перехід на сцену тренування після прибуття

Тест пройдено успішно.

- Тест № 6. Зміна швидкості часу

Передумови: Гравець повинен знаходитись на сцені тренування

Кроки: Змінити значення слайдера зліва зверху.

Очікуваний результат: Помітна зміна швидкості часу

Фактичний результат: Помітна зміна швидкості часу

Тест пройдено успішно.

- Тест № 7. Зміна спавн-поінту своїх військ

Передумови: Гравець повинен знаходитись на сцені тренування

Кроки: Натискаємо на кнопку «My spawn point» та ПКМ по бажаному положенню

Очікуваний результат: На наступну епоху бійці будуть в іншій позиції

Фактичний результат: На наступну епоху бійці створились в іншій позиції

Тест пройдено успішно.

- Тест № 8. Зміна спавн-поінту військ противника

Передумови: Гравець повинен знаходитись на сцені тренування

Кроки: Натискаємо на кнопку «Enemy spawn point» та ПКМ по бажаному положенню

Очікуваний результат: На наступну епоху бійці будуть в іншій позиції;

Фактичний результат: : На наступну епоху бійці створились в іншій позиції

Тест пройдено успішно.

- Тест № 9. Нанесення шкоди супротивнику:

Передумови: Гра має бути в стані бою або тренування

Кроки: Куля, випущена зброєю персонажу, має візуально потрапити в ціль

Очікуваний результат: Віднімання хелз-поінтів

Фактичний результат: Противник отримав шкоду, яка дорівнює параметру Damage того, хто відкрив вогонь

Тест пройдено успішно.

- Тест № 10. Захоплення міста противника:

Передумови: Гра має бути в стані глобальної мапи

Кроки: Обираємо підзроділ і відправляємо його на вороже поселення

Очікуваний результат: Вікно вибору режиму битви та в випадку перемоги – перехід міста під контроль гравця

Фактичний результат: Вікно вибору режиму битви та перехід міста під контроль гравця

Тест пройдено успішно.

- Тест № 11. Захоплення міста противником:

Передумови: Гра має бути в стані глобальної мапи та противник має підрозділ, розміром більше, ніж 7 воїнів

Кроки: Очікування початку штурму

Очікуваний результат: Комп'ютер атакував реального гравця або іншого опонента

Фактичний результат: Комп'ютер атакував реального гравця

Тест пройдено успішно.

- Тест № 12. Перевірка нарахування очків за попадання:

Передумови: Гра повинна бути в стані бою або тренування

Кроки: Наносити шкоду будь-яким чином противнику

Очікуваний результат: +2 до ефективності

Фактичний результат: +2 до ефективності

Тест пройдено успішно.

Дане тестування показує, що гра відповідає функціональним вимогам, виділених на етапі постановки завдання, працює коректно, помилок під час гри не виявлено.

Висновки до третього розділу

Всі ігрові підсистеми повністю реалізовані і успішно інтегровані в єдину ігрову систему, яка дозволяє запустити ігровий процес.

Створену гру було повністю протестовано. Вона працює стабільно, без збоїв та коректно. Дані зберігаються та завантажуються. Всі події, які потребують декількох переходів між сценами, коректно обробляються та відображаються на глобальній мапі.

Нейромережа реалізована та потребує часу на навчання. Цим процесом можна керувати, змінюючи спавн-поінти будь-якої з сторін, встановлюючи пріоритетні точки та час епохи. Інтерфейс додатку відповідає потребам та відображає реальні дані.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи магістра було спроектовано та створено гру стратегію з елементами шутера, яка повністю відповідає, поставленим на початку проектування, вимогам.

Було проаналізовано ігри, які є аналогами програмного продукту, що розробляється. При аналізі розглянуто основні елементи гри, їх геймплей та тематику. Розібрано основні елементи та характеристики жанру. Також отримано теоретичний фундамент для подальшої роботи і повністю сформовано уявлення про гру, що створюється.

В ході аналізу було визначено метод навчання нейромережі та реалізоване ядро, яке є ключевим елементом геймплею, адже від неї залежить результат віртуальної битви.

Всі механіки працюють належним чином без збоїв. Гра розроблялася на рушії Unity3D, який добре себе показав під час роботи. Інтуїтивно зрозумілий інтерфейс, швидкість роботи та величезна кількість документації і повчальних роликів на YouTube допомагають в будь-якому питанні. Великі можливості для повторного використання коду та вбудована оптимізація графіки, фізики. Він повністю задовольняє нас за своїми базовими характеристиками до того ж має потужну підтримку товариства, що значно знижує поріг входження.

Локації були розроблені за допомогою стандартних можливостей рушія з використанням додаткових файлів, наприклад, текстур з офіційного AssetStore. Сцени мають охайний та завершений вигляд, зберігаючи чисту та ефективну архітектуру об'єктів.

Всі ігрові підсистеми повністю реалізовані і успішно інтегровані в єдину систему, яка дозволяє запустити керувати ігровим процесом. Розроблено текстову та ілюстраційну частини, які демонструють зовнішній вигляд закінченої гри. В результаті роботи було спроектовано програмний продукт, який відповідає всім поставленим вимогам, працює коректно та виконує поставлені задачі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Хокінг Д. Unity in Action: Multiplatform Game Development in C# / Джозеф Хокінг., 2015. – 352 с.
2. Фавлер М. UML Distilled: A Brief Guide to the Standard Object Modeling Language / Мартін Фавлер., 2003. – 208 с.
3. Unity User Manual (2019.3) / UI [Електронний ресурс] – 2020. – Режим доступу до ресурсу: <https://docs.unity3d.com/Manual/UISystem.html>
4. VIDE DIALOGUES. DIALOGUE SYSTEM AND EDITOR PLUGIN FOR UNITY3D [Електронний ресурс] – 2018. – Режим доступу до ресурсу: <https://videdialogues.wordpress.com/about/>
5. Lean Localization. User Manual [Електронний ресурс] – 2018. – Режим доступу до ресурсу: <http://carloswilkes.com/Documentation/LeanLocalization>
6. Cinemachine. Обучаючий матеріал [Електронний ресурс] – 2017. – Режим доступу до ресурсу: <https://learn.unity.com/tutorial/cinemachine>
7. Бонд Д. Г. Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C# / Джеремі Гібсон Бонд., 2017. – 1024 с.
8. Роберт Сесіл Мартін. Книга «Чиста архітектура», 2019 р.
9. Хокінг Д. Unity у дії. Мультиплатформна технологія на C#
10. Боулер Д. Розум без меж! Навчайся, зростає і живи без бар'єрів, 2022 р
11. Ентоні Девіс, Тревіс Батіст, Рассел Крейг, Райан Станкел. Розробка 3D-ігор у Unity 2020р.
12. Tim Nugent, Paris Buttfield-Addison, Jon Manning. Unity Game Development Cookbook: Essentials for Every Game, 1st Edition
13. Unity Terrain Manual [Електронний ресурси] – 2021р. – Режим доступу до ресурсу: <https://unity.com/resources/sculpting-terrain-unity>
14. Unity Neural AI [Електронний ресурси] – 2017р. – Режим доступу до ресурсу: <https://medium.com/@HolographicInterfaces/learn-how-to-implement-neat-ai-in-unity-157168eeae7e>

15. Abhishek Nandy, Manisha Biswas. Neural Networks in Unity: C# Programming for Windows 10 – 2018 p.
16. Robert Nystrom. Game Programming Patterns – 2020 p.
17. David Horache. Creating E-Learning Games with Unity – 2020 p.
18. Jiadong Chen. Game Development with Unity for .NET Developers: The ultimate guide to creating games with Unity and Microsoft Game Stack – 2022 p.
19. Patrick Felicia. Unity From Zero to Proficiency (Intermediate): A step-by-step guide to coding your first FPS in C# with Unity – 2016 p.
20. Patrick Felicia. Unity From Zero to Proficiency (Advanced): Create multiplayer games and procedural levels, and boost game performances: a step-by-step guide – 2016 p.
21. Unity Q-Learning [Электронный ресурс] – Режим доступа до ресурсу: <https://blog.unity.com/engine-platform/unity-ai-reinforcement-learning-with-q-learning>

ДОДАТКИ

Figure.cs:

```

using Diploma.Game.Models;
using Sirenix.OdinInspector;
using TMPPro;
using UnityEngine;
using Zenject;

namespace Diploma.Game.Global
{
    public abstract class Figure : MonoInstaller
    {
        public static string GetRandomID(int length = 25)
        {
            var symbols = new char[] { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h' };
            var result = string.Empty;
            for (var x = 0; x < length; x++)
                result += symbols[Random.Range(0, symbols.Length)];
            return result;
        }

        [ReadOnly]
        public string UniqueID;
        public string Title;
        public ArmyModel Army;
        public InventoryModel Inventory;

        public int TeamID =>
            Army.TeamID;

        [SerializeField] protected GameObjectContext goContext;
        [SerializeField] protected TextMeshPro titleText;

        [FoldoutGroup("Selection"), SerializeField] GameObject selectionHighlight;
        [FoldoutGroup("Selection"), SerializeField] GameObject altHighlight;

        [Inject] protected GlobalMap global;

        public bool SelectionActive
        {
            get => selectionHighlight != null ?
                selectionHighlight.activeInHierarchy :
                false;

            set => selectionHighlight?.SetActive(value);
        }

        public bool AltSelectionActive
        {
            get => altHighlight != null ?
                altHighlight.activeInHierarchy :
                false;

            set => altHighlight?.SetActive(value);
        }

        public override void Start()
        {
            UpdateTitle(Title);
        }

        public void RegenerateUniqueID()
        {

```

```

        if (UniqueID == string.Empty)
            UniqueID = GetRandomID();
    }

    public override void InstallBindings()
    {
        goContext.
            Container.
                Bind(GetType()).
                FromInstance(this).
                AsSingle();
    }

    public void UpdateTitle() =>
        UpdateTitle(Title);

    protected void UpdateTitle (string text)
    {
        titleText.text = $"{text}\n{Army.Units.Length}";
    }

    public void UpdateColor (Color teamColor)
    {
        titleText.color = teamColor;
    }

    public virtual void UnregisterFromData ()
    {
    }
}
}

```

GlobalPlayer.cs:

```

using Cysharp.Threading.Tasks;
using Diploma.Application.Services.UI;
using Zenject;

namespace Diploma.Game.Global.Players
{
    public sealed class GlobalPlayer : GlobalMapPlayerBase
    {
        [Inject] UIService ui;

        private GlobalPlayerControllerScreen screen;

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

            screen = await ui.Screens.CreateElement<GlobalPlayerControllerScreen>();
            screen.SetupPlayer(this);
        }

        public override async UniTask<bool> AnswerContactMode(Figure other)
        {
            return await screen.WaitAnser();
        }
    }
}

```

GlobalAIPlayer.cs:

```

using Diploma.Game.Global.Settlements;

```

```

using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.AI;
using Zenject;

namespace Diploma.Game.Global.Players
{
    public sealed class GlobalAIPlayer : GlobalMapPlayerBase
    {
        [Inject] UnitProductionLine units;
        [Inject] BuildingsData buildingsData;
        [Inject] GlobalMap global;

        public int TargetSquadsAmount = 3;
        public int DefaultSquadSize = 5;
        public int PatrolRadius = 15;
        public float MinTrainRateInSec = 10;
        public float MaxStayTime = 5;
        public float BuildingRate = 15;
        public float MinAssaultRate = 120;

        private SquadFigure targetFigure;
        private SquadFigure assaultFigure;

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

            CheckNeedStartAssault();
            if (targetFigure == null)
            {
                if (PlayerData.createdSquads != null &&
                    PlayerData.createdSquads.Count > 0)
                    FindAndFocusOnTarget();

                if (targetFigure == null &&
                    TryCreateNewSquad())
                    FindAndFocusOnTarget();

                PatrolOtherFigures();
            }
            else
            {
                if (targetFigure.Army.Units.Length >= DefaultSquadSize *
                    Mathf.Clamp(PlayerData.Settlements.Length, 1, 10))
                {
                    targetFigure = null;
                    return;
                }

                if (targetFigure.CurrentPoint != null)
                {
                    PatrolOtherFigures();
                }
                else
                {
                    var nearest = GetNearestCity(targetFigure, true);
                    if (nearest == null || canTrain == false)
                    {
                        if (canPatrolNext)
                        {
                            nearest = GetNearestCity(targetFigure, false);
                            if (nearest == null)
                            {

```

```

        //find and assault nearest or
        //find and assault city with low def
    }
    var targetPos = nearest.transform.position + new
Vector3(Random.insideUnitCircle.x * PatrolRadius, 0, Random.insideUnitCircle.y *
PatrolRadius);
    NavMesh.SamplePosition(targetPos, out var hit, 900, -1);
    targetFigure.MoveTo(hit.position,
        (res) =>
        {
            lastIdleTime = Time.time;
            genStayTime = Random.Range(0, MaxStayTime);
        });
    }
}
else
{
    targetFigure.MoveTo(nearest.transform, ComplectCurrentTarget);
}
}
}

UpdateBuildings();
}

#region Squads Controll
private void PatrolOtherFigures ()
{
    foreach (var squad in PlayerData.createdSquads)
    {
        if (squad == targetFigure ||
            squad.CurrentPoint != null ||
            squad == assaultFigure)
            continue;

        var near = Random.value > 0.5f ?
            GetRandomCity () :
            GetNearestCity(squad, false);
        var targetPos = near.transform.position + new
Vector3(Random.insideUnitCircle.x * PatrolRadius, 0, Random.insideUnitCircle.y *
PatrolRadius);
        NavMesh.SamplePosition(targetPos, out var hit, 900, -1);
        squad.MoveTo(hit.position);
    }
}

private void CheckNeedStartAssault ()
{
    if (canBeginAssault == false || assaultFigure != null)
        return;

    foreach (var squad in PlayerData.createdSquads)
    {
        if (squad.Army.Units.Length >= DefaultSquadSize)
        {
            assaultFigure = squad;

            var randomEnemy = GetRandomEnemy(Random.value > 0.5f);
            if (randomEnemy != null)
            {
                Debug.Log($"{squad.Title} begin assault on {randomEnemy.Title}
(Team: {randomEnemy.TeamID})");
                assaultFigure.MoveTo(randomEnemy.transform, (result) =>
                {
                    if (result)

```

```

        {
            global.InitializeBattle(ActionSceneName.City_First,
assaultFigure, randomEnemy);

            Debug.Log($"Begin battle with: {randomEnemy.Title}");
        }
        assaultFigure = null;
    });
    return;
}
}
}

private float lastAssaultTime;
private bool canBeginAssault =>
    Time.time - lastAssaultTime > MinAssaultRate;

private Figure GetRandomEnemy (bool squadOrSett)
{
    var enemiesID = new List<int>();
    for (var x = 0; x < global.Players.Length; x++)
    {
        var otherPlayer = global.Players[x];
        if (otherPlayer == this)
            continue;
        enemiesID.Add(x);
    }

    var result = global.Players[enemiesID[Random.Range(0,
enemiesID.Count)]]?.GetRandomFigure(squadOrSett);
    lastAssaultTime = Time.time;
    return result;
}

private float lastTrainTime;
private bool canTrain =>
    Time.time - lastTrainTime > MinTrainRateInSec;

private float lastIdleTime;
private float genStayTime;
private bool canPatrolNext =>
    Time.time - lastIdleTime > genStayTime;

private void ComplectCurrentTarget (bool achivedCity)
{
    if (achivedCity)
    {
        var city =
targetFigure.CurrentPoint.Target.GetComponent<SettlementFigure>();
        if (city != null)
        {
            city.CreateRecruitSquad();
            lastTrainTime = Time.time;
        }
    }
}

private void FindAndFocusOnTarget ()
{
    if (PlayerData.createdSquads.Count > 0)
    {
        foreach (var squad in PlayerData.createdSquads)
        {
            if (squad.Army.Units.Length < DefaultSquadSize)

```

```

        {
            targetFigure = squad;
            break;
        }
    }
}

private bool TryCreateNewSquad ()
{
    if (PlayerData.Armies.Length >= TargetSquadsAmount +
        PlayerData.ownedSettlements.Count)
        return false;

    if (PlayerData.EmptySettlementList == false) {
        foreach (var sett in PlayerData.ownedSettlements)
        {
            if
            (sett.Inventory.CanTakePrice(units[Models.UnitRank.Recrut].Needed))
            {
                sett.CreateRecruitSquad();
                return true;
            }
        }
    }

    return false;
}
#endregion

private SettlementFigure GetRandomCity ()
{
    return PlayerData.ownedSettlements[Random.Range(0,
        PlayerData.ownedSettlements.Count)];
}

private SettlementFigure GetNearestCity (SquadFigure targetFigure, bool
includeRecruitPrice)
{
    SettlementFigure best = null;
    if (PlayerData.EmptySettlementList == false)
    {
        foreach (var sett in PlayerData.ownedSettlements)
        {
            if (includeRecruitPrice)
            {
                if
                (sett.Inventory.CanTakePrice(units[Models.UnitRank.Recrut].Needed) == false)
                {
                    continue;
                }
            }

            if (best == null)
                best = sett;
            else
            {
                var bestDist =
                    Vector3.Distance(targetFigure.transform.position, best.transform.position);
                var currDist =
                    Vector3.Distance(targetFigure.transform.position, sett.transform.position);
                if (currDist < bestDist)
                    best = sett;
            }
        }
    }
}

```

```

    }

    return best;
}

#region Settlements Controller
private float lastBuildingTime;
private bool canBuildByTimer =>
    Time.time - lastBuildingTime > BuildingRate;

private void UpdateBuildings ()
{
    if (canBuildByTimer == false)
        return;

    for(var x = 0; x < PlayerData.ownedSettlements.Count; x++)
    {
        var sett = PlayerData.ownedSettlements[x];
        if(sett.Buildings.HasAnyFreePlace(out _) &&
            sett.Buildings.HasBuildingProcess() == false)
        {
            if (sett.Buildings.HasBuilding("Houses") == false)
            {
                TryBuild(sett, "Houses");
            }
            else
            {
                var variants = buildingsData.Items.Where((x) =>
x.CanBuildIntoVillage && sett.IsVillage && sett.Buildings.CountBuildings(x.StorageID
< 2).ToArray();
                if (variants.Length > 0)
                {
                    var randomBuildID = variants[Random.Range(0,
variants.Length)];
                    TryBuild(sett, randomBuildID.StorageID);
                }
            }
        }
    }
}

private bool TryBuild (SettlementFigure settlement, string buildingID)
{
    lastBuildingTime = Time.time;
    if (settlement.Buildings.HasAnyFreePlace(out var slot) &&
        settlement.CanBeginBuilding(slot, buildingID))
    {
        Debug.Log($"Bot {name} begin build: {settlement.Title},
{buildingID}");
        settlement.Buildings.TryBuild(slot, buildingID);
        settlement.Inventory.TakePrice(buildingsData[buildingID].Price);
        return true;
    }

    return false;
}
}
#endregion
}

```

TrainingMode.cs:

```

using Diploma.Application.Services.UI;
using Diploma.Game.Models;
using System.Collections.Generic;
using Unity.Burst.Intrinsics;

```

```

using UnityEngine;
using Zenject;

namespace Diploma.Game.Modes
{
    public class TrainingMode : ModeBaseBehaviour
    {
        public const float MAX_TRAINING_SEC = 300;

        [Inject] UIService ui;

        [Range(0, MAX_TRAINING_SEC)] public float EpochTime = 5;
        [SerializeField] float physicsSimulateStep = 0.02f;

        public int Epoch { get; private set; }
        public List<GameObject> AutoclearObjects = new List<GameObject>();

        private float lastEpochTime;
        private Transform enemySP;
        private Transform playerSP;

        public ArmyModel LeftSide;
        public ArmyModel RightSide;

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

            playerSP = GameObject.Find(LEFT_SP_NAME).transform;
            enemySP = GameObject.Find(RIGHT_SP_NAME).transform;

            PrepareLeftTeam();
            if (leftTeam.Load() == false)
                leftTeam.RandomizeGenerations();

            var dupArmy = StartData.LeftSide;
            dupArmy.TeamID = 1;
            PrepareTeam(dupArmy, enemySP);
            if(rightTeam.Load() == false)
                rightTeam.RandomizeGenerations();

            BeginNewEpoch();
            Physics.simulationMode = SimulationMode.Script;

            var screen = await ui.Screens.CreateElement<TrainingScreen>();
            screen.Mode = this;
        }

        protected override void OnSimulateTick()
        {
            Physics.Simulate(physicsSimulateStep);

            if(Teams.Count > 0)
            {
                if(timeFromLastEpoch >= EpochTime)
                {
                    leftTeam.Evolve(Epoch);
                    leftTeam.Save();
                    rightTeam.Evolve(Epoch);

                    BeginNewEpoch();
                }
            }

            base.OnSimulateTick();
        }
    }
}

```

```

    }

    private float timeFromLastEpoch =>
        Time.time - lastEpochTime;

    private void BeginNewEpoch ()
    {
        for(var x = AutoclearObjects.Count - 1; x >= 0; x--)
            if (AutoclearObjects[x] != null)
                Destroy(AutoclearObjects[x]);
        AutoclearObjects.Clear();

        Epoch++;
        lastEpochTime = Time.time;
    }

    private void OnDestroy()
    {
        Physics.simulationMode = SimulationMode.FixedUpdate;
        Time.timeScale = 1;
    }

    protected override void HandleControllValues(RaycastHit pos, Vector3 dir)
    {
        base.HandleControllValues(pos, dir);
        if (Controlled == NowControlled.MySpawnPoint)
            SetMySpawnPoint(pos.point, dir);
        else if (Controlled == NowControlled.EnemySpawnPoint)
            SetEnemySpawnPoint(pos.point, dir);
    }

    public void SetMySpawnPoint (Vector3 pos, Vector3 dir)
    {
        playerSP.transform.position = pos;
        playerSP.transform.rotation = Quaternion.LookRotation(dir);

        var deployment = GetDeploymentByType(LeftSide.Deployment);
        for (var x = 0; x < Teams[0].Units.Length; x++)
        {
            var data = deployment.CalculateUnitPosition(Teams[0].Units.Length,
playerSP);

            Teams[0].Units[x].InitialPos = data.Item1;
            Teams[0].Units[x].InitialRot = data.Item2;
        }
    }

    public void SetEnemySpawnPoint(Vector3 pos, Vector3 dir)
    {
        enemySP.transform.position = pos;
        enemySP.transform.rotation = Quaternion.LookRotation(dir);

        var deployment = GetDeploymentByType(RightSide.Deployment);
        for (var x = 0; x < Teams[1].Units.Length; x++)
        {
            var data = deployment.CalculateUnitPosition(Teams[1].Units.Length,
enemySP);

            Teams[1].Units[x].InitialPos = data.Item1;
            Teams[1].Units[x].InitialRot = data.Item2;
        }
    }
}

```

GameResultScreen.cs:

```

using Diploma.Application;

```

```

using Diploma.Application.Services.UI.Elements;
using Diploma.Game.Models;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace Diploma.Game.Modes
{
    [AssetPath("UI/Screens/Result")]
    public class GameResultScreen : UIScreen
    {
        [SerializeField] SideInfo left;
        [SerializeField] SideInfo right;
        [SerializeField] GameObject winTitle;
        [SerializeField] GameObject defeatTitle;

        public void SetupInfo (bool isPlayerWin)
        {
            winTitle.SetActive(isPlayerWin);
            defeatTitle.SetActive(isPlayerWin == false);
            left.ShowInfo(ModeBaseBehaviour.ResultData.Left);
            right.ShowInfo(ModeBaseBehaviour.ResultData.Right);
        }

        public void BackToGlobalMap ()
        {
            SceneManager.LoadScene("GlobalMap");
            UI.Screens.RemoveElement<GameResultScreen>();
        }

        [System.Serializable]
        public class SideInfo
        {
            [SerializeField] TextMeshProUGUI title;
            [SerializeField] RectTransform parent;
            [SerializeField] TextMeshProUGUI linePrefab;

            public void SetTitle (string text)
            {
                title.text = text;
            }

            public void AddText (string text)
            {
                var instance = Instantiate(linePrefab, parent);
                instance.text = text;
            }

            public void ShowInfo (ArmyModel army)
            {
                foreach(var unit in army.Units)
                {
                    AddText($"[{unit.Rank.ToString()[0]}] {unit.Name}:
{(unit.WasKilled ? "dead" : "+" + (int)unit.Experience)}");
                }
            }
        }
    }
}

```

GlobalMap.cs:

```

using Diploma.Application.Services.UI;
using Diploma.Game.AI;
using Diploma.Game.Global.Players;
using Diploma.Game.Global.Settlements;

```

```

using Diploma.Game.Modes;
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using UnityEngine;
using UnityEngine.SceneManagement;
using Zenject;

namespace Diploma.Game.Global
{
    public class GlobalMap : MonoInstaller
    {
        [Inject] UIService ui;

        public const int MAX_TIMESCALE = 15;
        public GlobalMapData Data;
        public GlobalMapPlayerBase[] Players;
        public float AutoSaveRate = 15;

        public float Timescale
        {
            get => Time.timeScale;
            set => Time.timeScale = Mathf.Clamp(value, 0, MAX_TIMESCALE);
        }

        private long lastSaveTime;

        public GlobalMapPlayerBase this[int teamID]
        {
            get
            {
                foreach (var team in Players)
                {
                    if (team.PlayerData.TeamID == teamID)
                        return team;
                }

                return null;
            }
        }

        public Color GetTeamColor(int teamID)
        {
            for (var x = 0; x < Players.Length; x++)
                if (Players[x].PlayerData.TeamID == teamID)
                    return Players[x].TeamColor;

            return Color.white;
        }

        public override void InstallBindings()
        {
            Container.
                Bind<GlobalMap>().
                FromInstance(this).
                AsSingle().
                NonLazy();
        }

        public override void Start()
        {
            lastSaveTime = DateTime.Now.Ticks;

            if (HasSave())
            {
                Load();
                HandleLastBattle();
            }
        }
    }
}

```

```

    }
    else
    {
        foreach (var player in Players)
            player.FirstInit();
    }
}

private void Update()
{
    CheckAutosave();

    if (Input.GetKeyDown(KeyCode.F5))
        Save();
    else if (Input.GetKeyDown(KeyCode.F6))
        Load();
}

private void CheckAutosave()
{
    var lastSaveDate = new DateTime(lastSaveTime);
    var diff = DateTime.Now - lastSaveDate;
    if (diff.TotalSeconds > AutoSaveRate)
    {
        Save();
        lastSaveTime = DateTime.Now.Ticks;
    }
}

#region IO
public void Save()
{
    Debug.Log($"Saving");
    Data.Ticks = EnviroSkyMgr.instance.GetDateAsDateTime().Ticks;
    using (var fs = File.Open(WorldSavePath(), FileMode.OpenOrCreate))
    {
        var bf = new BinaryFormatter();
        bf.Serialize(fs, Data);
    }

    foreach (var player in Players)
        player.Save();
    Debug.Log($"Saving completed");
}

public void Load()
{
    using (var fs = File.Open(WorldSavePath(), FileMode.Open))
    {
        var bf = new BinaryFormatter();
        Data = (GlobalMapData)bf.Deserialize(fs);
        EnviroSkyMgr.instance.SetTime(new DateTime(Data.Ticks));
    }

    foreach (var player in Players)
        if (player.HasSave())
        {
            //player.ClearCurrentSquads();
            player.Load();
        }
}

public bool HasSave() =>
    File.Exists(WorldSavePath());

```

```

        private string WorldSavePath() =>
            UnityEngine.Application.persistentDataPath + "/World" +
NeuralNetwork.NN_FILE_EXTENSION;
        #endregion

        private void HandleLastBattle()
        {
            if (ModeBaseBehaviour.HasBattleInfo == false ||
                ModeBaseBehaviour.StartData.IsTraining)
                return;

            var resultData = ModeBaseBehaviour.ResultData;
            var startData = ModeBaseBehaviour.StartData;

            if (resultData != null)
            {
                var leftTeamID = resultData.Left.TeamID;
                var rightTeamID = resultData.Right.TeamID;
                var leftFigure = FindFigure(leftTeamID,
resultData.Left.FigureUniqueID);
                var rightFigure = FindFigure(rightTeamID,
resultData.Right.FigureUniqueID);

                if(resultData.IsLeftWin)
                {
                    if(rightFigure is SettlementFigure)
                    {
                        rightFigure.UnregisterFromData();
                        var sett = rightFigure as SettlementFigure;

                        this[leftTeamID].PlayerData.AddSettlement(sett);
                        sett.UpdateColor(this[leftTeamID].TeamColor);
                        sett.Army.Units = new Models.UnitModel[0];
                    }
                    else
                    {
                        this[rightTeamID].PlayerData.RemoveSquad(rightFigure as
SquadFigure);
                    }
                }
                else
                {
                    leftFigure.UnregisterFromData();
                }

                ModeBaseBehaviour.ClearBattleDatas();
            }

            Save();
        }

        private Figure FindFigure(int team, string figureUniqueID)
        {
            foreach(var player in Players)
            {
                if(player.PlayerData.TeamID == team)
                {
                    for(var x = 0; x < player.PlayerData.createdSquads.Count; x++)
                    {
                        var element = player.PlayerData.createdSquads[x];
                        if (element.Army.FigureUniqueID == figureUniqueID)
                            return element;
                    }

                    for(var x = 0; x < player.PlayerData.ownedSettlements.Count; x++)
                    {

```

```

        var element = player.PlayerData.ownedSettlements[x];
        if (element.UniqueID == figureUniqueID)
            return element;
    }
}
return null;
}

public async void InitializeBattle (ActionSceneName scene, Figure attacker,
Figure defender, bool forceManual = false)
{
    Save();
    Timescale = 0;

    var leftPlayer = this[attacker.TeamID];
    var rightPlayer = this[defender.TeamID];

    var focusPoint = attacker.transform.position;
    FocusIfPlayer(attacker.TeamID, focusPoint);
    FocusIfPlayer(defender.TeamID, focusPoint);

    void FocusIfPlayer (int teamID, Vector3 point)
    {
        if (this[teamID] is GlobalPlayer)
            this[teamID].transform.position = point;
    }

    var leftPlayerAnswer = forceManual ? false : await
leftPlayer.AnswerContactMode(defender);
    var rightPlayerAnswer = forceManual ? false : await
rightPlayer.AnswerContactMode(attacker);

    if (defender is NavigableFigure)
        (defender as NavigableFigure).EndCurrent(false);

    Timescale = 1;

    if(leftPlayerAnswer && rightPlayerAnswer)
    {
        var leftForce = attacker.Army.CalculateForce();
        var rightForce = defender.Army.CalculateForce();

        var rightIsSett = defender is SettlementFigure;
        var diff = Mathf.Abs(leftForce - rightForce);
        var max = Mathf.Max(leftForce, rightForce);
        var percent = (float)diff / max;
        if(leftForce > rightForce)
        {
            defender.UnregisterFromData();
            if(rightIsSett)
                leftPlayer.PlayerData.AddSettlement(defender as
SettlementFigure);

            attacker.Army.Damage(percent);
            attacker.UpdateTitle();
            defender?.UpdateColor(leftPlayer.TeamColor);
        }
        else
        {
            attacker.UnregisterFromData();
            defender.Army.Damage(percent);
            defender.UpdateTitle();
        }
    }
}

```

```

                Debug.Log($"Autobattle end ({attacker.TeamID}, {attacker.Title} vs
{defender.TeamID}/{defender.Title}): {(leftForce > rightForce ? "attack sucess" :
"attack failed")}");
            }
            else
            {
                ui.Screens.RemoveElement<GlobalPlayerControllerScreen>();
                ModeBaseBehaviour.StartData = GenerateData(scene, attacker, defender);
                RunManualBattle(scene);
            }
        }

        public void InitTraining (Figure squad, SettlementFigure settlement)
        {
            InitializeBattle(ActionSceneName.Training, squad, settlement, true);
        }

        private void RunManualBattle (ActionSceneName scene)
        {
            SceneManager.LoadScene(scene.ToString());
        }

        private ModeStartData GenerateData (ActionSceneName scene, Figure attacker,
Figure defender)
        {
            var result = new ModeStartData
            {
                SceneName = scene.ToString(),
                IsTraining = scene == ActionSceneName.Training,
                LeftTitle = attacker.Title,
                LeftSide = attacker.Army,
                RightTitle = defender.Title,
                RightSide = defender.Army,
                IsRealPlayerLeft = attacker.TeamID == 0
            };
            return result;
        }
    }

    [Serializable]
    public class GlobalMapData
    {
        public long Ticks;
    }

    public enum ActionSceneName
    {
        City_First,
        City_Second,
        Village,
        Forest,
        Field,
        Training
    }
}

```

NeuralUnit.cs:

```

using Diploma.Game.Animations;
using Diploma.Game.Equipement;
using Sirenix.OdinInspector;
using UnityEngine;
using RaycastPro.RaySensors;
using System.Collections.Generic;

```

```

using UnityEngine.Events;

namespace Diploma.Game.AI
{
    public abstract class NeuralUnit : RBMovableNeuralAgent
    {
        public const int UNIT_MAX_HEALTH = 100;
        public const int CIVILIAN_TEAM_ID = -1;

        public int Health = UNIT_MAX_HEALTH;
        public int Team;

        [FoldoutGroup("Animations")]
        public UnitAnimation Anims;
        [FoldoutGroup("Animations")]
        public float AnimValueSmooth = 0.15f;

        [FoldoutGroup("Vision")]
        public RadialRay VisionRays;
        [FoldoutGroup("Vision")]
        public bool WriteDistanceInInputVision = true;

        public Vector3 InitialPos;
        public Vector3 InitialRot;

        public UnityEvent OnKilled;

        public bool IsDead =>
            Health <= 0;

        protected virtual void Start ()
        {
            InitialPos = transform.position;
            InitialRot = transform.eulerAngles;
        }

        public override void Move(float axisX, float axisY)
        {
            base.Move(axisX, axisY);

            Anims.Moving = Mathf.MoveTowards(Anims.Moving, axisX, AnimValueSmooth);
        }

        protected override List<double> Inputs()
        {
            var result = base.Inputs();

            for(var x = 0; x < VisionRays.raycastHits.Count; x++)
            {
                var ray = VisionRays.raycastHits[x];
                var targetInVision = GetFromGO(ray.collider);
                result.Add((int)targetInVision);

                if (WriteDistanceInInputVision)
                {
                    var distance = targetInVision != DetectedTargetType.Empty ?
                        ray.distance :
                        0;
                    result.Add(distance);
                }
            }

            return result;
        }
    }
}

```

```

protected bool HasIntoView (DetectedTargetType target, out GameObject result)
{
    result = null;
    for(var x = 0; x < VisionRays.raycastHits.Count; x++)
    {
        var type = GetFromGO(VisionRays.raycastHits[x].collider);
        if (type == target)
        {
            result = VisionRays.raycastHits[x].collider.gameObject;
            return true;
        }
    }

    return false;
}

protected virtual DetectedTargetType GetFromGO (Collider result)
{
    if (result != null)
    {
        if (result.TryGetComponent<NeuralUnit>(out var otherUnit))
        {
            if (otherUnit.Team == CIVILIAN_TEAM_ID)
                return DetectedTargetType.Civilian;
            else if (otherUnit.Team == Team)
                return DetectedTargetType.Freind;
            else
                return DetectedTargetType.Enemy;
        }
        else
            return DetectedTargetType.Scene;
    }
    else
    {
        return DetectedTargetType.Empty;
    }
}

public virtual void DamageFromProjectile (Projectile projectile)
{
    if (IsDead)
        return;

    Health -= projectile.Owner.Damage;
    if(IsDead)
        OnDead();
}

protected virtual void OnDead ()
{
    gameObject.SetActive(false);
    OnKilled?.Invoke();
    EnabledAIControll = false;
}

public virtual void Restore ()
{
    SetScore(0);
    Health = UNIT_MAX_HEALTH;
    EnabledAIControll = true;
    gameObject.SetActive(true);

    transform.position = InitialPos;
    transform.rotation = Quaternion.Euler(InitialRot);
}

```

```

    }

    public enum DetectedTargetType
    {
        Civilian = -2,
        Empty = -1,
        Freind = 0,
        Enemy = 1,
        Scene = 2,
    }
}

```

RBMovableNeuralAgent.cs:

```

using System.Collections.Generic;
using UnityEngine;

namespace Diploma.Game.AI
{
    public abstract class RBMovableNeuralAgent : NeuralAgentBehaviour
    {
        public float SpeedRotate = 3f;
        public float SpeedMove = 3f;

        protected Rigidbody body;

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

            body = GetComponent<Rigidbody>();
        }

        protected override List<double> Inputs()
        {
            var result = base.Inputs();
            result.Add(body.velocity.magnitude / SpeedMove);
            return result;
        }

        public virtual void Move(float axisX, float axisY)
        {
            axisX = Mathf.Clamp(axisX, -1, 1);
            axisY = Mathf.Clamp(axisY, -1, 1);

            var velocity = transform.right * axisY + transform.forward * axisX;
            velocity.y = 0;
            body.velocity = velocity * SpeedMove * Time.deltaTime;
        }

        public virtual void Rotate(float axisX)
        {
            //body.angularVelocity = Vector3.up * axisX * speedRotate;
            transform.rotation *= Quaternion.Euler(0, axisX * SpeedRotate *
Time.deltaTime, 0);
        }

        protected override void ImplementOutput(double[] output)
        {
            Move((float)output[(int)NeuralAgentOutputOption.MoveForward] - 0.5f, 0);
            Rotate((float)output[(int)NeuralAgentOutputOption.Rotate] - 0.5f);
        }
    }
}

```

TacticalNeuralUnit.cs:

```

using Diploma.Game.Equipement;
using Diploma.Game.Modes;
using Sirenix.OdinInspector;
using System.Collections.Generic;
using UnityEngine;
using Zenject;

namespace Diploma.Game.AI
{
    public abstract class TacticalNeuralUnit : NeuralUnit
    {
        [Inject] protected ModeBaseBehaviour mode;

        [FoldoutGroup("Operation")] public Transform CenterOperationZone;
        [FoldoutGroup("Operation")] public float CorrectZoneDistance = 3f;
        [FoldoutGroup("Neural Rewards-Penalty")] public float OperationFollowReward =
.25f;

        public EquipementController Equipement;

        public float DistanceToCenterZone =>
            mode.PriorityEnabled ?
            Vector3.Distance(mode.GetPriorityPoint(), transform.position) :
            0;

        protected override List<double> Inputs()
        {
            var result = base.Inputs();
            result.Add(DistanceToCenterZone/CorrectZoneDistance);
            return result;
        }

        public override double CalculateFitness()
        {
            if(CenterOperationZone == null)
                return base.CalculateFitness();
            else
            {
                var result = base.CalculateFitness();
                if (DistanceToCenterZone <= CorrectZoneDistance)
                    result += OperationFollowReward;
                else
                    result -= (DistanceToCenterZone / CorrectZoneDistance);
                return result;
            }
        }
    }
}

```

BattleMode.cs:

```

using Diploma.Application.Services.UI;
using Diploma.Game.AI;
using Diploma.Game.Cameras;
using Diploma.Game.Equipement;
using Diploma.Game.Models;
using Sirenix.OdinInspector;
using UnityEngine;
using UnityEngine.Events;
using Zenject;

namespace Diploma.Game.Modes
{
    public class BattleMode : ModeBaseBehaviour
    {
        [Inject] UIService ui;
    }
}

```

```

public UnityEvent<bool> OnWinnerSetted;
[ShowInInspector, ReadOnly]
public bool IsGameEnded { get; private set; }

[SerializeField]
private GameObject crosshair;
private BattleScreen screen;

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

    PrepareUI();
    PrepareLeftTeam();
    PrepareRightTeam();
}

private async void PrepareUI ()
{
    screen = await ui.Screens.CreateElement<BattleScreen>();
    screen.SetupMode(this);
}

protected override NeuralUnit PrepareUnit(ArmyModel army,
                                             int unitIdx,
                                             Vector3 pos,
                                             Vector3 lookDir,
                                             Transform parent = null)
{
    var result = base.PrepareUnit(army, unitIdx, pos, lookDir, parent);
    result.OnKilled.AddListener(() => UnitDead(result));
    return result;
}

private void UnitDead (NeuralUnit unit)
{
    if (unit == CurrentControlledUnit)
        OutOfControll();

    if (leftTeam.TeamID == unit.Team)
        leftTeam.Army.Units[unit.UnitArmyIdx].WasKilled = true;

    if(rightTeam.TeamID == unit.Team)
        rightTeam.Army.Units[unit.UnitArmyIdx].WasKilled = true;

    if (leftTeam.LiveUnitsCount <= 0)
    {
        GameEnded(false);
    }
    else if (rightTeam.LiveUnitsCount <= 0)
    {
        GameEnded(true);
    }
}

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

    if (CurrentControlledUnit)
    {
        ControllCurrent();
        if(Input.GetKeyDown(KeyCode.Escape))
            OutOfControll();
    }
}

```

```

    }
}

private void ControllCurrent ()
{
    var fwd = Input.GetAxis("Vertical");
    var side = Input.GetAxis("Horizontal");

    CurrentControlledUnit.Move(fwd, side);
    CurrentControlledUnit.Rotate(Input.GetAxis("Mouse X"));
    var equipped =
CurrentControlledUnit.GetComponent<EquipementController>().Contains[0];
    if (Input.GetMouseButtonDown(0))
        equipped.TryUse();

    crosshair.transform.LookAt(Camera.main.transform);
    crosshair.transform.position = equipped.GetForwardPoint();
}

private void GameEnded (bool winnerLeftSide)
{
    if (IsGameEnded)
        return;

    var result = new ModeResultData()
    {
        IsLeftWin = winnerLeftSide,
        Left = null,
        Right = null
    };
    foreach (var team in Teams)
    {
        var targetPlace = result.Left == null;
        foreach (var unit in team.Units)
        {
            team.Army.Units[unit.UnitArmyIdx].Experience += unit.Experience;
            team.Army.Units[unit.UnitArmyIdx].LastBattleExp = unit.Experience;
            team.Army.Units[unit.UnitArmyIdx].WasKilled = unit.IsDead;
        }

        if (targetPlace)
            result.Left = team.Army;
        else
            result.Right = team.Army;
    }

    ResultData = result;
    ModeSimulation = false;

    OnWinnerSetted?.Invoke(winnerLeftSide);
    IsGameEnded = true;
}

protected override void HandleControllValues(RaycastHit pos, Vector3 dir)
{
    base.HandleControllValues(pos, dir);

    if (Controlled == NowControlled.DirectControll &&
        pos.collider.TryGetComponent<NeuralUnit>(out var unit) &&
        unit.Team == 0)
    {
        TakeControll(unit);
        HandleCurrentControllType = false;
    }
}

```

```

public NeuralUnit CurrentControlledUnit;

private void TakeControl (NeuralUnit unit)
{
    CurrentControlledUnit = unit;
    CurrentControlledUnit.EnabledAIControl = false;

    EnabledTPSCameras(true);
    screen.SetupControl();
    crosshair.gameObject.SetActive(true);
}

private void OutOfControl ()
{
    if(CurrentControlledUnit)
    {
        CurrentControlledUnit.EnabledAIControl = true;
    }

    screen.ClearControl();
    crosshair.gameObject.SetActive(false);
    EnabledTPSCameras(false);
    HandleCurrentControlType = true;
    CurrentControlledUnit = null;
}

private void EnabledTPSCameras (bool enable)
{
    Camera.main.GetComponentInParent<StrategyCamera>().enabled = !enable;
    var tpsCamera = Camera.main.GetComponentInChildren<ThirdPersonCamera>();
    tpsCamera.Target = CurrentControlledUnit.transform;
    tpsCamera.enabled = enable;
}
}
}

```