Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“___ ” _____ 2021 р.

КОМПЛЕКСНА ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи візуалізації покрокової стратегічної гри”

Виконав здобувач вищої освіти
ІІ курсу, групи ____КІ-20 М_(1,4)_____
ОПП «Комп’ютерна інженерія»
спеціальності 123«Комп’ютерна інженерія»
_____ Железняк Б. Ю.
«____» _____ 2021р.

Керівник проекту
кандидат технічних наук, доцент
_____ Олександр ДРЄЄВ
«____» _____ 2021р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
“____” _____20__року

З А В Д А Н Н Я

КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Железняку Богдану Юрійовичу

(прізвище, ім’я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи візуалізації покрокової стратегічної гри*

Керівник роботи *Дрєєв Олександр Миколайович, доцент, доцент кафедри КбПЗ*

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу “__”_____20__року №___

2. Строк подання студентом роботи до захисту *22.12.2021 р.*

3. Мета та завдання випускної кваліфікаційної роботи *Метою даної роботи є дослідження та програмна реалізація системи візуалізації, що реалізовано за допомогою розробки програмного забезпечення покрокової стратегічної гри та створення методу обрання системи комп’ютерної візуалізації з врахуванням вимог що до розв’язуваних задач*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Реалізація роботи. Розрахунки та експериментальні дані, що підтверджують вірність проектних і програмних рішень.

5. Впровадження системи в промислову експлуатацію.

6. Наукова новизна.

7.Економічна ефективність розробленої програми.

8.Заходи з охорони праці та техніки безпеки.

9. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатка	*4 аркуша*
Наукова новизна	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В., к.т.н., доцент	25.10.2021	12.11.2021
Охорона праці	Оришака О.В., к.т.н., доцент	04.11.2021	20.11.2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2021 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2021 р.	
3.	Розробка моделі компонента	20.10.2021 р.	
4.	Розробка структур даних	25.10.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2021 р.	
6.	Програмування алгоритмів	10.11.2021 р.	
7.	Розрахунок економічної ефективності	13.11.2021 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2021 р.	
9.	Оформлення ПЗ	17.11.2021 р.	
10.	Попередній захист роботи	03.12.2021 р.	

Дата видачі завдання
«___»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«___»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Железняка Б. Ю. Дослідження та програмна реалізація системи візуалізації покрокової стратегічної гри. 123 Комп'ютерна інженерія . Центральноукраїнський національний технічний університет. Кропивницький 2021.

У даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти ставиться завдання дослідити та розробити програмне забезпечення системи візуалізації покрокової стратегічної гри, що дозволяє брати участь у битві двом та більше гравцям та надає можливість отримувати задоволення від боїв із штучним інтелектом.

В роботі був реалізований комплекс дій з процесу проектування та розробки програмного забезпечення. Розглянуто проблему підсилення об'єктивної складової в процесі обрання фреймворку, зокрема системи комп'ютерної візуалізації при створенні архітектури програмного забезпечення.

Були розглянуті основні типи та характеристики існуючих покрокових стратегічних ігор та проаналізована гра, яка побудована на основі генератора випадкових чисел.

В результаті запропоновано метод обрання системи візуалізації, який базується на аналізуванні задач до системи візуалізації з виведенням вимог до побудованого зображення. Показано застосовність багатокритеріальної оптимізації для відокремлення застосовних систем візуалізації та вибору найкращої з них.

Загальний обсяг роботи 115 сторінок, 27 рисунки, 67 бібліографічних найменувань.

Ключові слова: проектування, програмне забезпечення, прийняття рішення, комп'ютерна система візуалізації, покрокова стратегія, графічний рушій, юніти, бафи, гекси, гексогонна глобальна карта, шейдери,префаби, перки.

ABSTRACT

Zhelezniak B. Y. Research and software implementation of step-by-step strategic game visualization system. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2021.

In this final qualifying work for the second (master's) level of higher education, the task is to research and develop software for a step-by-step strategy game visualization system that allows two or more players to participate in battle and gives the opportunity to enjoy fighting with artificial intelligence.

A set of actions on the process of software design and development was implemented in the work. The problem of strengthening the objective component in the process of choosing a framework, in particular the computer visualization system when creating a software architecture, is considered.

The main types and characteristics of existing turn-based strategy games were considered and the game, which is based on a random number generator, was analyzed.

As a result, a method of choosing a visualization system is proposed, which is based on the analysis of tasks to the visualization system with the derivation of requirements for the constructed image. The applicability of multicriteria optimization for the separation of applicable visualization systems and the selection of the best of them is shown.

The total volume of the work is 115 pages, 27 figures, 67 bibliographic titles.

Keywords: design, software, decision making, computer visualization system, step-by-step strategy, graphic engine, units, buffs, hexes, hexagonal global map, shaders, prefabs, perks.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

ОД – очки дії

AI – штучний інтелект

AP – очки дії

HP – очки здоров’я

SEAL(Software-optimized Encryption Algorithm.) - криптографічний алгоритм

UI (user interface) інтерфейс користувача

ЛКМ – ліва кнопка комп’ютерної миші або його аналог

ПКМ – права кнопка комп’ютерної миші або його аналог

ВСТУП

На сьогодні неможливо уявити повсякденне життя без постійного притоку інформацій. Для задоволення постійної потреби в інформації використовують такі сучасні засоби мультимедіа як: текстова інформація, телебачення, інтернет, та інше.

Мультимедіа – це комбінування різноманітних форм представлення інформації на одному носії. Наприклад, текстовому анімаційному, графічному. Поділяється же мультимедіа на лінійну та нелінійну.[1]

До лінійного способу можна віднести перегляд інформації з різних джерел, на які ми не можемо вплинути, наприклад, при перегляді кіно. Людина, що переглядає кіно (інформацію), не може жодним чином вплинути на його зміст, звісно якщо це не є інтерактивним кіном, що надає можливість впливати на сюжет, хоча насправді ми не впливає на таке кіно в цілому, а впливаємо лише на свій перегляд, обираючи між декількома варіантами продовження, ми не можемо в них в будь який час змінити положення предметів або персонажа в кіно так як ми звикли при проходженні ігор, тому таке кіно все ж таки стоїть відносити до лінійного способу подання інформації, а не до нелінійного.

Нелінійний же спосіб подає інформацію так, щоб користувач міг брати участь у поданні інформації, взаємодіючі із засобами відображення мультимедіа. Гарним прикладом можуть слугувати комп’ютерні ігри, які дозволяють користувачу взаємодіючі зі світом гри, змінювати його.

Варто відзначити, що особливо в комп’ютерних іграх спосіб подання інформації досить сильно залежить від методів побудови зображення, саме від них залежить, як користувач буде «погружатися» в світ гри.

На сьогодні існує багато методів побудови цифрового зображення (растерізація) з геометричного завдання об’єктів сцени, властивостей

поверхонь цих об’єктів та параметрів освітлення в сцені. Також сюди входять і засоби створення сцен з врахуванням растрових шаблонів цифрового зображення (текстур). Всі ці методи є відмінними по якості створення зображення по критеріям стабільності, реалістичності, правильності а також мають колосальні відмінності в часі створення окремого кадру цифрової сцени. Все це безумовно впливає на швидкість та якість побудованого зображення, і різноманіття засобів та налаштувань систем візуалізації призводить до ускладнення процесу визначення засобів і налаштування системи візуалізації для вирішення конкретних задач.

В результаті маємо об’єктивне протиріччя, коли з одного боку маємо велику кількість комп’ютерних систем візуалізації, а з іншого є потреба в швидкому та правильному вибору конкретної системи візуалізації з врахуванням вимог що до візуалізації для вирішення конкретних задач. Для вирішення визначеного протиріччя було поставлено однієї з мети магістерської роботи, обрання системи комп’ютерної візуалізації, що задовільнить основним потребам для створення комп’ютерної гри.

Комп’ютерні ігри набули небувалої популярності та різноманіття за останні роки. Комп’ютерні ігри настільки багатогранні, що охоплюють не лише сферу розваг, але і сферу розвитку та освіти. Особливого визнання та зацікавленості заслуговують стратегічні ігри, що розвивають людину. Найхарактернішим піджанром стратегічних ігор можна віднести покровові стратегічні ігри, завдяки яким можна не тільки плавно розвивати стратегічне мислення, але і насолоджуватися потраченим часом.

Метою даної роботи є збір теоретичних відомостей про сучасні комп’ютерні ігри, жанри ігор та їх піджанри, зосереджуючи увагу на стратегічних іграх та їх піджанрах, створити метод обрання системи комп’ютерної візуалізації з врахуванням вимог що до розв’язуваних задач та за допомогою аналізу популярних ігор, розробити архітектуру гри.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Моєю задачею під час роботи над магістерського дипломною роботою була розробка програмного забезпечення системи візуалізації покрокової стратегічної гри, що дозволяє брати участь у битві двом та більше гравцям та надає можливість отримувати задоволення від боїв зі штучним інтелектом.

1.1 Призначення системи

Розроблена система призначена, перш за все, для використання у розвивальних цілях. На мою думку , майже кожна стратегічна гра, дозволяє користувачеві розвивати вміння швидко і якісно виконувати поставлену задачу, приймати оптимальні рішення в складних та нетипових ситуаціях та покращувати навички в розпізнанні та запам’ятовуванні об’єктів.

Ще в 1980-х роках було поставлено питання про вплив відеоігор на психічне та фізіологічне здоров’я людини в цілому. В ті роки була виявлена ігрова залежність, особливо, у дітей, внаслідок новизни та некоректного регулювання часу, відведеного на відеоігри. Вперше про шкоду відеоігор було заявлено після виступу міністра охорони здоров'я США Еверетта Купа 9 листопада 1982 року, коли він вказав на залежність дітей від комп’ютерних ігор.[2] За роки досліджень було встановлено, що надмірне захоплення відеоіграми та, згодом, комп’ютерними іграми, може перерости у хворобливу залежність, яка має як фізіологічні, так і емоційні симптоми.

На противагу шкідливому впливу відеоігор та комп’ютерних ігор можна протипоставити наочні приклади, що демонструють позитивний вплив. Так, використання контролерів, які передбачають активну рухову діяльність, розвиває рухові навички, підвищує мотивацію до занять фізичними вправами і поліпшує соціалізацію. Крім того, надані іграми моделі діяльності тренують у

дітей вміння шукати вирішення проблем, приймати оптимальні рішення, полегшують спілкування з однолітками, розвивають навички командної роботи, знімають стрес. Як у дітей, так і у підлітків та дорослих, відеоігри та комп'ютерні ігри покращують навички впізнавання об'єктів та орієнтування в просторі, робочу пам'ять та увагу.

Варто відзначити, що існують не тільки ігри, які були створенні з розважальною метою, а і ігри сугубо навчальні. Наприклад, використовуючи різнопланові симулятори можна навчитися керуванню реальними транспортними засобами. Рівним чином, навчальні ігри допомагають опанувати різноманітними науками, від гуманітарних до технічних. Тому, під час розробки програмного забезпечення системи візуалізації покрокової стратегічної гри, я хотів розробити систему, завдяки якій можна було б не тільки насолоджуватися грою, але і пасивно навчатися.

Отже, програмне забезпечення системи візуалізації покрокової стратегічної гри призначене для розвитку та навчанню користувачів, а саме, тренуванню навичок стратегічного мислення в різноманітних умовах, якісному розподілу наявних ресурсів для боротьби з опонентом (опонентами), корегуванню та пристосуванню до поточної ситуації.

1.2 Область застосування

На мою думку, о собливо доцільним буде застосувати запропоноване програмне забезпечення для дітей, які тільки починають знайомитися з стратегічними іграми. Це можна стверджувати тому, що в моїй грі немає візуальної демонстрації бойових дій між юнітами гравців, а є лише статистики, що відображає їх параметри. Таким чином, діти, які ще не готові для від критої чи прикритої демонстрації агресивного середовища, не пізнають агресії, але зможуть зробити перший психологічний крок у стратегічному розвитку.

Для кращої візуалізації стратегічного середовища, мною було обрана розробка саме покрокової стратегічної гри, щоб користувач, який лише знайомиться з жанром стратегій, міг не лише краще підготуватися до змін на глобальній карті та наступного кроку супротивника та і встигнути застосувати свої навички.

Власне глобальна карта гри є сценою бою, що може змінюватися за допомогою ручного редагування або шляхом встановлення коефіцієнту «порожнин» при автоматичному створені карти, який автоматично створює порожнини. Варто попередити, що користувачеві, який редагуватиме карту з використанням порожнин, потрібно бути особливо обережним, тому що може створитися ситуація неможливості проведення бойових дій.

Також, слід відзначити, що за рахунок різних «ящиків з бафом» стає можливим кардинально модифікувати ситуацію на полі бою, і тим самим, дозволити гравцям швидше навчатися і адаптуватися до різноманітних змін.

Відомо, що граючи в ігри, діти можуть тренувати вміння шукати вирішення різноманітних, нетипових проблем, приймати оптимальні рішення та підстроюватися під поточну ситуацію. Ігри командою полегшують спілкування з однолітками, розвивають навички колективної роботи та, на сам перед, ігри допомагають зняти стрес.

Варто підкреслити, що на думку педагогів та психологів та моє розуміння, обмежене проведення часу за різноманітними відеоіграми, особливо за іграми, що розвивають інтелектуальні властивості дитини, не є шкідливим способом проведення дозвілля, а навпаки, демонструє позитивний вплив на розвиток особистості.

Таким чином, внаслідок застосування моєї гри, при належному обмеженню часу проведення дітьми за грою та при якісному консультуванні, можливо досягти успіху у знятті шкоди від стресу та полегшенні спілкування між однолітками. З деяким часом можна буде спостерігати і значне підвищення навичок інтуїтивного вирішення проблем.

2. ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень при розробці програмного забезпечення системи візуалізації покрокової стратегічної гри

Основною характеристикою покрокових стратегічних ігор є дискретність ігрового процесу. Гра складається з фіксованих у часі моментів («кроків» або «ходів»), які завершуються тільки по команді гравця або автоматично при завершені ОД всіх членів команди гравця. Під час цих ходів гравець робить свої дії або дивиться за ходом супротивника. Варто пам’ятати, що сам хід в різних іграх є різним . Наприклад, один хід може відповідати проміжку у багато років у ігровому світі, за які гравець встигає впоратися з подіями в кожному місті імперії і віддати накази сотням військових загонів.

У більшості покрокових стратегій гравці роблять ходи один за одним, як в так ій класичній настільній грі , як шахи. Прикладами таких ігор є Sid Meier's Civilization і Heroes of Might and Magic, Battle for Wesnoth.

Існує тип стратегічних ігор, які зазвичай відносять до покрокових, але гравці в них роблять ходи одночасно. Іноді їх ще називають покроковими стратегіями в реальному часі або tick-based strategy (від англ. Tick - мітка). В таких іграх всі гравці роблять ігрові дії одночасно і в реальному часі. Такою грою є Sid Meier's Civilization, в якій гравці закінчують одночасно свої можливі або бажані дії. Після завершення дій, вони підтверджують, що готові до наступного ходу. Після того, як всі гравці просигналізували про завершення ходу, починається наступний хід. Ігри даного типу часто відносять до покрокових стратегій, тому що в основі ігрового процесу лежить «покроковість», незважаючи на те, що дії відбуваються одночасно.

Деякі стратегічні комп'ютерні ігри комбінують в собі елементи

покрокової стратегії і стратегії в реальному часі . Такі ігри, як Lords of the Realm від Impressions Games і Total War від Creative Assembly використовують покрокову механіку на рівні глобальної стратегії і управління, а самі тактичні битви відбуваються в реальному ча сі. Проте, результат боїв часто залежить від рішень прийнятих під час покрокової фази, і в цілому ігровий процес ближче до покрокових стратегій, ніж до стратегій в реальному часі. У багатьох «гібридних» іграх битви в реальному часі можуть бути пропущені, а їх результат розраховується автоматично. Але, нажаль, в тій же самій серії ігор Total War, на мою думку, автоматичне завершення битви призводить скоріше до поразки, чим до перемоги, якщо сили з обох сторін рівні, тому AI тактичних битв в подібних іграх є слабкою частиною. Але незважаючи на низьку ефективність автобою, такого типу ігри все ще є досить популярними.

Одними із кращих зразків покрокових стратегічних ігор, на мою думку, є наступні представники:

- Ancient Chronicles;
- Civilization;
- Heroes of Might and Magic;
- Panzer General;
- Disciples II: Dark Prophecy;
- Total War;
- Battle for Wesnoth.

Кожна із перелічених ігор, внесла важливий вклад у розвиток покрокової індустрії стратегічних комп'ютерних ігор. Я перелічив лише маленьку частину ігор, слід зауважити, що це ще не всі представники покрокових стратегічних ігор, які допомогли гравцям пережити безліч неймовірних вражень.

Вважаю доцільним детально проаналізувати і розглянути одну з самих улюблених ігор, що містить в собі піджанр покрокових стратегічних ігор,

Battle of Wesnoth. Вона охоплює в собі неймовірну кількість різноманітних компаній, які були створені як розробниками, так і звичайними користувачами. Battle of Wesnoth – це гра, що була побудована на основі генератора випадкових чисел. В цьому плані, механіка битви та механіка захисту є цікавою ідеєю, тому я сподіваюсь, що зможу довести свою гру до цього рівня.

Battle of Wesnoth – безкоштовна покрокова стратегія з відкритим кодом, яка легко заткне за пояс багатьох платних ігор.Унікальність даної гри в тому, що незважаючи на безкоштовність, вона відноситься до ігор, які відкрили свій вихідний код, що дало користувачам можливість розробити власну компанію. У даної гри налічується понад 20 різноманітних рас, в кожній з яких є свої бійці, багато з яких мають свої бонуси в залежності від місцевості або внутрішньо ігрового часу. Також цікаво, що в Battle of Wesnoth є власна система прокачування, яка орієнтована на просування класу юніта вверх. Наприклад, якщо був у нас боєць списоносець , то ми можемо просувати його на 3 класи: на мечника, метальника списа та метальника дротиків. (Рисунок 2.1)

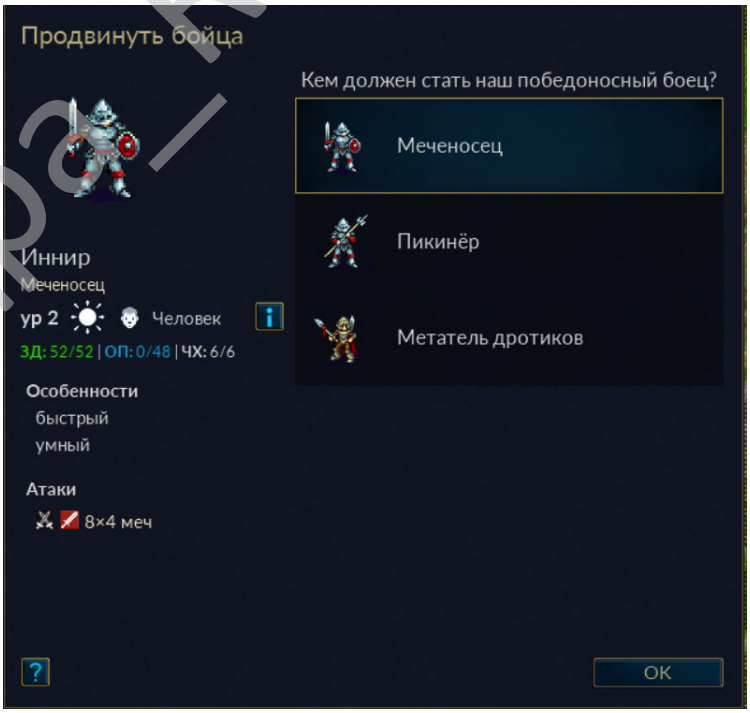

Рисунок 2.1 – Просування списоносця в наступний тур юніта

Ще в далекому 2003-му році, в золотий час покрокових фентезі стратегій, Девід Уайт придумав концепцію гри Battle of Wesnoth. У цей же час, у 2001-му році з'явилися «Деміурги», через рік вийшли ігри із серій Disciples і Age of Wonders, Heroes of Might and Magic 3.

На відміну від комерційних ігор жанру, що блищали в той час на світовому ринку, Battle for Wesnoth на самому початку нічого не могла запропонувати, крім своєї унікальної ігрової механіки. Предтечею формування унікальної механіки в Battle for Wesnoth можна вважати комерційну гру Panzer General 1994 року і, її варіант 1996 року, Fantasy General.

Саме Fantasy General врятувала гру і привернуло до неї увагу спочатку лише невеликої кількості людей. На той час самобутня концепція гри лише мерехтіла відблисками неограненого діаманту з-під товщі сокирної графіки (Рисунок 2.2). Але гра поширювалася безкоштовно, вихідний код був відкритий для вільного користування, тому ц і чинники допомогли зібрати незалежну команду розробників, що взялися розвивати проект. В результаті ми отримали феномен – безкоштовну покрокову стратегію, яка за якістю і інтересом шанувальників перевершує, навіть, багатьох платних ігор.

Рисунок 2.2 – Порівняння інтерфейсу старої версії з новою

Однією з особливостей Battle for Wesnoth є подача ігрового процесу гравцю. Якщо розглядати Heroes of Might and Magic, то ігровий процес там

багатовимірний і ділиться на дві основні складові – глобальну карту, на якій відбувається вся дія, і битви між гравцями, або гравцем і AI, на окремих ігрових дошках. Але в Battle for Wesnoth вони злиті воєдино, а карта з'являється в компаніях лише для кращого розуміння просування сюжету компаній.

Окремої уваги потрібно надати компаніям, в них ігрова дія також поділяється на 2 окремі частини і зосереджена на протистоянні гравця проти AI. Слід відзначати, що в окремих компаніях є можливість вибирати якою дорогою буде подорожувати гравець, що може змінити кількість унікальних по характеру бійців. Такий підхід дає гравцю можливість вплинути на пригоди в компанії.

Компанії поділяють на карту компанії та глобальну карту. Карта компанії демонструє пройдений шлях героїв компанії, місця з інтерактивне бойовим проходженням та місця з інтерактивне сюжетними вставками, з можливими варіаціями дій.

Глобальна карта складається із багатьох шестикутників (гексагонів). Один гексагон містить в собі будь-яку частину ландшафту, замку, природного інтер'єру або населеного пункту. Останнє, до речі, особливо важливо. Економіка в грі не дуже розвинена, в подальшому, економіку потрібно удосконалювати. Єдиним ресурсом гри є золото. Його приносять в якості доходу, податків з захоплених сіл, воно витрачається на покупку юнітів та на їх заробітну плату. Кожного ходу потрібно виплачувати платню своїм бійцям. Але можна не хвилюватися, коли кількість золота буде менше 0, тому що власні юніти не покинуть нас, хоча ми не зможемо купити нового юніта.

Основною концепцією гри є битви, тому майже вся гра присвячена їм. Бойові зіткнення відбуваються безпосередньо на глобальній карті. Самі гравці надані на карті у вигляді персонажів лідерів, які нарівні з усіма можуть брати участь в бою.

Головна мета – знищити лідера противника(Рисунок 2.3.) за будь-яку ціну і при цьому зберегти свого. Те, що відбувається, нагадує якийсь аналог шахів, де замість нудних чорно-білих фігур гравець управляє ельфами, орками і іншими дивовижними істотами зі своїми вміннями та характеристиками .

Рисунок 2.3 – Боротьба між двома ворожими лідерами

Світ Battle for Wesnoth багатий на різноплановість юнітів, серед яких трапляються найрізноманітніші істоти, починаючи від звичайних людей та закінчуючи розумними деревами, що ходять. У грі двадцять рас і деякі з них об'єднані в шість офіційних фракцій, доступних в звичайних сценаріях без доповнень.

Характеристики бійців – тема, яку слід вивчити і спробувати зрозуміти перед початком гри. Основні параметри істоти, крім кількості одиниць здоров'я, це її спосіб атаки, сприйнятливість до видів шкоди і відносини з різними типами місцевості.

Місцевість – це основний критерій пошуку при виборі позиції, куди може і повинен переміститися підлеглий гравця. Можна помітити, що в цей

момент з'являються цифри в шестикутнику курсора, мінливі в залежності від виду місцевості. Ці цифри означають ступінь захисту від ворожих атак, одержані бійцем на даній ділянці карти. Вони залежить від раси і фракції істоти. Ельфи, наприклад, люблять ліс, а гноми люблять проводити час на пагорбах і горах, звідки їх практично нереально вибити. У решти рас свої оригінальні відносини з місцевістю

Рисунок 2.4 – Вибір юніта та демонстрація його характеристик

Більш того, час в грі нараховується цілодобово та складаються з шести фаз, під кожну з яких відведено свій хід для всіх гравців. Б ільшість істот отримують істотний позитивний або негативний бонус залежно від того, чи висить сонце над горизонтом, що значно поглиблює і розширює тактичну складову гри. Зрозуміло, що бійцям "порядку" найкраще атакувати при сяйві сонячного світла, а при заході відступати, адже "хаотичні" бійці починають масовий наступ під покровом ночі.

Але загальні характеристики не єдине, чим можуть похвалитися прості рядові бойові одиниці. У кожного з них є своє особисте ім'я і індивідуальні особливості, які впливають на їх параметри. І найважливіше – бійці можуть

підвищувати свій рівень. Всі, від звичайного солдата до якоїсь тварюки, мають шанс стати крутіше, більше і сильніше, просунувшись по кар'єрних сходах.

Зараз Battle for Wesnoth є повноцінною грою. Її вже давно перевели на величезну кількість мов. Готові пакети доступні для Microsoft Windows, Mac OS X, різних версій Linux та інших операційних систем, навіть, для Android і iOS. Нещодавно Battle for Wesnoth офіційно вийшла в Steam, також планується повноцінний перехід на Godot Engine.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Програмне забезпечення було розроблено за допомогою багатоплатформного інструменту Unity для розробки дво- та тривимірних додатків та ігор. Unity – є гарним вибором для людини, що тільки почала знайомитися з концепцією створення комп'ютерних ігор, тому що Unity – є міжплатформним середовищем розробки комп'ютерних ігор, що дозволяє створювати додатки, які працюють під більш ніж 20 різними операційними системами, такіми як Windows, OS X, Android, Apple iOS, Linux, а також на гральних консолях Wii, PlayStation 3 і XBox 360, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-додатки та інші.

Основними перевагами Unity є наявність візуальної середовища розробки, міжплатформної підтримки і модульної системи компонентів. До недоліків відносять появу складнощів при роботі з багатокомпонентними схемами і труднощі при підключенні зовнішніх бібліотек.

Також варто акцентувати увагу на можливості створювати інтернет-додатки за допомогою спеціального приєднуваного модуля для браузера Unity, а також за допомогою експериментальної реалізації в межах модуля Adobe Flash Player. Застосунки, створені за допомогою Unity, підтримують DirectX та OpenGL, що надають можливість розроблювати для ігор не тільки

локальні сервери, а й використовувати хмари для підтримки бази даних.

Основними технічними характеристиками Unity є наступні:

– Ігрова логіка пишеться за допомогою C#, раніше також була можливість використовувати Boo та JavaScript, але розробники відмовились від їх підтримки;

– Ігровий рушій повністю пов'язаний із середовищем розробки. Це дозволяє випробовувати гру прямо в редакторі;

– Робота з ресурсами можлива через звичайний Drag&Drop.

– Підтримка імпортування великої кількості форматів файлів;

– Вбудований генератор ландшафтів;

– Вбудована підтримка мережі;

– Інтерфейс редактора дуже гнучкий, є можливість писати свої вікна редактора та всілякі розширення для нього;

– Існує рішення для спільної розробки — Asset Server. Також можна використовувати зручний для користувача спосіб контролю версій. Наприклад, SVN або Source Gear.

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Рушій підтримує три сценарних мови: C #, JavaScript (модифікація). Проект в Unity ділиться на сцени (рівні) — окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв і налаштувань. Сцени можуть містити в собі як об'єкти (моделі), так і порожні ігрові об'єкти — тобто ті, які не мають моделі. Об'єкти, в свою чергу , містять набори компонентів, з якими і взаємодіють скрипти. Відповідно у них є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого предмета на сцені обов'язково присутній компонент Transform — він зберігає в собі координати місця розташування, повороту і розмірів по всіх трьох осях. У об'єктів з видимою

геометрією також за умовчанням присутній компонент Mesh Renderer, що робить модель видимою.

На додачу, Unity підтримує фізику твердих тіл і тканини, фізику типу Ragdoll (ганчіркова лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів. Для підключення скрипту, потрібно перемістити скрипт в редактор компонентів, таким чином всі переміні, що були зазначені будуть автоматично продемонстровані в редактору.

При імпорті текстури в рушій можна згенерувати alpha-канал, mip-рівні, normal-map, light-map, карту відображень, проте безпосередньо на модель текстуру прикріпити не можна — буде створено матеріал, з яким буде призначений шейдер, і потім матеріал прикріпиться до моделі. Редактор Unity підтримує написання і редагування шейдерів. Крім того, він містить компонент для створення анімації, яку також можна створити попередньо в 3D-редакторі та імпортувати разом з моделлю, а потім розбити на файли.

Графічний рушій використовує DirectX (Windows), OpenGL (Mac, Windows, Linux), OpenGL ES (Android, iOS), та спеціальне власне API для Wii. Також підтримуються bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), динамічні тіні з використанням shadow maps, render-to-texture та повноекранні ефекти post-processing. З кожною новою версією, можливості Unity поповнюються.

Unity підтримує файли 3ds Max, Maya, Softimage, Blender, modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks та Allegorithmic Substance. В ігровий проект Unity можна імпортувати об'єкти цих програм та робити налаштовування за допомогою графічного інтерфейсу.

Для написання шейдерів використовується ShaderLab, що підтримує шейдерні програми написані на GLSL або Cg. Шейдер може включати декілька варіантів реалізації, що дозволяє Unity визначати найкращий варіант

для конкретної відеокарти. Unity також має вбудовану підтримку фізичного рушія Nvidia PhysX (колишнього Ageia), підтримку симуляції одягу в системі реального часу на довільній та прив'язаній полігональній сітці (починаючи з Unity 3.0), підтримку системи ray casts та шарів зіткнення.

Скриптова система ігрового рушія зроблена на Mono — вільному відкритому проекті з реалізації .NET Framework. Програмісти можуть використовувати UnityScript (власна скриптова мова, подібна до JavaScript та ECMAScript), C# або Boo (мова програмування, подібна до Python).[9] Починаючи з версії 3.0, до Unity входить перероблена версія MonoDevelop для налагодження скриптів.

Після виходу нової версії Unity (версія 5.2.) передбачається вбудована можливість редагувати скрипти у середовищі Visual Studio. Варто відзначити, що раніше Unity також підтримувала Visual Studio, але з версією 5.2. з'явилася можливість встановленняVisual Studio Code, що відкидає непотрібні функції для розробки ігор в Unity, але і залишає і доповнює різноманітними можливостями, середовища.

Сутність розробки ігор — це робота в команді, внаслідок чого, в Unity включений інструментарій для спільної розробки на базі Unity , систему контролю версій Unity Asset Server для ігрових об'єктів та скриптів. Сервер ресурсів Unity це доповнення, яке додає повнофункціональне рішення для контролю версій у функціонал Unity. Як і все інше у Unity, він простий у використанні.

Система використовує PostgreSQL, роботу зі звуком, побудовану на основі бібліотеки FMOD (з можливістю програвати Ogg Vorbis аудіофайли), відеопрогравач із кодеком Theora, рушій для побудови ландшафтів рослинності, вбудовану систему карт освітлення (Beast), мережу для мультиплеєру (RakNet) та вбудовані навігаційні меші для пошуку шляху.

2.3 Розгорнута постановка завдання

Проаналізувавши поставлену задачу про розробку програмного забезпечення системи візуалізації покрокової стратегічної гри, я дійшов висновку, що доцільною буде розробка гри, яка наслідуватиме концепцію гри The Battle for Wesnoth та дух її розробників, які протягом довгого часу розробляли свою гру. Тому характерними особливостями виконання завдання розробки демо-версії покрокової гри повинні стати:

1. Розробити генерацію гексагонної глобальної карта, на якій будуть проводитися битви між гравцями або битві між гравцем та комп'ютером. Карту може генерувати лише розробник, але, в майбутньому, я сподіваюсь розробити редактор карт, за допомогою якого з'явиться можливість створювати карту разом;

2. Зробити AI та виробити декілька режимів поведінки штучного інтелекту, таких як: агресивний, пасивний, що реагуватиме, коли противник перебуватиме в зоні бачення юніта, полузахисний – з реагуванням на ворожий юніт, коли він буде в зоні досягнення атаки із можливістю переміщення та захисний, при якому він не здійснить переміщення, але буде переходити в контрнаступ при спрацьовуванні певного скрипта, або коли противник знаходиться близько юніта;

3. Розробити хоча б один юніт для кожної фракції, в цілому , потрібно розробити цілий пак різноманітних юнітів, таких як: кавалерист, мечник, стрілець, списоносець, вбивця, чародій, священик і тд;

4. Необов'язкове завдання для демо-версії – розробка більш ніж однієї раси, тобто, окрім людей, ще можна розробити: ельфів, у яких буде додатковий захист в гексах лісового покриття, гномів з їх додатковим баченням в темряві та захисті в печерах та горах, орків, тролів, гоблінів, розробити різноманітних монстрів з їх особливостями і тд;

5. Необов'язкове завдання для демо версії – розробити різноманітні

типи гексів, що різняться на 2 види: гекс, що містить лише 1 тип місцевості та змішаний типа гекса, наприклад, ліс сам по собі містить 2 типи, сам по собі – ліс, та тип місцевості, на якій росте ліс;

6. Розробити виділення території, на яку може переміститися юніт, та підкреслити зону виділення зеленим кольором;

7. Юніт матиме 2 фази дії в своєму ході: 1 -ша фаза – переміщення, 2-га фаза – атака, фази будуть поєднанні, тобто, не обов'язково переміщуватися, щоб здійснити атаку, але варто пам'ятати, що при атаці згоратимуть і очки переміщення, і сама дія атаки.

8. Розробити концепцію бафів та, можливо, дебафів, які знаходяться в ящиках і будуть мати свої певні ефекти. Наприклад, зцілення на певну суму здоров'я, або на відсоток від повного здоров'я, додаткова атака, посилення атаки (додатковий шанс на кріт), додатковий рейж, лучнику, або іншому юніту (кинуть зброю в суперника).

9. Для якісного управління програмним забезпеченням, потрібно розробити систему менеджерів, головним завданням яких буде допомога користувачу в редагуванні та додаванні корисних можливостей.

10. Розробити менеджер для управління персонажів ігри, тобто, юнітів, в якому ми зможемо змінювати характеристики юнітів, змінювати їх ім'я, префаб, що візуалізує його зовнішній вигляд. Також менеджер юнітів буде тісно зв'язаний з менеджерами що будуть управляти фракціями та з менеджерами по управлінню гри та генерування карти.

11. Розробити менеджер для управління системи бафів, що буде підтримуватися за допомогою менеджеру по управлінню атрибутів та перків. Таким чином в менеджері бафів, ми зможемо розроблювати бафи, та використовувати за допомогою них систему атрибутів.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

Перш ніж перейти до розробки програмного забезпечення необхідно провести якісне дослідження програмного оточення задачі, тобто потрібно визначити які системні компоненти потрібно використовувати в ході роботи, щоб найефективніше вирішити поставлену задачу, який вони будуть мати інтерфейс і які функції зможуть надавати розробнику.

3.1 Аналіз комп'ютерних систем візуалізації з метою алгоритмізації та обґрунтування щодо їх використання.

Для обрання методу системи комп'ютерної візуалізації, необхідно вирішити наступні задачі

Так як однією із мети роботи обрання системи комп'ютерної візуалізації, що була коротко розглянута в попередньому розділі, необхідно провести дослідження з метою метод обрання системи комп'ютерної візуалізації з врахуванням вимог що до розв'язуваних задач.

Така поставлена мета містить ряд невизначеностей, які необхідно вирішити розв'язавши наступні задачі:

1) Провести класифікацію вимог до систем візуалізації та пов'язати ці вимоги з задачею, яку поставлено до системи візуалізації.

2) Провести класифікацію систем комп'ютерної візуалізації з врахуванням визначених вимог.

3) Побудувати метод визначення комп'ютерних систем візуалізації, які задовольнятимуть визначеним вимогам.

Відповідно до визначення вимог до задач, які повинна вирішувати система комп'ютерної візуалізації, потрібно провести насамперед класифікацію задач, які вирішує система візуалізації. В наш час комп'ютерні

системи використовуються не лише в технічних областях, але вирішують задачі і в побуті, мають значну вагу в мультимедіа та використовуються для розваг та ігор. Відповідно, для кожної із задач, вимоги до системи візуалізації є суттєво різними. Наприклад, для мікроконтролерного пристрою, який надає кілька виміряних значень, може мати світлодіодний індикатор на кілька цифр, а система проектування повинна розгортати креслення з високою роздільною здатністю. Як показало дослідження, в більшості джерел [3, с.6; 4, с.20; 5, с.5; 6, с.3] використовують поділ комп'ютерної графіки не за класифікацією до вимог, що стоять до процесу утворення зображення, а по сферам людської діяльності, наприклад в [4] наведено класифікацію, яку відображає наступний список:

1) інженерна графіка;
2) автоматизовані системи наукових досліджень;
3) інформаційні системи;
4) системи ілюстративної та ділової графіки;
5) системи машинної геометрії;
6) анімаційні задачі;
7) комп'ютерні ігри;
8) відео тренажери;
9) мистецтво та видавнича діяльність.

Однак, в розрізі ділової графіки, необхідно враховувати засоби демонстрації графічних матеріалів. Наприклад, графік для зображення динаміки змін даних в часі для презентації матиме інші вимоги до системи візуалізації ніж фотографічне зображення планування жилого комплексу як і по роздільній здатності, так і в розрізі правильності передачі кольору. Більш детальна класифікація [3, с.6-7] містить опис специфічних вимог до систем візуалізації, але ці вимоги в одному пункті можуть мати протиріччя.

Наприклад, за наведеною класифікацією інженерна графіка може вимагати моделювання в реальному часі та реалістичність візуалізації, що може

привести до протиріччя вимог. Тому стає актуальним розв'язання задачі побудови класифікації вимог до систем візуалізації та розробки методики проекції поставлених задач діяльності на ці вимоги до систем візуалізації.

Пропонується використовувати для пошуку системи візуалізації, систему яка базується на вимогах до результату візуалізації. Тому за основу можна запропонувати наступні вимоги до комп'ютерних систем візуалізації:

1) швидкодія;

2) правильність побудови освітлення (накладання тіней повинно бути правильним, але їх зображення може бути схематичним);

3) реалістичність побудови освітлення (допускає порушення законів оптики, пропущення тіней, але будує якісні для ока зображення);

4) підтримка динамічного освітлення;

5) точність відображення кольорів;

6) врахування властивостей поверхні при розрахунку кольору пікселя (відблиски, розсіювання, поглинання, світіння, колір поверхні);

7) використання текстур;

8) використання мап рельєфу;

9) зображення синтезоване;

10) зображення натуральне;

11) допуск втрат інформації при стисненні або відтворенні зображення;

12) вимоги до складності сцени;

13) можливість змінювати метод зображення (показати лише вершини, ребра, ігнорування текстур або матеріалу або інші опції, які можна змінити в довільний момент);

14) можливість будувати стереоскопічні пари зображення сцени;

15) інші вимоги.

З точки зору виконавця технічного проекту, в якому використовуються системи візуалізації, формулювання вимог не є першим етапом, а в першу

чергу формується задача, яку потрібно розв'язати за допомогою системи візуалізації. Приклади таких задач показано в наступному переліку:

1) плакати;
2) ділова графіка;
3) публікація в Інтернеті;
4) 3D графіка, художня;
5) графіка для динамічних ігор;
6) графіка для повільних ігор;
7) графіка інженерна;
8) графіка конструкторська;
9) технічні інтерактивні схеми для устаткування;
10) відображення графіків та діаграм;
11) відеомонтаж;
12) відеоспостереження;
13) медичне зображення;
14) наукове зображення;
15) відображення текстів для редагування;
16) системи віртуальної реальності;
17) інші.

Створені списки є суб'єктивно-орієнтовані і можуть відрізнятися в залежності від кола інтересів робочих груп. Тому метод може трансформуватися до конкретних алгоритмів, які відповідатимуть потребам конкретних груп розробників або замовників програмного забезпечення.

На основі наведених переліків вимог та задач пропонується метод обрання систем візуалізації на основі наступної послідовності дій (термінологія наукових методів, які використані в блок-схемі, взято з [7]; Рисунок. 3.1.):

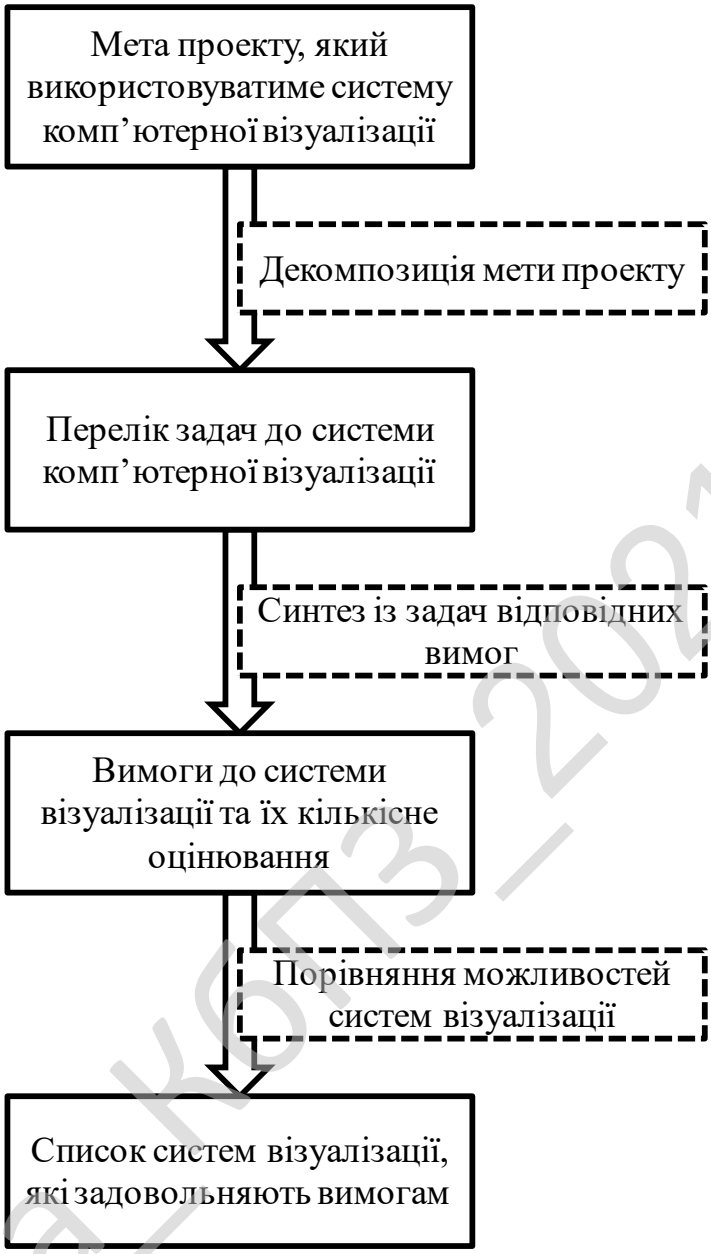

Рисунок 3.1 – Метод визначення сумісних комп'ютерних систем візуалізації

Побудований метод не є детермінованим, навіть при умові фіксованих списків задач та вимог, тому що метод містить дії, які мають інтелектуальні складові при виконанні дій декомпозицій та синтезу. Процес порівняння та конкурс комп'ютерних систем візуалізації є можливість формалізувати при умові фіксації списку вимог до систем візуалізації та засобів їх кількісного оцінювання, наприклад, використавши на основі теорії множин та кортежів кількісних оцінок правила багатокритеріальної оптимізації Парето [8].

Авторами розглянуто приклад застосування методу обрання системи комп'ютерної візуалізації на прикладі проектування програмного забезпечення, яке реалізує покрокову стратегію. Однак, формулювання вимог до системи комп'ютерної візуалізації можливий лише після аналізу вимог до отриманого зображення з порівнянням вимог до інших програмних засобів де використовується графіка. Тому першим етапом обрання системи візуалізації є формулювання мети проекту програмного забезпечення – побудова зображення для комфортної гри в жанрі покрокової стратегії.

3.1.1 Формування переліку задач на основі декомпозиції мети проекту.

Швидкість побудови зображення є однією із головних особливостей динамічної (ігрової) візуалізації, оскільки від швидкості побудови зображення залежить сприйняття користувачем розробленої гри. Це зобов'язання виникло внаслідок необхідності враховувати швидкість сприйняття інформації людиною. У середньому, людський зір здатний реагувати лише на 20-25 кадрів у секунду. Вимоги до швидкості побудови зображення різняться від динаміки жанру гри, наприклад, у шутерах та гонках спостерігається часта зміна зображення і для забезпечення вчасного реагування на зміни в оточенні часто вимагається можливість побудови 120 та більше кадрів за секунду. У стратегічних іграх, графічних редакторів та інженерних рішеннях, зокрема і в покроковій стратегії, швидкість зміни кадрів повинна бути лише достатньою для забезпечення видимості плавності анімацій та відсутності мерехтіння – 30 кадрів за секунду. Більш детально описано про дотримання частоти зміни кадрів наведено в [9]. Покрові стратегічні ігри насамперед базуються на принципах неспішності, тому динамічна зміна зображення в таких іграх створюється для подоби живої картини. Застосування такої динаміки вдало спостерігається на прикладі гри з серії Civilization. Починаючи з перших ігор у

серії, пересування ігровиз персонажів, побудова внутрішніх та зовнішніх об'єктів у містах, використання зброї масового ураження та природні лиха супроводжувалися динамічною зміною на карті. Такий виборчий підхід до динаміки зміни зображення дозволив підтримувати плавну та захоплюючу атмосферу, а також задовольнити потребу користувача відстежити прогрес в грі.

Велику швидкість зміни зображення можна простежити на прикладі гри Quake та її модифікації DeFRaG. Гра Quake є класичним шутером від першої особи, в якому швидкість зміни кадрів середньо-швидка. В її модифікації DeFRaG, при активному проходженні мап, на певний час можлива швидка зміна кадрів. Швидкість руху в гоночних іграх може досягати більш 250 кілометрів за годину, тому, з огляду н а те, що під час проходження траси потрібно часто змінювати положення камери, зміна кадрів також повинна відбуватися з достатньою частотою.

Користувач повинен чітко розуміти особливості поверхні та рельєфу в ігрових локаціях. Наприклад, якщо розглядати культову гру Heroes Might of Magic 3, то в ній спостерігається розподілення на глобальну карту та на карти битв. На глобальній карті рельєф, перш за все, впливає на швидкість переміщення героя на тій чи іншій місцевості. В битвах ігрові одиниці, що приписані місцевості на якій йде битва, отримують прибавку у 1 одиницю наступних показників: атака, захист, швидкість (як переміщення, так і ініціатива). Крім того, в битвах має певне значення рельєф карти в вигляді перешкод, що інколи кардинально змінює результат битви. Місцевість впливає не лише на бійців, але і на самого героя, забороняючи використовувати магію вище 1-го рівня, або дозволяючи чаклувати усі заклинання на рівні «експерт», що має найвищий рівень [10].

При створенні рельєфу варто звертати увагу не тільки на технічну складову, а і на візуально динамічну. Динамічні сцени покрокової стратегії використовують найчастіше для створення живого ефекту та поділяються на

«пасивні» та «активні», що реалізується за допомогою об’єктів на карті (наприклад, курка, яка дзьобає щось на землі, працюючий млин, дим з будинків і т.д.), пересування хмар та води, дій ігрового персонажу на місці (наприклад, виконує анімацію перевірки свого клинку на гостроту). Спочатку проминають такі моменти, адже найчастіше покрокові комп’ютерні стратегії розробляються як аналог настільній гри. Але якщо приділити анімаціям увагу, то гра отримає більш живу атмосферу, а гравець можливість більш зануритися в атмосферу гри.

Однією із складових якісної гри є оптимізація. Спрощення візуалізації об’єкта є найлегшим способом досягти підвищення швидкості формування зображення. Під спрощенням візуалізації передбачаються дії, які приводять до зменшення потреб в обробці зображення. До описаних дій належить прив’язка статичної тіні об’єкта до власне моделі об’єкта. Проблема згаданої прив’язки може бути лише в динаміці тіней, тому такий підхід до оптимізації гри є рентабельним. Якщо розглядати точність побудови тіні об’єкта, то для початку потрібно знати як саме тінь була створена, чи є вона частиною моделі об’єкта, або динамічно формується при зміні кута та сили освітлення. Якщо аналізувати ігри від першого особи, то можна спостерігати динамічну зміну тіней або її відсутність при взаємодії об’єкта з навколишнім середовищем, тобто зміни освітлення за допомогою умовно «факела», що активується і підводиться до об’єкта дослідження. Якщо досліджувати більш складну техніку освітлення "Muzzle Flash Shading", у якій створюється ефект реалістичних тіней від об'єктів при стрільбі з вогнепальної зброї, то варто відзначити, що такий ефект спостерігається не в кожному шутері, а його реалізації в іграх від третьої особи і зовсім немає і по цей час. Згаданий ефект мав би бути у Gears of War 1, так як це було демонстровано в трейлері гри, але був вирізаний з релізної версії, тому "Muzzle Flash Shading" можна зустріти лише в іграх Doom 3, STALKER [11].

Зазвичай саме в іграх з віддаленою камерою та при наявності умовно 3D об’єктів застосовуються статичні тіні, так в грі Heroes Might of Magic 3, можна

спостерігається прив’язану тінь об’єкта, але навіть така тінь є необхідною для створення об’ємності та правильності об’єкта.

Ще однією особливістю динамічної візуалізації є технологія трасування променів. Згадана технологія дозволяє придати грі більшій реалістичності, завдяки обчислюванню траєкторії кожного променя світла від об’єкта до камери, і виводить на екран картинку з урахуванням проведених розрахунків. Перш ніж використовувати технологію трасування променів потрібно ретельно ознайомитися з матеріалами та кольорами, які будуть на сцені, так як одні з них краще відображають світло, інші гірше, а скло і зовсім переломлює промені, що створює інший ефект. Докладний підхід дає можливість точно передати освітленість предметів, їхні тіні, заломлення, розсіювання світла й, взагалі, надавати зображенню на екрані реалістичніший вигляд. Окрім цього, трасування променів дозволяє спростити створення рівнів та їхнього оточення. Так при створення рівня, на якому буде горіти світильник, здебільшого, потрібно налагодити тіні та промені вручну. А за допомогою трасування променів, алгоритм, що відповідає за цю роботу, обробить складові рівня в автоматичному режимі. Окрім цього, трасування може працювати й зі звуком, створюючи набагато реалістичніше аудіо для ігор. Проблемою технології трасування є її витрати на ресурси комп’ютера, тому що обробка одного кадру за допомогою технології трасування може займати від декількох годин до декількох днів. Отже, цілковито застосовувати досліджувану технологію для створення ігор на теперішній час не є раціональним рішенням, потрібно її використовувати за потреби та лише для деяких об’єктів [12, 13]. Відповідно, технологія трасування променів хоча і є бажаною частиною гри, але не є складовою, особливо якщо розглядати такий жанр як стратегія, в якій технологія майже не застосовується.

В багатьох випадках вимагається ілюзія реалізму або навмисна стилізація до рисунків. Ілюзія реалізму або стилізація потрібна при

створюванні ігор наближеної реальності: історичних, фантастичних, мультиплікаційних та інших історій [14, 15].

Існує велика кількість елементів, які неможливо спростити. Вдалим прикладом системи, в якій неможливо спростити елементи, може бути візуалізація результатів медичного обстеження, або креслення та схеми приладів або схем.

В більшості інженерних та художніх програмах є можливість переключатися між методами візуалізації. Відображення лише ребер об'єкту дозволяє чітко побачити утворення форми, наявність внутрішніх елементів та інше. Спрощений рендер без урахування особливостей поверхні, тобто матеріалу та кольору, тобто текстури, дозволить дуже швидко будувати складні зображення з більшим вмістом вершин та граней без спрощення форми. Спрощений рендер, хоча і враховує матеріал та текстуру, призначений лише для оцінювання отриманої картини або для художнього сприйняття. Остаточне зображення будується найбільш якісними рендерами, за допомогою яких можна досягти фотореалістичності, але при цьому побудова кадру, в залежності від складності сцени, може продовжуватися від кількох секунд до декількох годин.

Технічне зображення повинне бути зрозумілими, а не гарним. Для технічної візуалізації необхідна насамперед якість виконання роботи, зображення повинно містити лише необхідну інформацію. Непотрібно гнатися за гарним зображенням, головне щоб все було зрозуміло і не відволікало від роботи із програмою. Тому такі програми як AutoCAD, для створення технічних креслень, не має можливості створювати фото реалістичні сцени, проте має можливість швидко будувати складні об'єкти та їх перерізи із точним врахуванням освітленості. Це досягнуто за рахунок зниження реалістичності та зниження частоти кадрів [16]. Також тут підвищені вимоги правильного переходу від полігонального зображення об'єктів до його креслення в заданій проекції. Вдалим прикладом може слугувати будинок із

його переходом від цілого будинку до окремих поверхів, квартир, або можна уявити двигун та його складові.

В результаті можна відокремити наступні задачі візуалізації комп'ютерної покрокової стратегічної гри: забезпечити частоту кадрів; забезпечити зображення складних сцен; забезпечити зображення тіней від об'єктів сцени (правильність, реалістичність, динамічні/статичні), забезпечити врахування властивостей матеріалу; забезпечити використання текстур; забезпечити точність відтворення форм; врахувати відбиття та заломлення світла.

В результаті огляду технічних особливостей комп'ютерної візуалізації є можливість переходу до їх систематизації та аналізування вимог по кожній з відокремлених задач, що відповідає наступному пункту за схемою на рисунку 3.1.

3.1.2 Формування переліку вимог.

На основі зазначених міркувань, можна зробити витяг основних задач, які полягатимуть на систему візуалізації. За мету роботі прийнято обрання системи візуалізації покрокової стратегії, але для порівняння та підкреслення особливостей додаємо задачі візуалізації шутеру та системи попередньої демонстрації моделей для 3Д-друку (таблиця 3.1):

Таблиця 3.1 – Задачі, які поставлені до системи візуалізації

	Покрокова стратегія	Шутер	Демонстрація моделей 3Д-друку
Швидке генерування кадру	Близько 30	Більше 30, бажано 120 та більше	До 1..3 с.

Продовження таблиці 3.1

Динамічні тіні	Рельєф незмінний, можливо використати статичні тіні	Бажано використати динамічні тіні	Тіні розраховуються для кожного кадру, динамічно
Правильність тіней	Можливі схематичні тіні	Можливі схематичні тіні	Високі вимоги до правильності побудови тіней
Складність сцени	Висока складність сцени	Середня складність сцени	Дуже складна складність сцени
Реалістичність тіней	Бажаний реалізм	Бажаний реалізм	Реалізм не вимагається
Використання властивостей матеріалу	Високі вимоги до врахування матеріалу	Високі вимоги до врахування матеріалу	Достатньо загального кольору
Якість текстур	Висока	Висока	Текстури - відсутні
Можливість спрощення моделей	Так	Так	Ні
Врахування відбиття та заломлення світла	Бажано, приблизно	Бажано, приблизно	Є дуже важливою частиною

3.1.3 Кількісне оцінювання вимог до системи візуалізації

Наступний крок також містить суб'єктивну складову, яку виконують експерти, які проводять проектування програмного продукту. Для цього

відокремлені задачі з приблизними вимогами зводять в таблицю (таблиця 3.2). Тут проводиться кількісна оцінка вимоги та експертна оцінка важливості дотримання окремої вимоги:

Таблиця 3.2 – Кількісна оцінка вимог та їх важливості

			Тіні						
Показник	30	1000 об’єктів	Ні	Так	Так	Так	Так	Так	0 грн.
Важливість, 0..1	1	1	0,3	0,8	0,5	0,75	0,9	0,2	0,95

Після формування таблиці вимог, їх важливості, потрібно визначити характеристики доступних систем візуалізації, та обрати з них більш оптимальні за принципом Парето.

3.1.4 Вибір систем візуалізації, які задовольняють вимогам

Обрання системи візуалізації проведено за характеристиками, які наведено в [17-20] (таблиця 3.3):

Таблиця 3.3 – Кількісна оцінка вимог та їх важливості

			Тіні							

Продовження таблиці 3.3

Важливість, 0..1	1	1	0,3	0,8	0,5	0,75	0,9	0,2	0,95	
Цільові показники	30	1000 об’єктів	-	+	+	+	+	+	0	6,40
Unreal Engine [15]	+	+	-/+	+	+	+	+	+	0	6,25
Unity3D [16]	+	+	-	+	+	+	+	-	0	6,20
Blender 3D [17]	+	-/+	+	+	+	-/+	-	+	0	4,90
GoDot [18]	+	+	-	-	-	-	-	-	0	3,25

В результаті співставлення вимог, з врахуванням їх важливості, виділено перевагу систем створення ігрового програмного забезпечення Unreal Engine та Unity3D. Обидві системи мають високу відповідність вимогам. Тому для уточнення вибору потрібно враховувати вже не інженерні показники. Наприклад, важливою частиною аргументів що до вибору, є наявність підготовлених кадрів та програмістів, для роботи з тією або іншою системою.

3.2 Опис функціонування системи

Дослідивши поставлене завдання, мною було прийнято рішення розпочинати розробку програмного забезпечення з розгляду глобальної бойової карти. Для генерації карти, були розроблені моделі гекса та квадрата. За допомогою цих моделей створюємо та редагуємо глобальну карту.

На мою думку, для покр окових стратегічних ігор самою правильною формою є модель гекса, із-за його форми, що дозволяє діяти на всі 6 напрямів та уникнути проблеми з діагоналями в квадратах. Форма гекса була створена

за допомогою блендера і представляє собою шестикутну фігуру. Після чого фігура була перенесена в юніті і, за допомогою додаткових мешів, була створена можливість зміни кольору гекса (Рисунок 3.3.).

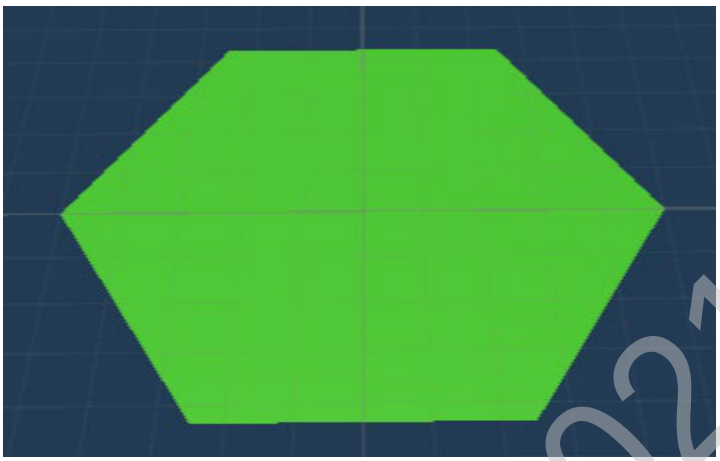

Рисунок 3.3 – Префаб гекса, для генерації глобальної карти

Наступним кроком, використовуючи наявний гекс, була розробка генерації самої карти. Для генерування карти бойових дій, на мій погляд, потрібно розробити 5 характеристик, за допомогою яких і буде створюватися та змінюватися карта. Такими характеристиками є: довжина, ширина, внутрішній радіус, зовнішній радіус і «п орожнини». Розглянемо кожну характеристику окремо:

Довжина та ширина – представляють кількість гексів в ряді та кількість стовбців. Довжина – це рядки, ширина – це стовбці.

Внутрішній радіус – представлення версії гексу, що змінюється за допомогою коефіцієнта, який надається розробником. Як мені здається, розмір, що розтягує гекс у 2.5 рази є ефективним для зорового представлення користувача

Зовнішній радіус представляє відстань гексів між собою. Якщо відстань дорівнює 1-ці, то карта – виглядає як суцільне полотно, таким чином, при зміні меша на «траву», ми зможемо спостерігати галявину.

Порожнини – це коефіцієнт, за допомогою якого автоматично вибираються гекси, що не будуть генеруватися при створенні карти. Якщо

використовувати цей коефіцієнт, то варто пам’ятати, що він виставляється від 0 до 1 . При виставленні коефіцієнту порожнини, рівному 1, карта згенерує максимальну кількість порожнин, що знівелює можливість проведення ближнього бою.

Отже, якщо генерувати карту за допомогою цих властивостей, ми зможемо спостерігати наступне (Рисунок 3.4.)

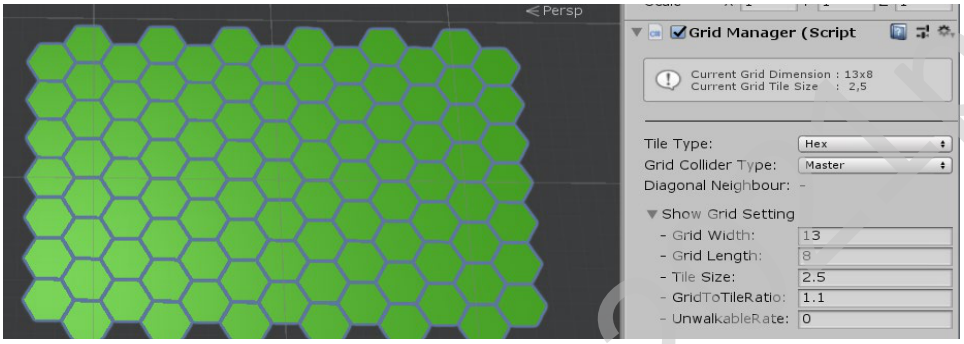

Рисунок 3.4 – Глобальна карта з детальними характеристиками

Етап генерація гексагональної карти був для мене складним моментом в потрібності згенерувати карту так, щоб гекси не накладалися один на одного, а займали строго те місце, що потрібно, займаючи відповідну дистанцію один від одного.

Генерація гексагональної карти розпочинається з аналізу характеристик, завданих розробником на початку генерації.

```
public static Grid GenerateHexGrid(int width=5, int length=5, float
gridSize=1, float GridToTileRatio=1, float unwalkableRate=0,
GridManager._GridColliderType colType=GridManager._GridColliderType.Master){
```

Потім вибирається префаб, за допомогою якого і буде генеруватися карта, в моєму випадку я створив 2 префаба, різниця між якими є в їх зовнішньому вигляді

```
string loadText="";
if (colType == GridManager._GridColliderType.Individual)
{loadText = "ScenePrefab/HexTile_Collider";}
else if (colType == GridManager._GridColliderType.Master)
{loadText = "ScenePrefab/HexTile";}
Transform tilePrefab = Resources.Load(loadText,
typeof(Transform)) as Transform;
if (loadText == "ScenePrefab / HexTile")
{Debug.Log("ScenePrefab == " + loadText);}
```

Після вибору відповідного префаба та отримання характеристик гексів, розпочинається створення карти, що ми можемо спостерігати в наступній частині коду :

```
                List<Transform> tileTransformList=new List<Transform>();
                      Vector3 pos=Vector3.zero;
                      Transform parentT=new GameObject("GridTemp").transform;
                      float spaceX=spaceXHex*gridSize*GridToTileRatio;//Ki
                      float spaceZ=spaceZHex*gridSize*GridToTileRatio;
                 Grid grid=new Grid(_TileType.Hex, width, length, gridSize,
          GridToTileRatio);
                      for(int i=0; i<width; i++){
                            float offsetZ=(i%2==1) ? 0 : (spaceZ/2);
                            int limit=i%2==1 ? length : length-1;
                      for(int n=0; n<limit; n++){
                                  pos=new Vector3(i*spaceX, 0,
                                  n*spaceZ+offsetZ);
                             Transform
                      tileT=(Transform)Instantiate(tilePrefab, pos, rot);
                            tileTransformList.Add(tileT);
                            tileT.localScale*=gridSize;
                            tileT.parent=parentT;
                            tileT.gameObject.layer=TBTK.GetLayerTile();
                            Tile tile=tileT.gameObject.AddComponent<Tile>();
                            tile.type=_TileType.Hex;
                            tile.x=i;
                            tile.y=-((i+1)/2)+n;
                            tile.z=-(i/2)-n;
                            grid.tileList.Add(tile);}}
                grid.AssignIndex();
```

Після створення гексів та поєднання їх в одну плоскість, виконується перевірка на порожнини та їх генерування, спочатку реалізується перевірка на наявність змін, після, якщо зміни є, тобто більше ніж 0 , здійснюється створення порожнин:

```
                      if(unwalkableRate>0){
                            int unwalkableCount=(int)
       (grid.tileList.Count*unwalkableRate);
                            for(int i=0; i<unwalkableCount; i++){
                                  int rand=Random.Range(0,
grid.tileList.Count);
                                  grid.tileList[rand].walkable=false;
                      grid.tileList[rand].gameObject.SetActive(false);}
                      float x=(width-1)*spaceX;
                      float z=(length-1)*spaceZ;
                      parentT.position-=new Vector3(x/2, 0, z/2);
                      Transform newParentT=new
GameObject("HexGrid").transform;
       for(int i=0; i<tileTransformList.Count; i++)
tileTransformList[i].parent=newParentT;
                      DestroyImmediate(parentT.gameObject);
                      grid.gridObj=newParentT.gameObject;
                      newParentT.gameObject.layer=TBTK.GetLayerTile();
                      if(colType==GridManager._GridColliderType.Master){
```

```
BoxCollider boxCol=newParentT.gameObject.AddComponent<BoxCollider>();
                        boxCol.size=new Vector3(5000, 0, 5000);}
                    return grid;}
```

Після завершення генерування порожнин, якщо було виставлено таке значення, завершується компоновка глобальної карти. Але слід пам'ятати, що порожнини генеруються випадковим чином, тому при виставленні коефіцієнту вище 1, може статися, що згенерується лише 1-н гекс, а це може призвести до помилки системи. Помилка виникне внаслідок того, що при запуску програмного забезпечення генеруються також бійці двух або більше команд, з'являються бафи, після завершення раунду та на початку старта гри. При бажанні, можна уникнути цієї помилки, використовуючи коефіцієнт метода «порожнин» від 0 до 0.4.

Для кращого адаптування програмного забезпечення та подальшого розвитку, майже на кожний вагомий компонент, був створений UI, що є частиною сценарію. Таким чином, полегшується подальша розробка та покращується можливість тестування програмного забезпечення.

Наступним кроком було створення юнітів та системи фракцій, що полегшить інтерфейс як розробнику, так і користувачу в подальшому тестуванні покрокової стратегічної гри.

Створення юнітів розпочав із побудови префаба для юніта та його меша. Меш був обраний, так же як і гекс, однокольоровим. При створені фракцій, я вирішив, що розділити юніта по фракціям потрібно допомогою зміни кольорову, тому виокремив для першої фракції золотистий колір, а для другої фракції було обрано синій колір.

Префаб юніта був створений посредством блендера. За допомогою редагування та додавання зброї та щита був створений списоносець, який був обраний тимчасовим, тестовим юнітом для обох фракцій, з різницею лише у кольорі. (Рисунок 3.5.)

Рисунок 3.5 – Модель списоносця для обох фракцій.

По моїй задумці, у кожній фракції в подальшому, ми зможемо спостерігати велику різноманітність юнітів, таких як: лучник, кавалерист, мечник і тд. Але для тестового програмного забезпечення, що візуалізує покрокову стратегічну гру буде достатньо і лише одного списоносця, що може атакувати, як в ближньому бою так і на відстані 2 кліток.

Для візуалізації покрокової стратегічної гри списоносець має потенціал атакувати як у ближньому бою, так і на відстані 2 кліток, та наділений наступними характеристиками: атака, здоров'я, кількість доступних атак, кількість доступних спроб переміщення, відстань, яку він може пройти за одну спробу на переміщення. Розглянемо кожну характеристику окремо:

Атака юніта – це показник шкоди, що відніме показник життя противника з певною вірогідністю та за певним показником. Наприклад, у списоносця вірогідність нанесення шкоди становить 65%, при умові, що при попаданні списоносець може нанести противнику від 3 до 6 одиниць шкоди. Також є вірогідність в 10%, що при умові вдалого нанесення шкоди, наприклад, в 6 одиниць і спрацьовуванні 10% вірогідності, шкода буде становити 12 одиниць здоров'я супротивника.

Здоров’я – це кількість умовних одиниць, що демонструє стан здоров’я юніта, при досяганні 0 одиниць здоров’я юніт запускається скрипт на видалення юніта з поля боя, тобто юніт вмирає. Ю ніт може поповнити своє здоров’я при умові активації регенерації, яку можна отримати шляхом переміщення юніта на відповідний баф.

Кількість доступних атак визначає, що при запуску гри у списоносця є лише 1 спроба атаки в кожному ході та, при умові активації додаткової атаки, через накладення відповідного бафа на юніта, юніт може отримати, протягом декількох ходів, додаткову атаку. Час дії бафа вказується в міні-характеристиках юніта.

Кількість доступних переміщень визначає. що при запуску гри, у списоносця є лише 1 спроба переміщення в кожному ході на певну відстань. Відстань списоносця становить 3 гекса.

Після створення юнітів та закріплення за ними відповідних характеристик, ми можемо перейти до розробки власне боротьби між фракціями. Кількість фракцій – необмежна, але для якісного тестування було прийнято рішення, використовувати лише 2 фракції. Фракції відрізняються між собою тим, що перша підвладна гравцю, користувачу, а інша була під управлінням AI. Візуальна різниця фракції в кольорі, їх зовнішній вигл яд ми спостерігали на рисунку 3.1.3.

Для якісного менеджменту ресурсів були розроблені менеджери для створення і управління новими юнітами, фракціями, бафами, можливими перками, глобальною картою та ін. Для кращого розуміння розберемо декілька менеджерів.

Розпочнемо аналіз з фракційного менеджера (рисунок 3.6). Фракційний менеджер дозволяє створювати за допомогою шаблона фракції, змінюючи їх колір, кількість одиниць, які буде заспавнено на місце проведення боя, їх різновид, тип управління фракції (гравець або AI), фракційні здатності (перки).

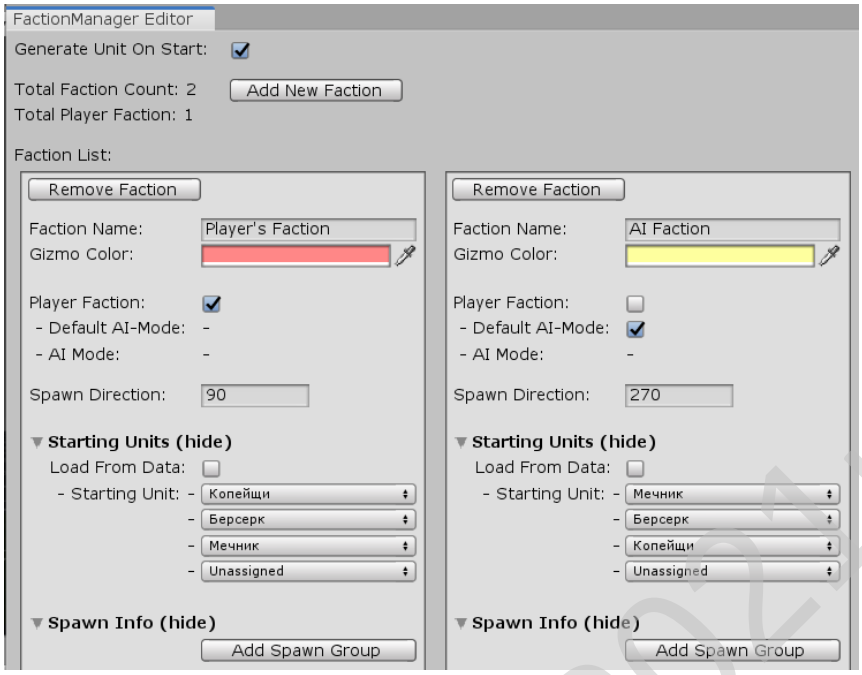

Рисунок 3.6 – Інтерфейс фракційного менеджеру

За допомогою кнопки Add New Fraction, ми можемо створити нову фракцію, використовуючи відповідний шаблон, що дозволить нам різноманіти кількість фракцій на поле битви та дозволить розвиватися разом із друзями та колегами. Шаблон версії ми можемо спостерігати на рисунку 3.7

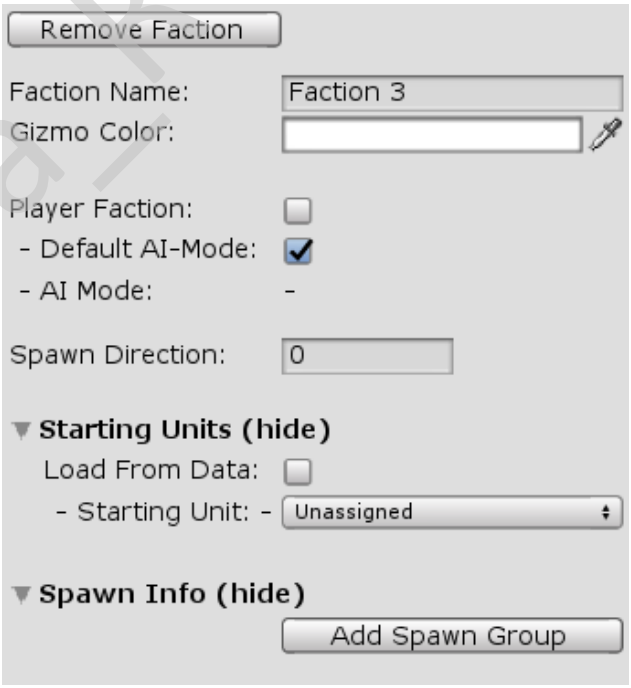

Рисунок 3.7 – Демонстрація шаблону нової фракції

Якщо нам незрозумілий якийсь пункт, або користувач/розробник хоче згадати за що відповідає певний пункт, він може навести курсор мишки на відповідний пункт, тоді з’явиться вікно-повідомлення з вказівкою про функцію певного пункту (рисунок 3.8.)

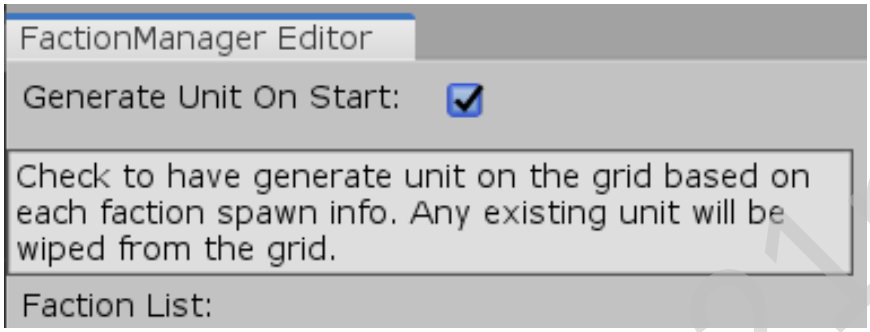

Рисунок 3.8 – Демонстрація вікна-повідомлення на наведений пункт

Для видалення фракції ми повинні натиснути на кнопку Remove Fraction і підтвердити видалення фракції натиснувши на Continue. Варто пам’ятати, що видалену фракцію, ми не зможемо повернути, нам залишиться лише створити нову фракцію.

Щоб змінити назву фракції нам достатньо обрати теперішнє ім’я, змінити його на те, що ми хочемо. Зміна назви фракції автоматично збережеться.

В даній версії користувач може змінювати назву фракції, її колір, в майбутньому і емблему. Управління фракції може брати на себе як гравець, так і штучний інтелект.

Spawn Direction відповідає за поворот юніта на карті, тобто, якщо spawn direction – дорівнює 90, то це означає, що юніти цієї фракції будуть повернені на 90 градусів наліво при спавні на карту.

За допомогою поля Starting Unit ми можемо обирати юнітів, що були додані до менеджеру юнітів та корегувати їх кількість та тип при стартовому спавні. При обирані юнітів, ми можемо спостерігати і юнітів іншої фракції, так як їх різниця лише в моделі і всі вони проходять через менеджер юнітів.

Faction Abilities – тестова версія фракційний перків, які будуть діяти як «божественна магія» в світі фентезі. Наприклад, за 20 одиниць енергії можна відновити 5 одиниць здоров'я своєму бійцю, або за 50 одиниць енергії нанести 10 одиниць шкоди конкретному юніту та по 5 одиниць шкоди сусіднім.

Один із головних менеджерів, що дозволяють якісно редагувати різновид можливих юнітів та надають їм клас юніта є Менеджер Юнітів.

Менеджер юнітів (рисунок 3.9.) розділяється на 2 зони. 1-ша зона, та що зліва, демонструє вікно всіх наявних класів і різновидів юнітів, які є на теперішній час, та вікно, за допомогою якого можна виокремити нового юніта, підв'язавши до префаба відповідний скрипт Unit.cs. До додаткових функцій лівого вікна можна віднести складування/розгортання лівої частини та можливість переміщення між собою класів юнітів.

Права частина менеджера юнітів розроблена для демонстрації, зміни та доповненні характеристик окремо вибраного класу юніта. В ній можна як змінювати іконку, що відображає міні модель юніта або його портрет, так і власне модель. Назву юніта можна змінювати в самому менеджері.

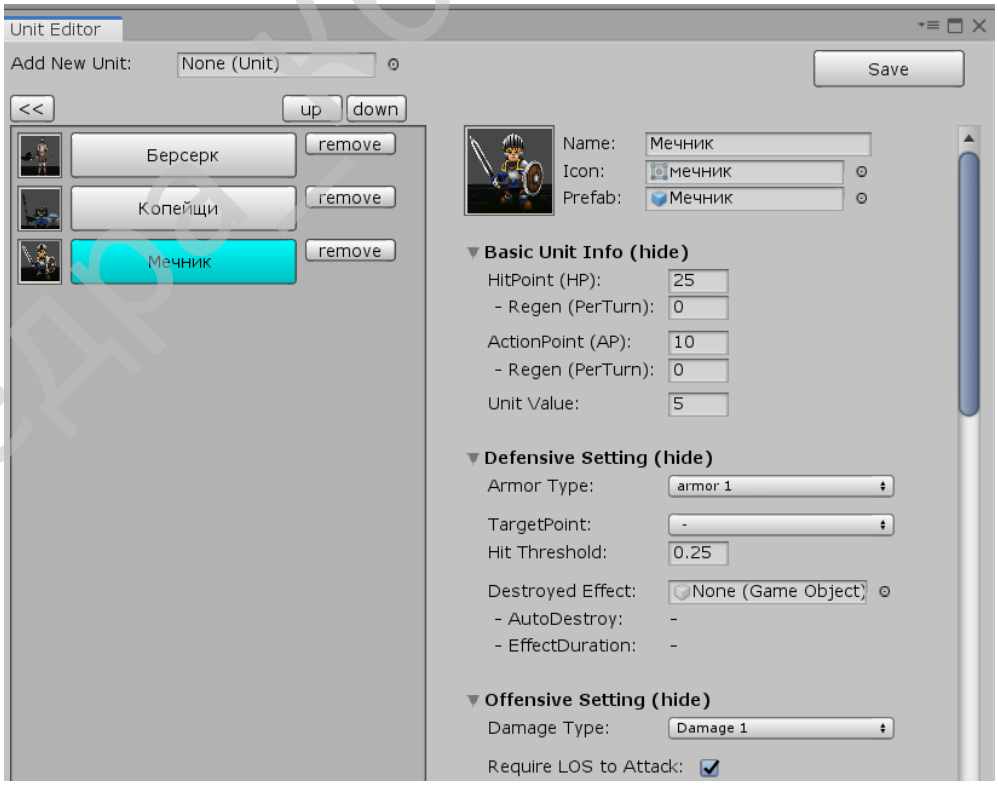

Рисунок 3.9 – Інтерфейс менеджеру юнітів

Щоб зберегти зміни, які були внесені, була розроблена кнопка Save, завдяки якій ми безпосередньо можемо контролювати зміни, що будуть вноситися і зможемо зменшити рівень механічних помилок.

У всіх юнітів є декілька основних характеристик, які ми можемо змінювати в менеджері, такі як: HitPoint (здоров'я), Per Tur n Regen, що регенерує, тобто добавляє до очків здоров'я, вказану кількість кожний новий хід. ActionPoint – це активні очки дії, що використовуються на атаку суперників, на переміщення та будуть витрачатися на активні навики юніта (в розробці).

За допомогою менеджера в майбутньому я хочу змінювати як тип атаки, так і тип захисту, що дозволить покращити та різноманітити гру та гарно вплине на формування різних підрозділів, заточених під окремо взятого юніта. Так, в майбутньому маг, який володіє магією світла, зможе краще боротися з тими юнітами, що мають слабкий захист від магії світла.

В подальшому буде створена анімація руйнування юніта, що дозволить продемонструвати загибель юніта, а саме, на місці юніта буде знаходитись як труп юніта, так і частина його екіпірування і його гроші. На даний час, після смерті юніт зникає з поля бою, на його місці нічого не залишається.

Статистика юніта (рисунок 3.10) містить в собі як основні , так і додаткові характеристики та особливості, що відповідають за витрату AP, за кількість доступних переміщень та атак у ход, за радіус атаки та багато іншого. Варто відзначити, що на момент тестування AP не витрачається, тому в самій битві вони витрачатися також не будуть.

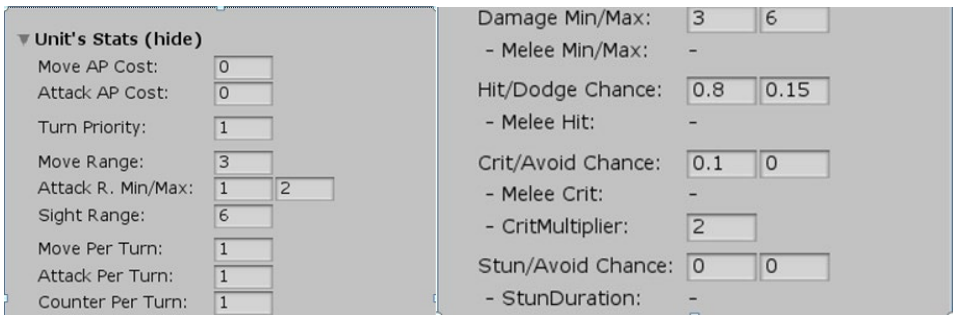

Рисунок 3.10 – Статистика списоносця в менеджері юніта

На даний момент, ще не були якісно протестовані здатності (перки) індивідуально для кожного класу юніта, так і не було розроблено індивідуальні здатності фракцій. Ще однією розробкою, що зможе урізноманітнити стратегію боротьби на полі бою є укриття та «туман війни».

Укриття буде слугувати додатковим захистом від різноманітних дальніх атак (лучника, мага і тд.), слугувати своєрідним укриттям від «радара» супротивника при включеному «туману війни». Можливо в подальшому, у укриття з'явиться свій власний показник здоров'я, при закінчені якого, укриття буде зруйновано.

Туман війні – це своєрідний показник доступного бачення суперників. Наприклад, списоносець може бачити на ту відстань, на яку він може переміститися, тому він баче відстань 3 гексів. При включені туману війні можна займатися скритним проникненням в стан ворога, що урізноманітнює тактичне та стратегічне бачення самої картини битв.

Керування юнітів на полі боя може здійснюватися 3 способами, які характерні покроковій стратегічній грі. Перший спосіб – це покроковий хід юнітів всієї фракції до тих пір, поки не закінчиться можливість ходу, після передається можливість дії іншій фракції, поки не закінчиться ОД у юнітів фракцій і гравець/AI не натисне на EndTurn. Другий спосіб схожий із першим, але при цьому почергово передається хід іншому грацю при закінчені ходу окремо взятого юніта фракції.

Штучний інтелект розроблений в 3 версіях, що урізноманітнює його тактичні навички проти гравця. Такими версіями є: агресивний, пасивний та тригерний режими. Хоча штучний інтелект на даний момент може лише «йти на ви», тобто атакувати суперника як тільки зможе його побачити, але в подальшому, при додатковій розробці глобальної карти та вдосконалюючи рельєфи на ній, я зможу навчити штучний інтелект підбирати краще гекс для атаки, щоб краще себе захистити, та можливо удосконалити якість атаки.

На даний момент існує 6 типів бафів, які на зовнішній вигляд мають різницю в кольорі правильного куба. Такими бафами є: регенерація, додаткова атака, збільшення можливої атаки в два рази, баф на ухилення (своєрідний баф на захист) та баф на збільшення дальності атаки, своєрідне кидання списа списоносцем. Ще був створений рандомний ящик, який надає один із 5 зазначених раніше бафів

Як для бафів, так і для перків були розроблені менеджери (рисунок 3.11), що допомагатимуть у підборі своєрідних ефектів. Так, наприклад, ефект на зцілення можна буде застосувати як перк одного із цілителів, фракційний перк, так і як своєрідний баф на юніта, що його підібрав.

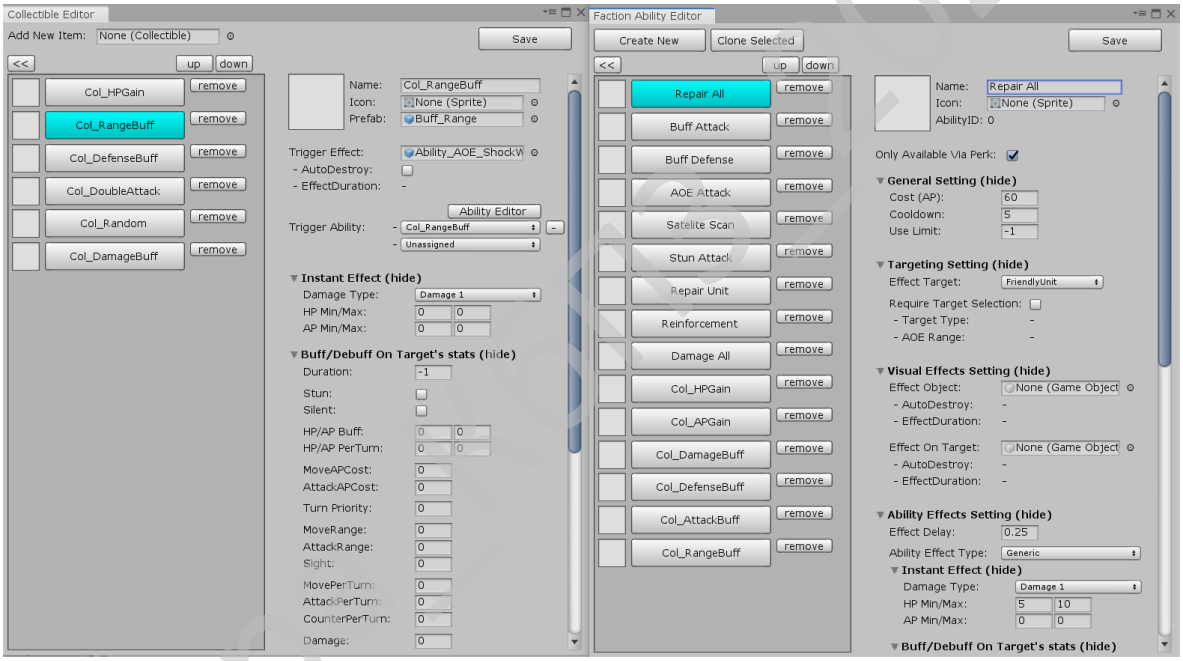

Рисунок 3.11 – Представлення інтерфейсу менеджера перків та бафі

Фракційні бафи або фракційні здатності – будуть розроблені використовуючи систему Collectible Editor та Faction Ability Editor, що в свою чергу – покращує та надає більш важливого значення в подальшому виборі фракцій. Так як при умові, що є дві фракції з представниками людей, з тими же юнітами з мінімальною різницею, стає дуже складно відрізнити їх. Тому система фракційних бафів, що знаходиться в процесі тестування – вирішить наявну проблему в різниці фракцій.

3.3 Розробка структурної схеми

При розробці структурної схеми (рисунок. 3.12.) програмного забезпечення були розглянуті основні модулі програми, що представляють головну сцену, тобто саму гру, та менеджери управління, що забезпечують підтримку та розробку програмного забезпечення.

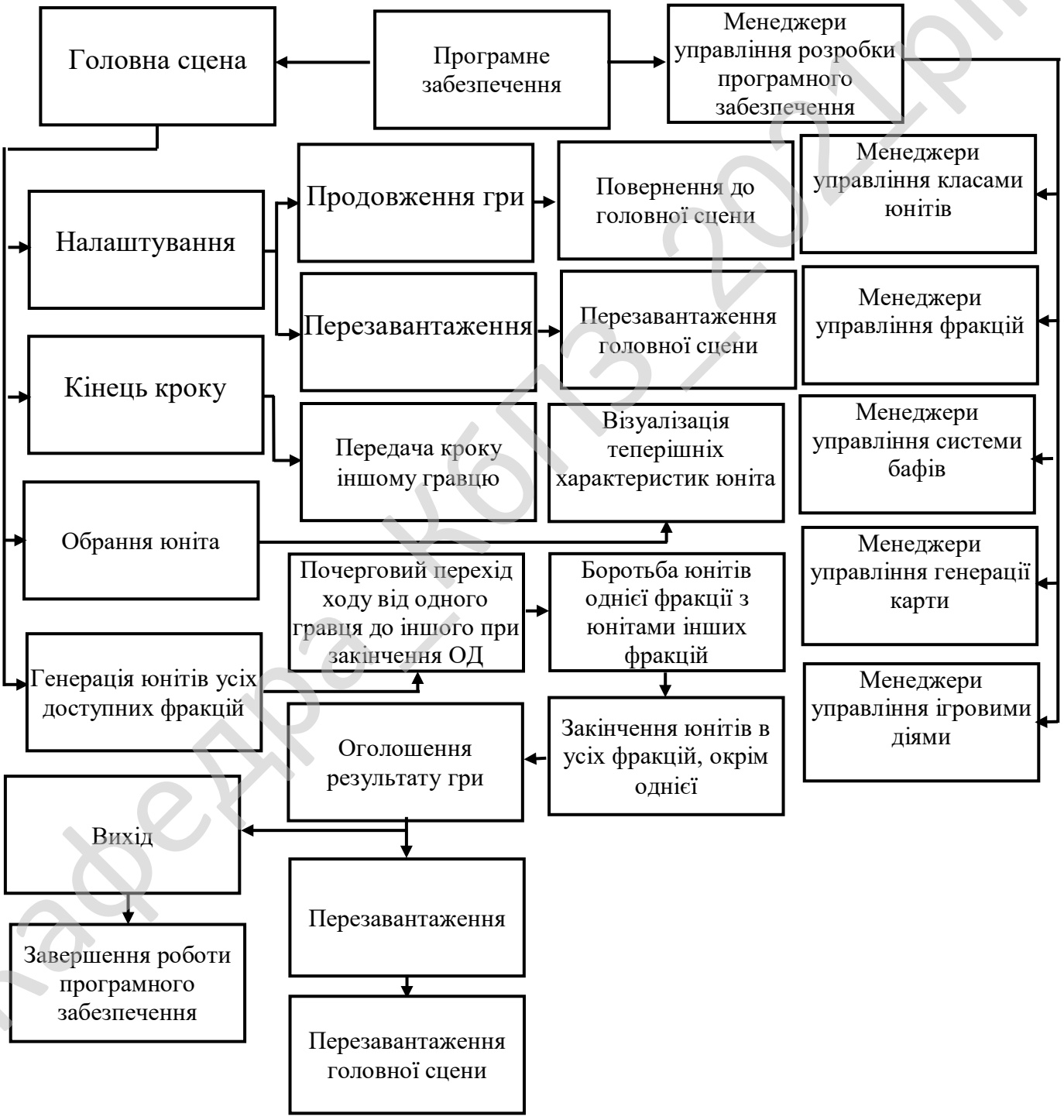

Рисунок 3.12 – Структурна схема

Як видно, структурна схема програмного забезпечення розділена на дві важливі частини, а саме на головну сцену покрокової стратегічної гри та на менеджери управління програмним забезпеченням. Щоб швидше зрозуміти структурну схему, потрібно розділити її на 2 частини та проаналізувати їх окремо, розпочнемо із головної сцени.

Головна схема поділяється на такі блоки, як налаштування, покрокова стратегічна гра, кнопку завершення ходу та вікна обрання юніта. При запуску програмного забезпечення відбувається генерування юнітів всіх фракцій, які були зазначені в менеджерах управління. Після генерування юнітів, автоматично вибирається юніт фракції та демонструється, в лівій, верхній частини, теперішня характеристика юніта та його іконка.

При активації налаштувань ми можемо обрати дві функції, першою є повернення до головної сцени, друга функція перезавантажує головну сцену. Варто пам’ятати, що при кожному перезавантаженні гри відбувається генерування юнітів на новому стартовому гексу.

Після генерування юнітів, усіх доступних фракцій, відбувається почерговий хід кожного юніта, кожної фракції, після того як юніти досягають достатньої відстані, щоб здійснити атаку, а це мінімум 1, ма ксимум 2 гекси – відбувається боротьба між юнітами, після завершення якої один із юнітів вмирає. Після смерті усіх юнітів фракції суперника, або фракції гравця, відбувається оголошення результату гри, після якого ми можемо як вийти із гри, так і перезавантажити головну сцену (саму гру).

Менеджери управління розробки програмного забезпечення були розроблені, щоб забезпечити якісний менеджмент усіх доступних сфер розробки, та в майбутньому застосувати як основу для внутрішньо ігрового редактору карт. Основними сферами менеджерів управління є: юніти, фракції, бафи та генерація карт.

В менеджері управління юнітів можна додавать нових юнітів, змінювати їх основні характеристики та особливості. Процес додавання юніта

в базу управління відбувається перенесенням скрипту Unit.cs до відповідного префаба, що відповідає за нового юніта. Після прив'язки скрипта потрібно зафіксувати в менеджері нового юніта, давши йому ім'я.

В менеджері по управлінню фракцій можна добавити нову фракцію, використовуючи та редагуючи шаблонну версію фракцій. При тестуванні було доведено, що при теперішній версії гри достатньо на одному полі битви двох фракцій, але це не перекреслює можливість мати більше ніж 2 фракції в менеджері фракцій. Щоб включити нову фракцію в боротьбу, потрібно за допомогою менеджера карти включити зону спавна нової фракції.

Менеджер управління бафами надає можливість редагувати та створювати нові бафи. Бафи бувають такі, що безпосередньо впливають на характеристики юніта, так і на його особливості. Наприклад, баф на здоров'я, діє як повне відновлення всього втраченого здоров'я, що дозволяє змінити шанси на перемогу на свою користь (якщо самі його активували).

Найголовнішим менеджером управління, на мою думку, є менеджер управління генерацією карти, завдяки якому можна змінювати як кількість самих гексів, так і утворювати «порожнини». За допомогою менеджера можлива зміна поверхні карти, додавання нових місць спавну при генерації карти, додавання як ящиків з бафами, так і самих юнітів.

При генерації нової карти, можна змінювати її розмір, змінюючи властивості ширини і довжини, змінювати розмір гексів та відстань гексів один від одного.

Щоб редагувати місця спавну юнітів та бафів, змінювати поверхню гексів та встановлювати чи очищувати укриття, потрібно активувати редактор, натиснувши на Enable Edit, таким чином активувавши вікно. Після кожного запуску гри рекомендується знову натиснути на редактор, якщо ми бажаємо редагувати.

Штучний інтелект знаходиться в ігровому контролері, в якому можна змінювати як поведінку AI в битвах, так і його тригерну систему. За

допомогою ігрового контролеру можна генерувати нову карту, юнітів, бафи зі стартом програмного забезпечення, а можна відключити галочки на генерування і отримати шаблону, пусту версію гри.

3.4 Розробка функціональної схеми

Функціональна схема (рисунок 3.13) розбита на блоки взаємодії, головними із яких є «Програмне забезпечення, що представляє з себе сукупність всіх систем в одну цілу. «Головна сцена» , що представляє з себе карту на якій відбувається ігрова дія користувача.

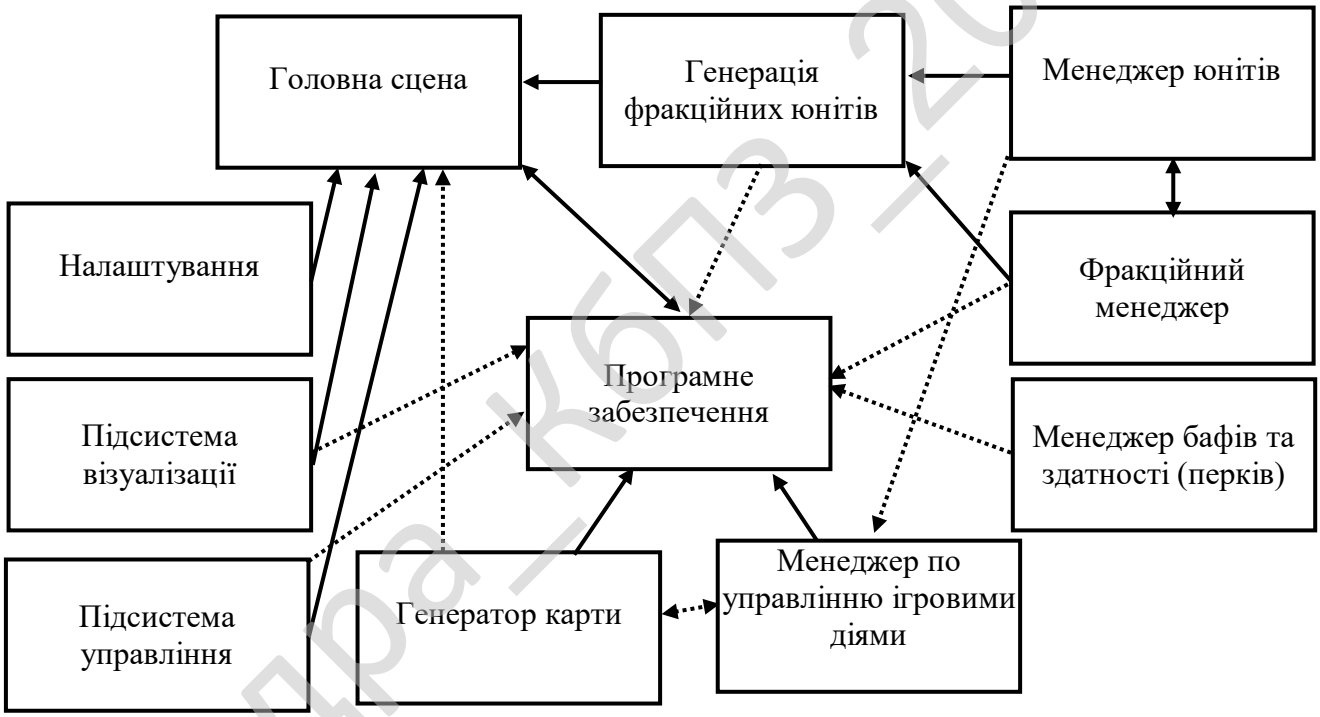

Рисунок 3.13 – Функціональна схема

Як ми бачимо по функціональній схемі, Головна сцена, включає в себе генерацію фракційних юнітів, що утворюються при запуску програмного забезпечення. Після запуску ми можемо використовувати налаштування гри, а за гру відповідатиме підсистема візуалізації, завдяки якій можливо грати у 3D-

гру, та підсистема управління, що аналізує кількість юнітів у фракцій та надає результат при поразці однієї із фракцій.

3.5 Розробка діаграми процесів

На діаграмі процесів системи (рис. 3.14) відображено взаємодію процесів системи, що розпочинаються з редагування програмного забезпечення за допомогою менеджерів управління. Одночасно можна вважати, що початком є «Головна сцена», що відображає карту битви на якій відбувається гра.

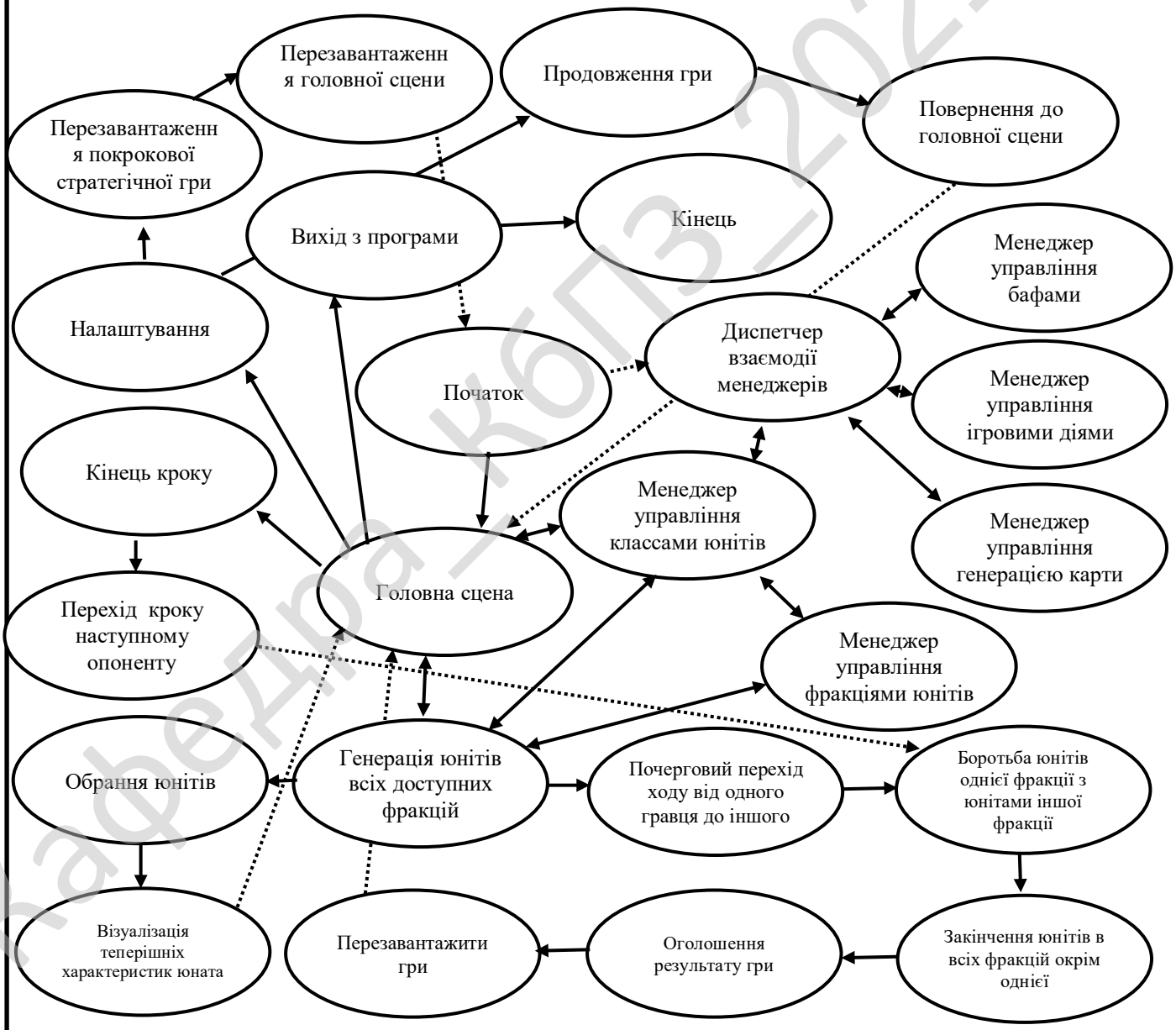

Рис 3.14 – Діаграма процесів покрокової стратегічної гри

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ ТА ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ І ПРОГРАМНИХ РІШЕНЬ

Після обрання в якості засобу програмування технологічне ядро Unity та обрання мови програмування C#, було розглянуто та створено функціональну та структурну схему, була розроблена діаграма процесів покрокової стратегічної гри.

Під час опису функціональної системи мною була розроблена концепція програмного забезпечення для системи візуалізації покрокової стратегічної гри. Була розглянута складова частина глобальної карти битв (головної сцени), а саме гекси, та як за допомогою їх ми зможемо формувати карту.

Була розглянута та продемонстрована ідея менеджерів підпрограм, за допомогою яких можна вкоротити витрату часу на створення і прописування характеристик новим юнітам. Використовуючи шаблонну версію юнітів і шаблонні версії фракцій можливо активно змінювати як юнітів, так і різноманітні фракції, використовуючи приємний інтерфейс.

Була продемонстрована статистика юніта, що є ефективним виразом різноманіття особливостей, які можна надати юнітам, і виразом можливої мікроекономіки, що буде відтворюватися шляхом витрат певної кількості одиниць енергії у юніта на різні маніпуляції, такі як: атака супротивника, застосування здібностей (перків), переміщення по карті.

Так як вже було проінформовано про різноманітні особливості як в описі функціональної системи, так і в подан ні та розкритті функціональних, структурних схем та діаграм процесів побудування покрокової стратегічної гри, то перейдемо до розробки алгоритмів системи та програмуванню рішення поставленої задачі. Для демонстрація алгоритму роботи, ми розглянемо гру користувачів за різні фракції, не затримуючи увагу на менеджерах управління.

4.1 Розробка блок–схем та опис алгоритмів функціонування системи

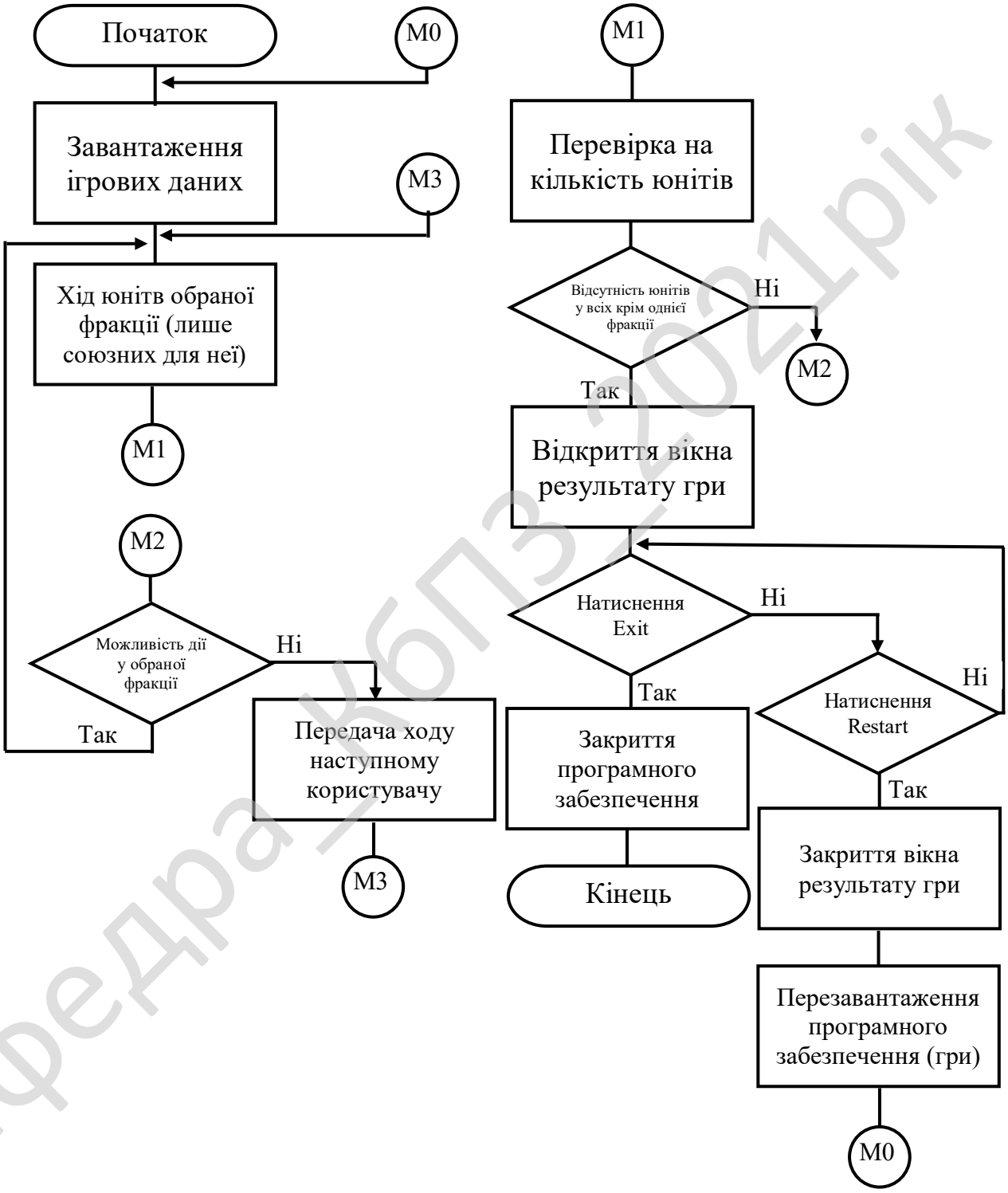

Рисунок 4.1 – Блок-схема алгоритму роботи основної системи

Розглянемо алгоритм роботи програмного забезпечення, що був представлений на основній блок-схемі системи візуалізації покрокової стратегічної гри (рисунок 4.1.) ті їх підпрограм. За допомогою основної блок-схеми ми можемо детально розглянути роботу головної частини програми, а саме нашої гри від її початку (запуску) до його завершення. А за допомогою алгоритмів підпрограм, що будуть представлені далі, ми зможемо якісно розглянути всю роботу гри, від самого початку до її логічного кінця. Також будуть пояснені моменти, що були пропущені або слабо пояснені в алгоритмі.

Програма розпочинає свою роботу із демонстрації головної сцени, що показує нам карту боротьби між фракціями. На даний момент здійснене тестування битв між двома різноманітними фракціями, використовуючи характерних юнітів. Також існує альтернативний запуск, який розпочинається з головного меню, де можна обрати місце боротьби, але він ще тестується і дороблюється, тому представлені рисунки будуть пояснювати боротьбу між ними. Варто відзначити, що суперників може бути більше ніж один, але в даному випадку, ми зосереджуймо свою увагу на наш поєдинок із одним суперником. В нашій і в команді супротивника по 2 бійці.

Так як в алгоритмі блок -схеми не було зазначено як саме формується карта, я вирішив прояснити це тим, що на даний момент карта була створена шляхом генерування до запуску гри в самому середовищі розробки. Власне карта створюється за допомогою наступної частини програми:

```
List<Transform> tileTransformList=new List<Transform>();
                Vector3 pos=Vector3.zero;
                Transform parentT=new GameObject("GridTemp").transform;
                float spaceX=spaceXHex*gridSize*GridToTileRatio;
                float spaceZ=spaceZHex*gridSize*GridToTileRatio;
                Grid grid=new Grid(_TileType.Hex, width, length, gridSize,
                GridToTileRatio);
                for(int i=0; i<width; i++){
                      float offsetZ=(i%2==1) ? 0 : (spaceZ/2);
                      int limit=i%2==1 ? length : length-1;
                      for(int n=0; n<limit; n++){
                            pos=new Vector3(i*spaceX, 0, n*spaceZ+offsetZ);
                            Transform
                      tileT=(Transform)Instantiate(tilePrefab, pos, rot);
                            tileTransformList.Add(tileT);
                            tileT.localScale*=gridSize;
                            tileT.parent=parentT;
                            tileT.gameObject.layer=TBTK.GetLayerTile();
```

```
                    Tile tile=tileT.gameObject.AddComponent<Tile>();
                    tile.type=_TileType.Hex;
                    tile.x=i;
                    tile.y=-((i+1)/2)+n;
                    tile.z=-(i/2)-n;
                    grid.tileList.Add(tile);}}
            grid.AssignIndex();
```

В цій частини коду, було упущено як саме з'являються дані в перемінних, але, на мою думку, повністю повторювати інформацію, що була надана в 3 розділи буде не раціональним, тому я вирішив лише нагадати основну частину формування самої карти.

Як ми пам'ятаємо, карта формується в результаті отримання даних по 6 змінним: width, length, gridSize, GridToTileRatio, unwalkableRate, GridManager.GridColliderType colType=GridManager.GridColliderType.Master, що отримують свою інформацію в блоці змінних в функції. В 3 розділі я детально не розглянув лише GridManager.GridColliderType colType = GridManager.GridColliderType.Master, що забезпечує обрання правильного префаба, завдяки якому і буде формуватися карта .

Після запуску гри, ми можемо спостерігати в лівій верхній частині гри налаштування, що представлена «галочкою». За допомогою цієї галочки ми можемо перезавантажити гру або вийти з неї.

На лівій правій частині спостерігається кнопка End Turn, за допомогою якої ми можемо закінчити свій хід та передати його супротивнику. Варто пам'ятати, що при натисненні хода, хід передається відразу, без попередження, тому будьте обережні при натисненні, щоб не передати хід тоді, коли це непотрібно. Хід не передається автоматично при закінченні ходу всіх юнітів гравця, але передається після завершення ходу всіх юнітів AI.

На лівій нижній та на лівій верхній частині можна спостерігати характеристику обраного юніта. Так як після завантаження нашої карти, нам автоматично вибирається союзний боєць, що ми можемо спостерігати на рисунку 4.3. Для того щоб зрозуміти, завдяки чому нам демонструється вікно характеристик юніта, була розроблена блок-схема алгоритму демонстрації вікна характеристик юніта (рисунок 4.2.).

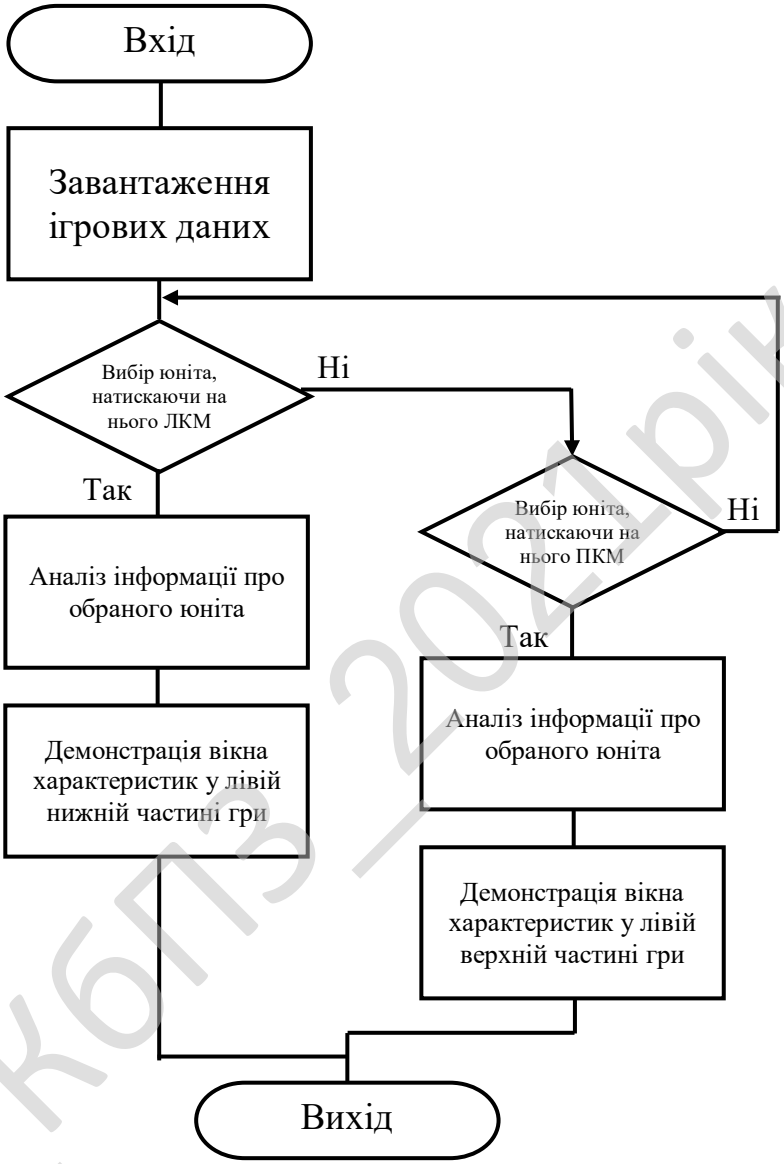

Рисунок 4.2 – Блок-схема алгоритму підпрограми, що демонструє вікна характеристик юніта, його здоров'я (HP) та очки дії (AP)

Як ми бачимо із блок-схеми, інформація, що надається в лівому нижньому вікні є той самой, що і в верхньому лівому. Різниця є лише в тому, що інформація в лівому верхньому вікні з'являється після натиснення на юніт ПКМ, а інформація на лівому -нижньому вікні, з'являється в результаті виділення союзного юніта, таким чином ми можемо ще проаналізувати який саме юніт був обраний, але простіше буде з'ясувати це за результатом характерного підсвічування під обраним юнітом. Варто пам'ятати, що

інформація передається в режимі реального часу, тобто поточні дані про обраного юніта.

Інформація, що надається про юніта обмежена і передає відомості про найважливіші характеристики нашого бійця, а саме HP – здоров'я бійця, яке позначено зеленим кольором та AP – очки дії, що позначені синім кольором. Запитальний знак, що стоїть біля характеристик юніта, буде надавати нам можливість в отриманні більш детальної інформації про юніта, яка доступна гравцю.

Рисунок 4.3 – Головна сцена, що представляє карту битви

При автоматичному обранні юніта (юніт може бути обраний самостійно, але при запуску гри юніт вибирається автоматично), ми спостерігаємо відображення можливого хода (радіус в 3 гекса від юніта), підсвічене яскраво-зеленим кольором, підсвічування слугує для якісного виразу можливого хода юніта.

Як нам відомо, у правому верхньому вікні ми спостерігаємо налаштування, що відображаються своєрідною «галочкою». Якщо ми натиснемо на нього, то спливе своєрідне меню, в якому зараз діє 2 функції, 1 - ша, це повернення до того, що було, тобто Continue , 2-ою є перезапуск битви, тобто Restart. Для більш наглядного розуміння, розглянемо рисунок 4.4.

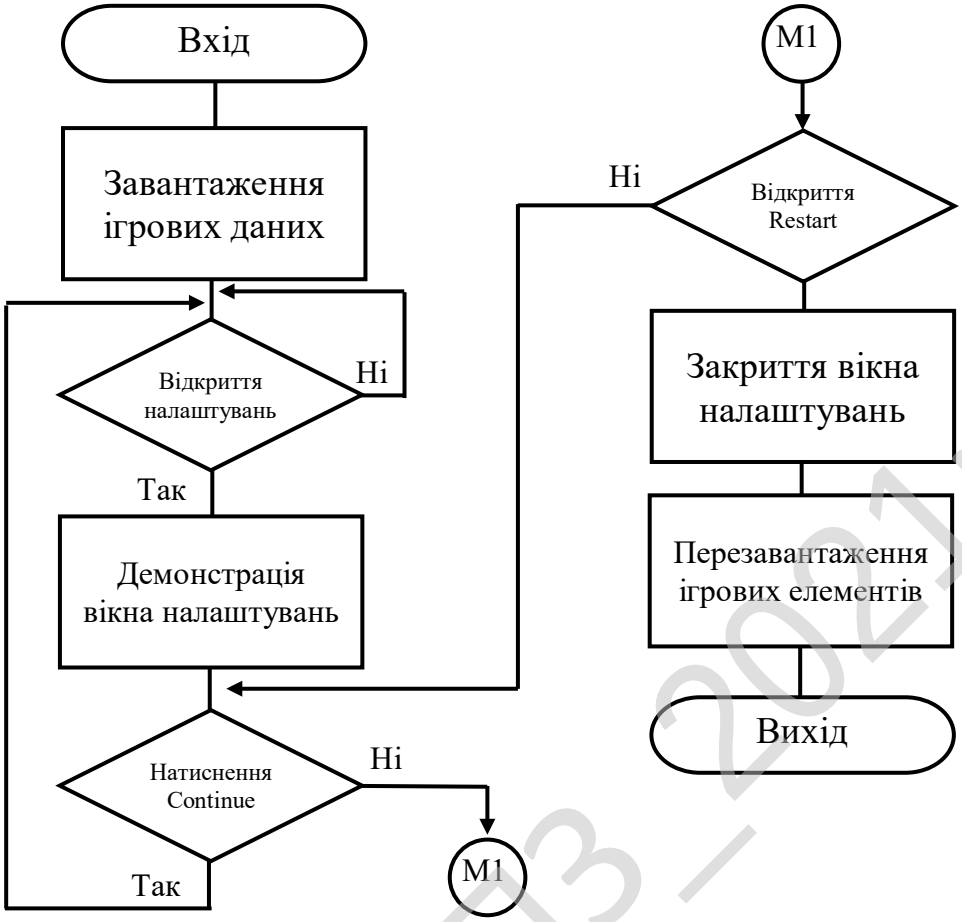

Рисунок 4.4 – Блок-схема підпрограми алгоритму, що відповідає за демонстрацію вікна налаштувань на його можливостей.

Особливе підсвічення гексів можливого ходу юніта здійснюється підкладкою під наявний гекс – яскраво-зелений гекс, hexMoveable, що і надає нам таку особливе підсвічення, така можливість відображається у наступному коді:

```
        if(GridManager.GetTileType()==_TileType.Hex){
                        moveableIndicator=hexMoveable;}
  for(int i=0; i<20; i++){
moveableIndicatorList.Add((Transform)Instantiate(moveableIndicator));
        moveableIndicatorList[i].gameObject.SetActive(false);
        moveableIndicatorList[i].parent=thisT;}
```

У кожній із юнітів фракції є 2 фази дії, 1 -ша – це пересування, 2-га – це атака. За умову, що юніт може пересуватися і є така можливість, він може перемістити свого бійця на підсвічене зеленим кольором поле. У даному прикладі представлений списоносець, його радіус атаки дорівнює 2-ом

гексам. Коли супротивник буде в радіусі атаки, то нам про це повідомлять, виділивши гекс червоним кольором під ворожим бійцем. Для кращого розуміння, розглянемо алгоритм роботи AI (рисунок 4.5.).

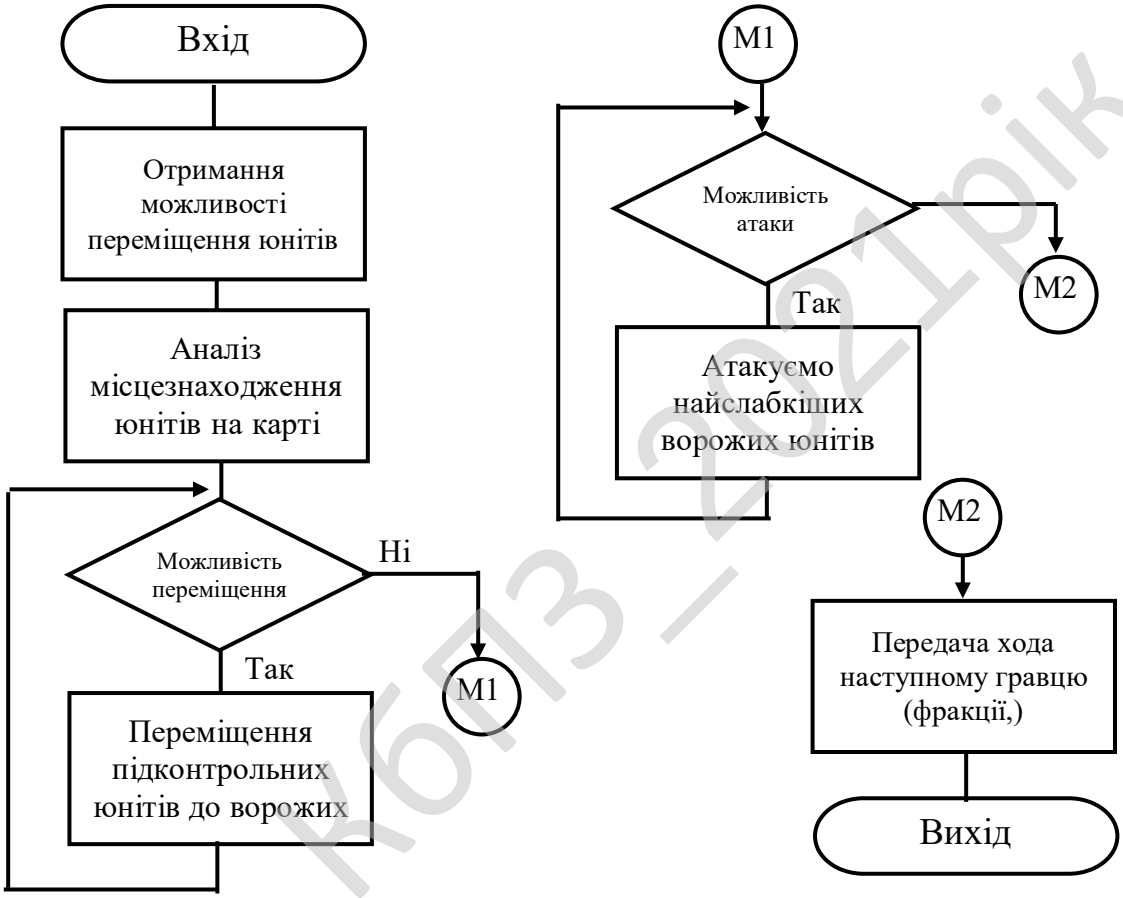

Рисунок 4.5 – Блок-схема алгоритму підпрограми керування AI

Коли ворожий юніт знаходиться в радіусі атаки, тоді ми можемо атакувати його, але варто звернути увагу на те, що ми не можемо з 100% впевненістю сказати, вдасться чи ні наша атака, є лише 70% імовірність якісного результату. В одному із тестових спроб я промахнувся 3 рази підряд, коли у останнього ворога залишалось лише 3 HP, тоді я програв.

Після завершення всіх можливих ходів у союзних юнітів, ми повинні самостійно натиснути на End Turn, ця дія запустить скрипт, умовою якого є передача хода опоненту.

Суперник, що грає за ворожу фракцію, поділяється за способом управління. Існує два способи управління фракцій, такі як самостійне управління, тобто користувач бере на себе управління фракцією, та управління за допомогою AI.

Щоб урізноманітнити алгоритм дії штучного інтелекту, тим самим покращуючи гру, було розроблено декілька видів алгоритмів дії AI, такі як:

Пасивний(Passive), коли юніт не буде рухатися, якщо в зоні бачення окремо обраного юніта фракції немає ворожого юніту. Варто відзначити, що він діє навіть тоді, коли Fog-Of-War не використовується. Тобто, як і в Fog-Of-War – ми бачимо лише той радіус на який може перемістить юніт, так і в пасивному режимі, ворожий юніт, аналізує лише радіус свого можливого переміщення, тому за радіусом – він не бачe супротивника

Спусковий механізм (Trigger), - це механізм в якому юніт не буде рухатися та атакувати, якщо в зоні бачення юніта не буде ворожого юніта . При появі ворожого юніта – юніт агресивно зреагує та нападе на нього. При умові включення режиму спостереження, при вході в зону бачення юніта, ворожого юніта – юніт атакує відразу як тільки ворожий юніт буде в зоні можливої атаки, для списоносця це відстань в один гекс.

Агресивний(Aggressive), коли юніт буде постійно рухатися, шукаючи потенційну ціль. Агресивний режим активно використовується при тестуванні гри, тому що штучний інтелект ставить мету на знищення ворога, що полегшує гравцю розробити детальний план атаки.

Якщо кількість HP юніта, дорівнює 0, то спрацьовую скрипт «смерть», для даного випадку я ще не зробив анімацію смерті та взагалі «трупа» або лута, що може залишити юніт, тому при спрацюванні скрипту «смерть», юніт, що втратив своє здоров’я, зникає з карти битви.

Варто пам’ятати, що коли кожен з юнітов союзної або ворожої фракції втратить всі свої очки здоров’я, спрацює скрипт «Результат битви», що повідомить нам про те, що ми або перемогли або програли.

Повідомлення про результат битви дасть нам лише один варіант дії, який направлений на перезавантаження гри. Після його активації відбудеться перезапуск гри, і ми можемо спробувати ще один раз.

Після розглядання алгоритму роботи системи візуалізації покрокової стратегічної гри, варто відзначити, що алгоритм роботи штучного інтелекту був розроблений, спираючись на три його режими роботи, що хоч і не дуже відрізняються після активації тригеру, але є досить цікавим. Тому я вирішив розповісти максимально якісно про алгоритм роботи штучного інтелекту.

Робота штучного інтелекту розпочинається з вибору його режиму дії, якщо режим роботи не визначений, буде обрано пасивний режим.

```
public _AIMode mode=_AIMode.Passive;
public static _AIMode GetAIMode(){ return instance.mode; }
```

Для переміщення юнітів штучний інтелект використовує функцію FactionRoutine, що переміщує юнітів усієї фракції, один за одним.

```
IEnumerator FactionRoutine(Faction faction){
                GameControl.DisplayMessage("AI's Turn");
                yield return new WaitForSeconds(0.3f);
```

Наступний список створений для того , щоб жодна одиниця не була пропущена, навіть при умові, що одна із них якось могла бути знищена, наприклад, контратакою, тому такий лист слугує страховкою, щоб система не вилетіла або не потрапила до вічного циклу. Після створення списку відбувається знаходження шляху до цілі та перехід до неї.

```
                List<Unit> unitList=new List<Unit>( faction.allUnitList );
                for(int i=0; i<unitList.Count; i++){
                        AIDebug("Start moving unit
"+unitList[i].gameObject.name);
                        yield return new WaitForSeconds(unitInterval);
                        if(unitList[i].IsStunned()) continue;
                        _AIMode activeAIMode=!faction.useDefaultAIMode ?
faction.aiMode : mode ;
```

Перевіряється чи може один із юнітів фракції переміщуватися або атакувати, при можливості однієї із дій – відбувається переміщення або атака.

```
                while(unitList[i].moveRemain>0 || unitList[i].CanAttack()){
                        AIDebug("Moving unit "+unitList[i].gameObject.name+"
(move remain:"+(unitList[i].moveRemain>0)+", can
attack:"+unitList[i].CanAttack()+")");
                                StartCoroutine(MoveUnitRoutine(unitList[i],
activeAIMode));
```

```
                              while(movingUnit) yield return null;
                              if(unitList[i]==null) break;
                              if(unitList[i].moveRemain>0 ||
unitList[i].CanAttack()) yield return new WaitForSeconds(.1f);
                        }
if(GameControl.GetGamePhase()==_GamePhase.Over) yield break;
                  }
                  GameControl.EndTurn();}
```

Під час завершення ходу автоматично відбувається натиснення на End Turn, а взагалі, відбувається лише виклик функції, що слугує завершенню ходу та передача його наступній фракції.

Попередня частина коду слугує тільки тоді, коли обрання режиму штучного інтелекту слугує для переміщення юнітів усієї фракції, а не тільки одного, наступний викликається тільки тоді, коли потрібно переміщувати лише одну одиницю юнітів, фракцій.

```
IEnumerator SingleUnitRoutine(Unit unit){
                  GameControl.DisplayMessage("AI's Turn");
                  yield return new WaitForSeconds(0.1f);
                  if(!unit.IsStunned()){
                        Faction
                        faction=FactionManager.GetFaction(unit.factionID);
                        _AIMode activeAIMode=!faction.useDefaultAIMode ?
                        faction.aiMode : mode ;
```

Наступна частина перевіряє чи може юніт переміщуватися по карті або атакувати ворожого юніта, при можливості однієї із дій, здійснює аналіз найкращого місця атаки чи переміщення і виконує закладені функції роботи штучного інтелекту.

```
                        while(unit.moveRemain>0 || unit.CanAttack()){
                              StartCoroutine(MoveUnitRoutine(unit,
                              activeAIMode));
                              while(movingUnit) yield return null;
                              if(unit==null) break;
                              if(unit.attackRemain>0 || unit.CanAttack()) yield
                        return new WaitForSeconds(.1f);}}
                  GameControl.EndTurn();}
```

При завершені ходу, як і в режимі роботи переміщення юнітів усієї фракції, автоматично відбувається натиснення на End Turn, а взагалі, відбувається лише виклик функції, що слугує завершенню ходу та передача його наступній фракції.

Код, що відображає переміщення юнітів, виглядає наступним чином:

```
IEnumerator MoveUnitRoutine(Unit unit, _AIMode activeMode){
```

```
                TBTK.OnUnitSelected(unit);
                movingUnit=true;
                if(activeMode!=_AIMode.Aggressive && !unit.trigger){
                      AIDebug("unit "+unit.gameObject.name+" is not
          triggered");
```

Після перевірки на агресивний тип взаємодії відбувається своєрідне патрулювання при включеному агресивному стилі поведінки,, що слугує знаходженню ворожих цілей та спробі їх знищення.

```
          if(!untriggeredUnitMove){
                StartCoroutine(EndMoveUnitRoutine());
                            unit.moveRemain=0;      unit.attackRemain=0;}
                      else{
                            if(Random.value<0.5f){
                                  StartCoroutine(EndMoveUnitRoutine());
                                  unit.moveRemain=0;
                                  unit.attackRemain=0;}
                            else{
                                  AIDebug("Randomly move unit
                                  "+unit.gameObject.name+" anyway");
                                  List<Tile>
walkableTilesInRange=GridManager.GetTilesWithinDistance(unit.tile, Mathf.Min(1,
unit.GetEffectiveMoveRange()/2), true);
                                  if(walkableTilesInRange.Count>0)
unit.Move(walkableTilesInRange[Random.Range(0, walkableTilesInRange.Count)]);}}
```

Після завершення підрахунку та реалізації переміщення, юніт повідомляє системі про успішність своєї операції та запускає функцію завершення переміщення

```
                AIDebug("End unit "+unit.gameObject.name+" turn");
                StartCoroutine(EndMoveUnitRoutine());
                yield break;}
          Tile targetTile=Analyse(unit, activeMode);
```

Щоб зрозуміти, де знаходиться юніт, прокласти йому характерний путь до ворога та не пропустити його і, можливо помилково, не спробувати зайняти його місце, потрібна наступна функція:

```
                      if(CameraControl.CenterOnSelectedUnit()){
                      bool visible=false;
                      if(unit.tile.IsVisible()) visible=true;
                      else if(targetTile!=unit.tile){
                            List<Tile> path=unit.GetPathForAI(targetTile);
                            for(int i=0; i<path.Count; i++){
                                  if(path[i].IsVisible()){
                                        visible=true;
                                        break;}}
                            targetTile=path[path.Count-1];
                            Debug.DrawLine(unit.tile.GetPos(),
targetTile.GetPos(), Color.red, 2);}
                      if(visible){
                            CameraControl.OnUnitSelected(unit, false);
                      while(CameraControl.IsLerping()) yield return null;}
```

Якщо в радіусі можливого переміщення немає ворога, то таргетна система дозволяє юніту переміщуватися як тільки він хоче (рандомно вибирає гекс, який не зайнятий союзним гексом) і переміщується на нього.

```
                if(targetTile!=unit.tile){
                      unit.Move(targetTile);
                      yield return new WaitForSeconds(.1f);     //wait until
the unit has moved into the targetTile
                      while(!TurnControl.ClearToProceed()){

                            AIDebug("waiting while unit is moving");
                            yield return null;}}
```

Якщо по якійсь причині юніт був знищений навиком спостереження, то відбуваються наступні кроки:

```
                if(unit==null || unit.HP<=0){ //in case unit is destroyed by
                      overwatch
                      StartCoroutine(EndMoveUnitRoutine());
                      yield break;}
                for(int i=0; i<targetTile.hostileInRangeList.Count; i++){
                      if(targetTile.hostileInRangeList[i].unit==null ||
targetTile.hostileInRangeList[i].unit.factionID==unit.factionID){
                            targetTile.hostileInRangeList.RemoveAt(i);i-=1;}}
```

Якщо в зоні дії юніта є ворожий юніт фракції , то тригерна система подасть своєрідний сигнал на атаку, такий сигнал подає наступна частина:

```
                if(targetTile.hostileInRangeList.Count>0){
                      if(unit.CanAttack()){
                            AIDebug("waiting, unit "+unit.gameObject.name+"
                            is attacking");
          int rand=Random.Range(0, targetTile.hostileInRangeList.Count);
          unit.Attack(targetTile.hostileInRangeList[rand].unit);}
                      else{ if(unit.moveRemain>0) unit.moveRemain-=1;}}
                else{if(unit.moveRemain<=0) unit.attackRemain=0;}
                AIDebug("End unit "+unit.gameObject.name+" turn");
                StartCoroutine(EndMoveUnitRoutine());
                yield return null;}
```

При завершені кроку та всіх можливих атак, відбувається переосмислення флагів, які були нанесені раніше і які займали ті чи інші юніти:

```
    IEnumerator EndMoveUnitRoutine(){
                while(!TurnControl.ClearToProceed()){
                      AIDebug("Waiting for all sequence to complete");
                      yield return null;}
                yield return null;
                movingUnit=false;}
```

Щоб знати, куди саме потрібно перемістити юніта, потрібно проаналізувати сітку, тому наступний код слугує саме цьому.

```
private Tile Analyse(Unit unit, _AIMode activeMode){
```

Далі аналізуємо всі доступні гекси у певному радіусі на можливість переміщення.

```
List<Tile>
walkableTilesInRange=GridManager.GetTilesWithinDistance(unit.tile,
unit.GetEffectiveMoveRange(), true);
walkableTilesInRange.Add(unit.tile);
```

Аналізуємо всі доступні гекси в певному радіусі юніта на можливість наявності ворога в межах дії радуса .

```
List<Unit> allHostileInSight =
FactionManager.GetAllHostileUnit(unit.factionID);
if(GameControl.EnableFogOfWar()){
for(int i=0; i<allHostileInSight.Count; i++){
if(!FogOfWar.IsTileVisibleToFaction(allHostileInSight[i].tile,
unit.factionID)){
allHostileInSight.RemoveAt(i); i-=1;}} }
if(GameControl.EnableCover()){
Tile tile=AnalyseCoverSystem(unit,
walkableTilesInRange, allHostileInSight);
if(tile!=null) return tile;}
```

При умові, що в радіусі є ворожі юніти, вступає в силу наступне:

```
if(allHostileInSight.Count>0){
```

Заповнюємо список ворожих юнітів в радіусі, а потім відфільтровуємо через прохідні плитки в радусі, що входять до діапазон, що сканує на ворожі юніти.

```
List<Tile> tilesWithHostileInRange=new List<Tile>();
GridManager.SetupHostileInRangeforTile(unit,
walkableTilesInRange);
for(int i=0; i<walkableTilesInRange.Count; i++){
if(walkableTilesInRange[i].GetHostileInRange().Count>0)
tilesWithHostileInRange.Add(walkableTilesInRange[i]);}
```

Якщо плитка з діапазоном tilesWithHostileInRange непорожня після процесу, це означає, що є плитки, в які блок може переміститися і атакувати, повернути одну з плиток з Hostile In Range, щоб блок міг атакувати:

```
if(tilesWithHostileInRange.Count>0){
```

Якщо плитка одиничного струму є однією з тих плиток, які ворогують, просто залишайтесь нападаючими:

```
if(tilesWithHostileInRange.Contains(unit.tile)){
```

Після рандомізуємо його, щоб пристрій рухався, але не залишався на місці весь час:

```
        if(Random.Range(0f, 1f)>0.25f) return unit.tile;}
             return tilesWithHostileInRange[Random.Range(0,
tilesWithHostileInRange.Count)];}
```

Якщо взагалі немає потенційної цілі, перевіряємо, чи є в підрозділі попередній нападник, якщо він є, ідемо за останнім ворожим юнітом:

```
                    if(unit.lastAttacker!=null) return unit.lastAttacker.tile;
```

Для агресивного режиму з вимкненим Fog Of War спробуємо рухатися до найближчого підрозділу ворожого юніта:

```
        if(activeMode==_AIMode.Aggressive && Random.Range(0f, 1f)>0.25f){
                            List<Unit>
allHostile=FactionManager.GetAllHostileUnit(unit.factionID);
                            float nearest=Mathf.Infinity; int nearestIndex=0;
                            for(int i=0; i<allHostile.Count; i++){
                                  float
dist=GridManager.GetDistance(allHostile[i].tile, unit.tile);
                                  if(dist<nearest){
                                        nearest=dist;
                                        nearestIndex=i;}}
                            return allHostile[nearestIndex].tile;}
```

Якщо і справді немає ворожого юніта, то ми просто переміщуємо випадковим чином в одну із доступних гексів:

```
                    int rand=Random.Range(0, walkableTilesInRange.Count);
```

Очищуємо у списку hostileInRange для всіх рухомих плиток, тому на всякий випадок, якщо список не порожній (hostileInRange не чиститься після кожного ходу), тому блок не намагається атакувати що-небудь, коли він рухається в targetTile:

```
        walkableTilesInRange[rand].SetHostileInRange(new List<Tile>());
             return walkableTile
```

4.2 Захист розробленого програмного забезпечення

Захист інформації програмного забезпечення системи візуалізації покрокової стратегічної гри буде здійснюється за допомогою системи, яка зберігає дані за допомогою криптографічної програми, що перетворюватиме інформацію, яка надається користувачем за допомогою криптографічного захисту. Криптографічний ключ, що буде використовуватись, генерується самостійно і складається з отриманих даних, які пройшли декілька різних

шифрувань.

При розробці цього алгоритму головна увага буде приділятись таким властивостям і ідеям: використання великої (приблизно 2 Кбайт) таблиці T, одержуваної з великого 160-бітного ключа; чергування арифметичних операцій (додавання і побітовий XOR), використання внутрішнього стану системи, яке явно не проявляється в потоці даних (значення n1, n2, n3 і n4, які змінюють регістри в кінці кожної ітерації); використання відмінних один від одного операцій в залежності від етапу ітерації і її номера.

Для шифрування і розшифрування кожного байта тексту шифр SEAL вимагає близько чотирьох машинних тактів. Він працює зі швидкістю приблизно 58 Мбіт/с на 32-бітному процесорі з тактовою частотою 50 МГц і є одним з найшвидших шифрів.

Алгоритм роботи мого захисту є наступним:

Спочатку генеруємо ключ, що буде складати 32 символи, які в свою чергу складаються із 8 наборів 4-х символьних зв’язків різноманітних випадковим чином обраних методів шифрування.

За допомогою такого ключа і буде проводитися криптографічне шифруваннями по методу Software-optimized Encryption Algorithm.

Але на даний час розробка по методу SEAL все ще триває, тому варто відзначити, що на даний час, моя гра залишається беззахисною. Але я вважаю, що незабаром я розроблю інший захист, який і буде застосований.

5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програмне забезпечення системи візуалізації покрокової стратегічної гри була розроблене з метою придбання нових знань, умінь і навичок в області безпеки життєдіяльності. Із-за своєї головної складової, ігри можуть використовуватись в закріплюючих і контролюючих функціях та будуть сприяти відпрацюванню наявних навичок безпечної поведінки.

Під час битви, за допомогою ігор, людям буде простіше виявляти і розвивати найбільш важливі здібності і навички, притаманні особистості, для її безпечного економічного та стратегічного розвитку. Наприклад, якщо у нашого юніта виявиться менша кількість одиниць здоров'я, чим у супротивника, та біля нас буде знаходитись баф на здоров'я, то ми за допомогою бафа вилікуємося, тим самим продемонструємо свою тактичну перевагу, економічну та розумову дію, яка сприятиме розвитку.

Також варто відзначити, що завдяки розробці великої кількості менеджерів управління, виявляється легким змінювати та адаптувати систему. На даний час існують такі менеджери, як менеджер управління фракціями, менеджер юнітів і тд, завдяки ним ми можемо відтворити хоча б юнітів різноманітних фентезі груп. І, якщо учень захоче створити свій світ фентезі, який буде складатись із гномів, орків, ельфів, людей і тд, то йому залишат ься лише підібрати відповідні префаби, настроїти характеристики юнітів та їх приналежність фракції. Таким чином, я вважаю, що мою гру можна допустити до впровадження в освітніх закладах, як метод розвитку у людей стратегічного та тактичного мислення, прояв фантазії та економічного ентузіазму.

Завдяки тому, що система була розроблена на Unity, вона дозволяє, з легкими виправленнями, впроваджуватися в усі доступні системи. Адже відомо, що Unity є міжплатформною, а гра була розроблена за допомогою C#.

Щоб якісно впроваджувати систему в промислову експлуатацію, потрібно зрозуміти, на які саме менеджери можна сподіватися, та за допомогою яких можливі зміни і різноманіття в навчальному та ігровому форматі.

Тому розпочнемо із фракційного менеджера (рисунок 5.1). Фракційний менеджер був створений, щоб фільтрувати юнітів, що були розроблені і додані до менеджеру юнітів. За допомогою менеджера, можна змінювати колір, кількість одиниць, тип управління фракції (гравець або AI), фракційні здатності (перки).

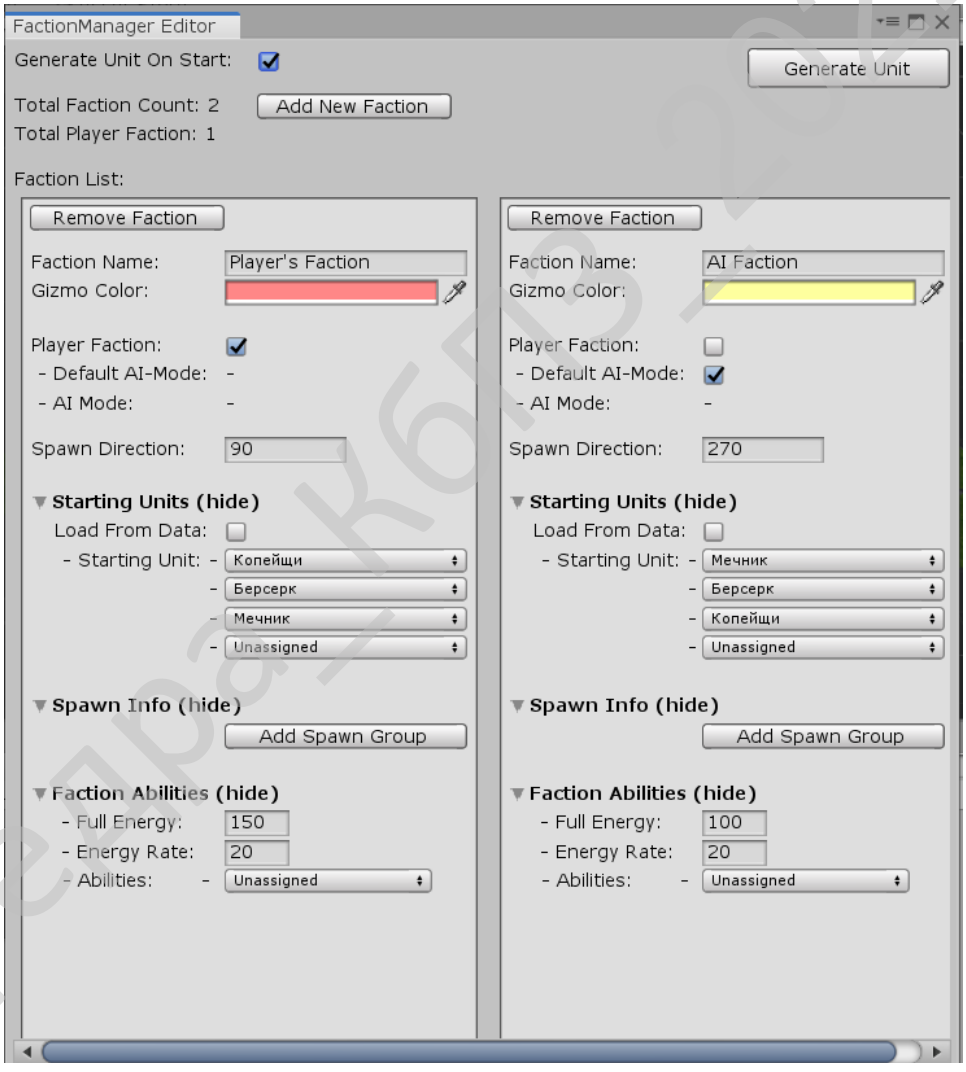

Рисунок 5.1 – Інтерфейс фракційного менеджеру

Список одиниць юнітів, що представлені в Starting Units, можна загружати за допомогою даних, які будуть збережені в окремому файлі, і при бажанні зашифровані, щоб зберегти в цілісності пак юнітів.

За допомогою же кнопки Add New Fraction, ми можемо створити нову фракцію, використовуючи відповідний шаблон, що дозволить урізноманітнити кількість фракцій на поле битви, та надасть можливість розвиватися разом із друзями та колегами. Шаблон версії ми можемо спостерігати на рисунку 5.2

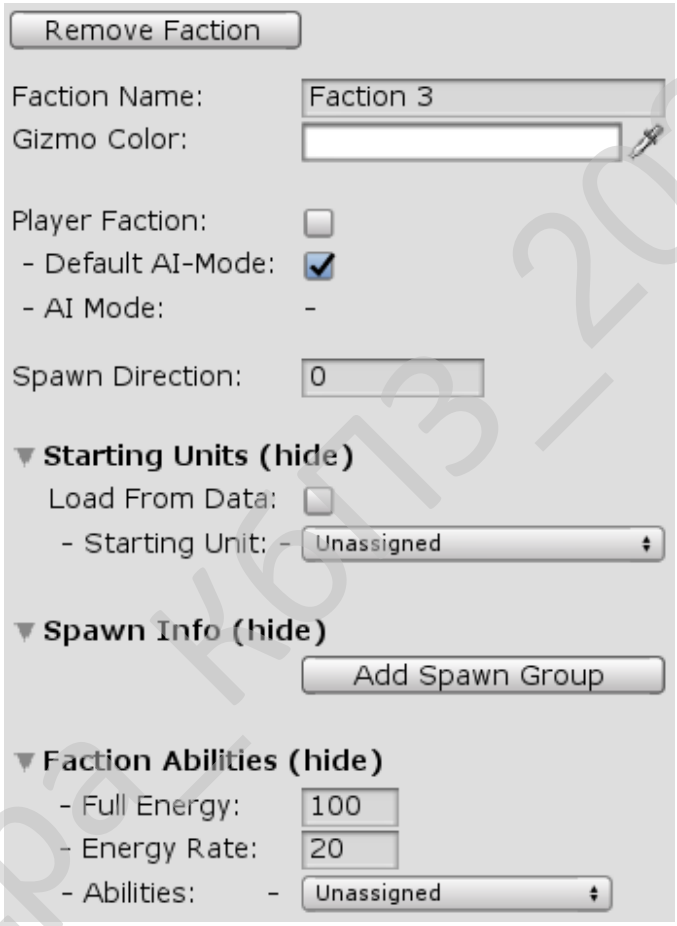

Рисунок 5.2 – Демонстрація шаблону нової фракції

Менеджер ігрової дії (рисунок 5.3.) був розроблений для слідкування за основними подіями в самій грі та для редагування як поля битви, так і стилю поведінки фракцій, за допомогою модуля поведінки AI. За допомогою ігрового менеджера можливо змінювати агресивний метод поведінки штучного інтелекту на пасивний, змінювати тип модуля переміщення юнітів в фракціях.

Також, до розгляду, менеджером ігрової дії пропонується велика кількість додаткових функцій, що зможуть урізноманітнити гру. Наприклад, якщо відключити використання АР для атаки чи переміщення, то АР збільшиться, що дозволить застосувати їх для своїх унікальних перків, юніта.

Плюс до всього, важливою особливістю ігрового менеджера є можливість зміни сценаріїв. Завдяки цій розробленій властивості можна побудувати цілий лабіринт із рівневих сценаріїв, що не тільки урізноманітнить, але і взагалі змінить гру. Так, якщо в кожному сценарії потрібно буде виконати якесь унікальне завдання, наприклад, виграти за допомогою списоносця і активного бафа на збільшення радіусу атаки, гравець отримає доступ до скритого рівня.

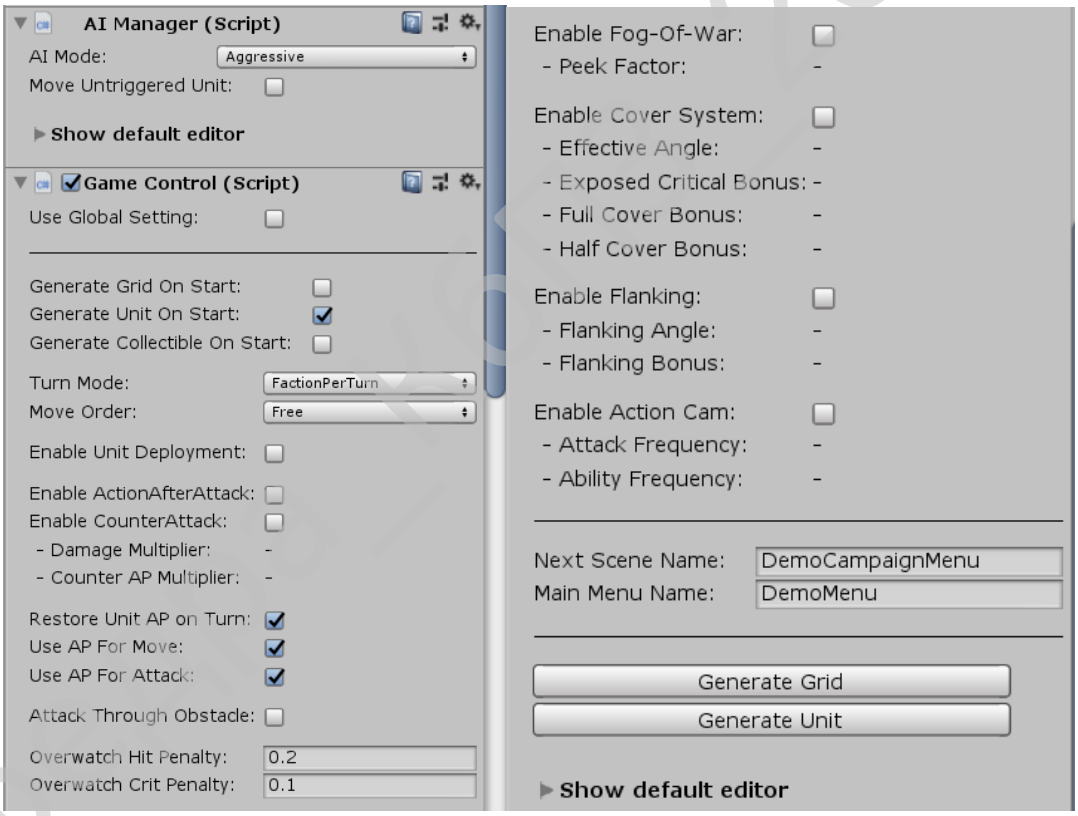

Рисунок 5.3 – Інтерфейс менеджера ігрової діяльності в Unity

Завдяки менеджеру, що може генерувати та редагувати карту в головному сценарії (рисунок 5.4), ми можемо змінювати не тільки розмір, а й сам тип карти, тобто змінювати гекси на квадрати і навпаки.

За допомогою Show Grid Prefab Assignment Setting, нам дається можливість тестувати та запускати генерацію карти бойових дій. Але, як я вже зауважував, якщо ми будемо змінювати характеристики в Show Grid Setting, а саме змінювати UnwalkableRate, тобто при створенні порожнинних місць, ми можемо в один момент дійти до того, що UnwalkableRate виявиться надто великим. Дана подія з нівелює можливість боротися з супротивником, тому варто виставляти не більше 0.4 пункта цієї характеристики.

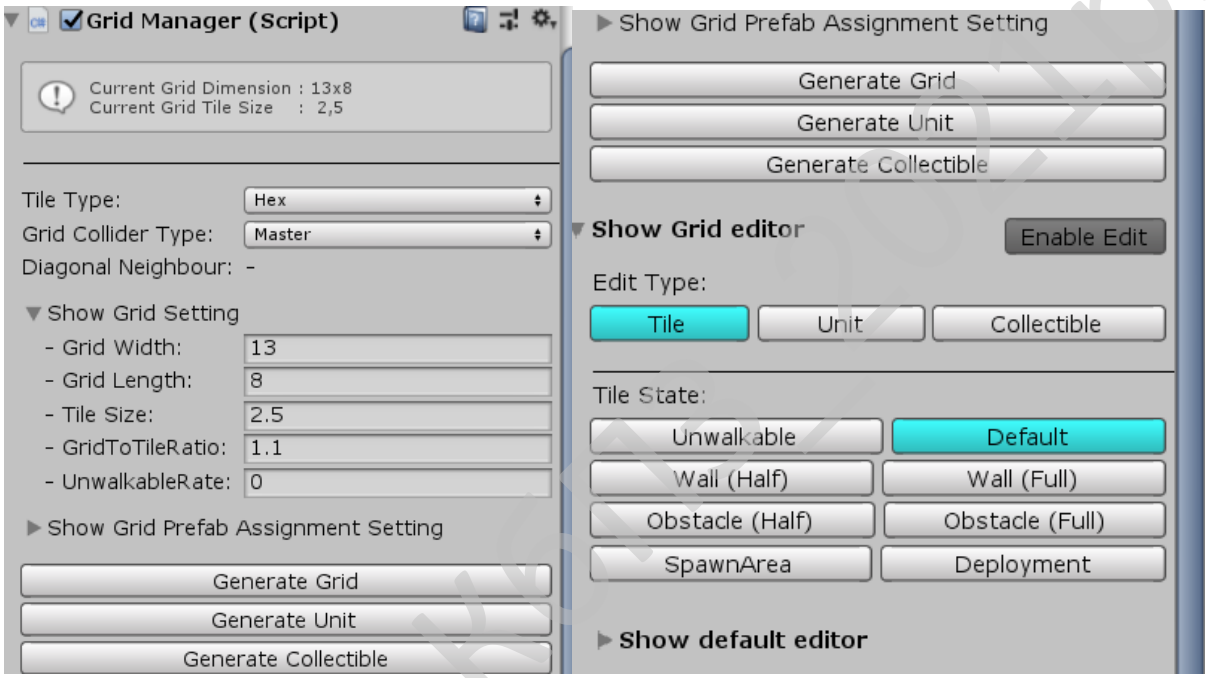

Рисунок 5.4 – Інтерфейс менеджера генератора карт в Unity

За допомогою Edit Type є можливим змінювати місткість вкладки, що спостерігається в The State. The State демонструє нам можливості змін, що притаманні карті, тобто ми можемо збільшувати або зменшувати висоту гекс а, додавати або видаляти систему укриттів (тестується). Але головним є те , що саме завдяки ігровому менеджеру та менеджеру фракцій ми можемо змінювати та створювати нові фракції. Так, завдяки менеджеру фракцій і менеджеру юнітів створюються фракції та їх наповненість, а ось завдяки ігровому менеджеру відбувається можливою реалізація їх в грі.

Завдяки категорії Collectible на карту додаються різноманітні бафи, але варто пам’ятати, що для активації змін потрібно натиснути на Enable Edit.

І наостанок підкреслю, що завдяки Unit на карту не тільки додаються нові юніти у потрібних місцях, але юніти вставляються у різні фракції. Завдяки системі туману війни та системі укриття відкривається неосяжний простір для творчості та втілення багатьох цікавих ідей та задумів.

6 НАУКОВА НОВИЗНА

Метою наукової роботи було створення метода обрання системи комп'ютерної візуалізації з врахуванням вимог що до розв'язуваних задач.

Задачі, що були розглянути для забезпечення якісного оцінювання комп'ютерної візуалізації були наступними :

1. Провести класифікацію вимог до систем візуалізації та пов'язати ці вимоги з задачею, яку поставлено до системи візуалізації.
2. Провести класифікацію систем комп'ютерної візуалізації з врахуванням визначених вимог.
3. Побудувати метод визначення комп'ютерних систем візуалізації, які задовольнятимуть визначеним вимогам.

В результаті виконання поставлених завдань, що були розглянути в виконаний в 3-му розділі, 1 підрозділі, було проведено класифікацію вимог до систем візуалізації та пов'язано ці вимоги з задачею, яку поставлено до системи візуалізації. За результатами класифікації вимог, сформовано класифікацію систем комп'ютерної візуалізації з врахуванням визначених вимог.

На основі введених класифікацій побудовано метод визначення комп'ютерних систем візуалізації, який використано для проектування ігрового програмного забезпечення покрокової стратегії, і в результаті отримано рекомендації що до використання фреймворків Unreal Engine або Unity3D.

Створений метод обрання системи комп'ютерної візуалізації є універсальним, і його можна використати для проектування програмного забезпечення. Метод дає системний підхід до розв'язання алгоритмічно невизначеної задачі обрання системи комп'ютерної візуалізації за допомогою експертних оцінок та багатокритеріальних порівнянь.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми дипломного проекту

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці).

В магістерській роботі були проведені дослідження та виконана програмна реалізація системи обробки навігаційних даних.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	40 (2 ост. цифри № зал* 10^1)
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б

Продовження таблиці 7.1

1	2	3
6. Складність алгоритму (1, 2, 3)	–	2
7. Кількість макетів вхідної інформації	–	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	1
10. Попередній досвід (1-6)	–	4
11. Гнучкість проекту ПП (1-6)	–	2
12. Детальність проекту ПП (1-6)	–	3
13. Рівень спрацьованості колективу (1-6)	–	3
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	5
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	1
17. Складність кінцевого програмного продукту (1-6)	–	3
18. Необхідний рівень забезпечення повторного використання (1-6)	–	3
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	3
20. Вимоги до швидкодії ПП (1-6)	–	4
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	1
23. Професійний рівень аналітиків (1-6)	–	4
24. Професійний рівень програмістів (1-6)	–	2
25. Постійність складу команди розробників (1-6)	–	4
26. Досвід розробки додатків (1-6)	–	1
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	1
29. Досвід роботи з програмними інструментами розробки (1-6)	–	2
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	1
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	1
32. Вартість ПЗ у розробника (НМА), грн.	–	20000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	50
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП як одна з найбільш тривалих і трудомістких робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \, \text{Size}^B, \qquad (7.1)$$

де: A – коефіцієнт Боема, $A = 2{,}45$;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

В – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001 \sum W_i, \quad (7.2)$$

де: *Wi* – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(1{,}62+4{,}86+2{,}53+2{,}97+2{,}73) = 1{,}02471.$$

$$T_{ном} = 2{,}45*5{,}4^{1{,}02471} = 13{,}79 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \quad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 13{,}79*(1{,}39 * 0{,}93 * 1 * 1 * 1 * 1{,}11 * 1 * 0{,}87 * 0{,}83 * 1{,}16 * 0{,}92 * 1{,}22 * 1{,}12 * 1{,}22 * 1{,}12 * 1{,}25 * 1{,}29) = 45{,}9 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \quad (7.4)$$

де: *C* – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3*3{,}23*45{,}9^{0{,}33+0{,}2(1{,}02471-1{,}01)}*82 = 284 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	284	Ф 7.1-7.4
Впровадження	13	Д13
Всього	325	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз}N}{F_{pq} - H_{ев}}, \quad (7.5)$$

де: F_{pq} – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{325*1}{60\text{-}5} = 5{,}9 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	385	15	5775	96
Монітор	160	15	2400	40
Клавіатура	140	15	2100	35
Маніпулятор «мишка»	30	15	450	7
Принтер матричний	185	2	370	6
Принтер лазерний	355	1	355	6
Принтер струминний	300	3	900	15
Сканер	155	2	310	5
Концентратор–маршрутизатор	155	3	465	8
Кабельні господарства ЛВС на 1 м. п.	2,5	120	300	5
Кабельне господарство електромережі	48	60	2880	48
Копіювальний пристрій	285	2	570	9
Усього за рік:			$З_ч$	280

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч * n_{mic}}{1,2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{280 * 3}{1,2} = 700 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф_{др}^{с}}{F_{др} * T_{зм}}, \quad (7.7)$$

$$Ч_{ел} = \frac{700}{60 * 8} = 1,458 \text{ ставки.}$$

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, Frame Relay, Wi-Fi, маршрутизатора Cisco, налаштування ADSL	3	1
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	1	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	1	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	3	
Всього		8	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	4	1
	Підтримка постійних клієнтів	2	
	Оформлення договорів, ведення тендерів	1	
	Контроль взаєморозрахунків з постачальниками	1	
Всього		8	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	4	1
	Створення графічних і стилістичних елементів сайту	2	
	Оформлення банерів і промо-сторінок	1	
	Розміщення графіки і контенту на Інтернет сторінках	1	
Всього		8	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	4	1
	Верстка друкованих видань	2	
	Додрукова підготовка макетів	1	
	Розміщення графіки і контенту на Інтернет сторінках	1	
Всього		2	

Продовження таблиці 7.4

	Проектування апаратних засобів обчислювальної техніки і інтелектуальних комп'ютерних систем	3	1
	Підбір, вивчення та узагальнення науково-технічної літератури, нормативних і методичних матеріалів за відповідним обладнанням	3	
	Налагодження, дослідна експлуатація і поетапне введення в дію апаратних засобів обчислювальної техніки і інтелектуальних комп'ютерних систем	2	
Всього		8	
	Ведення бухгалтерського обліку та складання фінансової звітності	2	1
	Оформлення операцій і організацію документообіг	3	
	Проведення інвентаризацій активів	3	
Всього		8	
	Здійснює інсталяцію, настроювання й оптимізацію системного програмного забезпечення й освоєння прикладних програмних засобів.	3	2
	Здійснює підключення і заміну зовнішніх пристроїв, проведення тестування засобів обчислювальної техніки	3	
	Розробляє і впроваджує прикладні програми	2	
Всього		8	

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	12000	36000
Продакт-менеджер	1	8000	24000
Інженер-програміст	6	48000	144000
Інженер-електронщик	1,5	10500	31500
Інженер-системотехнік	1	7000	21000
Адміністратор мережі	1	8000	24000
Системний програміст	2	14000	42000
Дизайнер WEB	1	7000	21000
Інженер-верстальник	1	7000	21000
Бухгалтер-економіст	1	7000	21000
Всього за період розробки	$R_{сп}$ = 16,5	-	$Ф_{роб}$ = 385500

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп}F_{pq}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{385500}{16{,}5 * 60} = 389 \ грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт.

$S_у$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних ТОВ науково-дослідницького консалтингового підприємства «Пекторалъ» (м. Кіровоград) ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 800…1600 у. о./м^2. Враховуючи, що курс складає 1 у.о. = 25 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8$м^2$. З урахуванням цього:

$Б_{уд}$ = 16*8*20000 = 2 560 000 грн.

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 256 000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$I_{нв} = R^1_{сп} * Ц_м, \quad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$I_{нв}$ =16*3500 = 56000 грн.

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу Інтернет-магазину Rozetra за 19.11.21 – джерело https://rozetka.com.ua/

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		21 600
Системний блок	Everest Home 4070 (4070_9414)	17 925
Процесор	Четырехъядерный Intel® Socket 1200 Core™ i3-10100F (3.6 - 4.3 ГГц)	-
Системна плата	ASUS PRIME H410M-R-SI / ASUS PRIME H410M-E	-
Відеокарта	ASUS nVidia GeForce CERBERUS-GTX1050TI-A4G-4G / PH-GTX1050TI-4G	-
Жорсткий диск	3.5" SATA Toshiba / Seagate 1 Tb 7200rpm	-
Оперативна пам’ять	DIMM DDR4 8Gb DDR 2666 Geil / Crucial / Lexar	-
Блок питання	GAMEMAX GM-500B, 500W, 12cm fan	-
Корпус	Logicpower Midle Tower ATX LP 8800	-
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-B, int. 3.5", 1*USB2.0+AUDIO+1394, multi: All Type Cards, black	-
Монітор	Монитор 24" BenQ GL2480 Black (9H.LHXLB.QBE/9H.LHXLB.VBE). Диагональ дисплея 24" Максимальное разрешение дисплея 1920 x 1080 Время реакции матрицы 1 мс (GtG) Яркость дисплея 250 кд/м2 Тип матрицы TN Контрастность дисплея 1000:1 Особенности Flicker-Free	3675
інше	Клавіатура, мишка, система охолодження	Подарунок

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341)	5500
Сканер	Epson Perfection V370 Photo (B11B207313)	6264
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Принтер матричний	Epson FX-2190, А3- Б/У	6050

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість,шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	15	21 600	32400	356400
Принтер лазерний	1	2700	270	2970
Принтер струминний	3	5500	1650	18150
Принтер матричний	2	6050	1210	13310
Копіювальний пристрій	1	5965	596,5	6561,5
Сканер	2	6264	1252,8	13780,8
Всього	–	–	37379	411172

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	2560000	-	-
2. Передавальні пристрої	256000	-	-
Всього по групі	2816000	5	140800
Група 4			
3. Обчислювальна техніка	411172	-	-
Всього по групі	411172	50	205586
Група 5			
4. Вимірювальні пристрої	9031		2257,75
5. Господарський інвентар	56000		14000
Всього по групі 5	65031	25	16257,75
Група 6			
6. Транспортні засоби	143000		28600
Всього по групі 6	143000	20	28600
Нематеріальні активи			
7. Нематеріальні активи	20000	10	2000
Разом	$К_р$= 3455203		$А_р$ = 393244

Примітка: вартість автомобіля Sens (Standard+) взята по даним з автосалону «Кіровоград-Авто», джерело http://kirovograd-avto.ukravto.ua /catalog/tm-9/model-80/description, складає 143000 грн

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} * Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = \frac{389 * 325}{40} = 3161 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о * Н_q * 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 3161*10*0{,}01 = 316 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с$ = 22% від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 * Н_с(З_о + З_д), \quad (7.13)$$

де: $Н_с$ – відрахування на соціальні потреби, %.

$$С_{оц} = 0{,}01*22*(3161 + 316) = 765 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $Н_г$ = 15% від основної зарплати:

$$Г_{осп} = З_о * Н_г * 0{,}01, \quad (7.14)$$

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 3161*15*0{,}01 = 474 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картріджей, тонеру, грн.;

$N_е$ – кількість екземплярів програм, шт.

Згідно виданих викладачем норм приймаємо одну пачку паперу на три місяці розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$ = 105 грн., визначаємо вартість паперу за період розробки $N_м$ = 3 міс:

$$З_{М1} = Ц_п * N_м * n. \quad (7.16)$$

$$З_{М1} = 105*1 = 105 \text{ грн.}$$

Згідно виданих викладачем норм до вартості запам'ятовуючих пристроїв входить вартість CD дисків в кількості, що дорівнює кількості екземплярів програм та одного DVD диска для збереження резервної копії програми:

$$З_{М2} = \sum Ц_д , \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 2 грн./шт., DVD-R LG 4,7Gb, 16x speed Cake box – 3 грн./шт.

$$З_{М2} = 20*2+3 = 43 \text{ грн.}$$

Згідно виданих викладачем норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів для Canon i-SENSYS LBP6030W – 209 грн.; для Epson Stylus Photo P50 – 279 грн.; для Canon i-SENSYS MF217W – 219 грн; для Epson FX-2190, А3- Б/У - 269; відновлення для MF217W – 570 грн.

$$З_{М3} = 209+279+219+269+570 = 1546 \text{ грн.}$$

$$З_М = (105 + 43 + 1546)/20 = 85 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати:

$$О_п = З_о * Н_п * 0{,}01, \quad (7.19)$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 3161*15*0{,}01 = 474 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 40 прим.):

$$A_м = \frac{A_р * N_{міс}}{N_е * 12}, \quad (7.20)$$

де: $A_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$A_м = \frac{393244*3}{(40*12)} = 2458 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$C_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$C_п = 3161 + 316 + 765 + 474 + 85 + 474 + 2458 = 7733 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності ($P_п$) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного продукту рівень рентабельності складає 50%.

$$П_р = 0{,}01 * P_п * C_п, \quad (7.22)$$

де: $P_п$ – рівень рентабельності, %.

$$П_р = 0{,}01*50*7733 = 3866{,}5 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	3161
2. Додаткова зарплата виконавців	$З_д$	316
3. Відрахувння на соціальні потреби	$С_{оц}$	765
4. Загальногосподарські витрати	$Г_{осп}$	474
5. Витрати на матеріали	$З_М$	85
6. Освоєння нових операційних систем, мов програмування	$О_п$	474

Продовження таблиці 7.9

1	2	3
7. Амортизація основних фондів	$А_м$	2458
8. Повна собівартість програмного забезпечення	$С_п$	7733
9. Плановий прибуток	$П_р$	3866,5
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	11599,5
11.Податок на додану вартість $ПДВ = 0.01 * Н_{дв} * Ц_п$	$ПДВ$	2319,9
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	13919,4

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.9.

Таблиця 7.10 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	13919,4
Всього капітальних витрат	–	13919,4

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Рішення зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	26840	10736
2. Витрати на електроенергію	$З_{ел}$	2722	2419
3. Витрати на амортизацію	$З_{ам}$	0	3479,85
Всього витрат за рік	$І$	29562	16634,85

Витрати на обслуговування системи:

$$З_р = Т_р * З_г * (1 + 0{,}01 * Н_q) * (1 + 0{,}01 * Н_c), \qquad (7.23)$$

де: $Т_р$ – кількість годин обслуговування системи за рік, *год.*;

$З_г$ – заробітна плата обслуговуючого персоналу, *грн/год.*

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 200 годин на рік до 80 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\ баз} = 200*100*(1+0.01*10)*(1+0.01*22) = 26840 \text{ грн.}$$

до:

$$З_{р\ нов} = 80*100*(1+0.01*10)* (1+0.01*22) = 10736 \text{ грн.}$$

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел}*Т_р*Ц_{ел}. \qquad (7.24)$$

$$З_{ел\ баз} = 0{,}45*2880*2{,}1 = 2722 \text{ грн.}$$

$$З_{ел\ нов} = 0{,}40*2880*2{,}1 = 2419 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	13919,4	–	3479,85
Всього відрахувань	-	–	13919,4	–	3479,85

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) * N_е - \sum_{i=1}^{m} E_{p_m} * K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$E_в$ = (11599,5-7733)*40-(0,05*2816000 + 0,5*411172+

+0,25*65031+0,2*143000+0,1*20000)*3/12 = 56349 грн.

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_p^*}{(Ц_п - C_п) * N_е} * \frac{n_{міс}}{12}, \quad (7.26)$$

де: K_p^* – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_б = \frac{639203*3}{(11599,5-7733)*40*12} = 1,03324\ року.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (29562\text{-}16634,85)\text{-}0,25*13919,4 = 9447,3 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \quad (7.28)$$

$$T_{сп} = \frac{13919,4}{29562-16634,85} = 1,08\ роки.$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	40
2. Повна собівартість розробленої програми	Грн.	7733
3. Ціна розробленої програми	Грн.	11599,5
4. Плановий прибуток від реалізації розробленої програми	Грн.	3866,5
5. Рентабельність програмної продукції	%	50
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	3455203

Продовження таблиці 7.13

Найменування показників	Одиниця виміру	Величина
7. Загальний прибуток від реалізації програмної продукції	Грн.	154660
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	56349
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	1,03324
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	13919,4
11.Величина економічного ефекту у користувача програмної продукції	Грн.	9447,3
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	1,08

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості управлінських рішень.

8 ЗАХОДИ ЩОДО ОХОРОНИ ПРАЦІ І ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

«З давніх давен людство приділяє прискіпливу увагу безпеці життя і охорони праці як її складової частини. Умови праці розглядали Арістотель (384-322 до н.е.) та Гіппократ (460-377 до н.е.)» [62].

Законом України “Про охорону праці” [61] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [63], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби пальців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робітників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

При розгляді шкідливих чинників роботи програмістів та інших спеціалістів ІТ будемо керуватись наступними нормативно-правовими актами:

«Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [63], та «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста включають наступні фактори:

- параметри повітряного середовища в приміщенні;
- вентиляція приміщення;
- освітлення приміщення;
- параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення впливу комп'ютера на організм програміста визначаємо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Програміст працює з електронно-обчислювальною машиною (ЕОМ) та іншим обладнанням, яке є джерелом небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. Так як програміст постійно перебуває в приміщенні, тому для комфортних умов праці в цьому приміщенні необхідно створити належний мікроклімат.

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори, як:

- ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.
- ризик виникнення пожежі;
- негативний вплив на органи зору людини;
- ризики ураження електричним струмом;
- недостатня, або надмірна освітленість робочого місця;

- монотонність праці;

- електромагнітні (у т.ч. високочастотні) електромагнітні випромінювання (коливання);

- несприятливі мікрокліматичні умови;

- нервово-емоційна напруженість праці;

- інтелектуальні навантаження;

- невідповідність ергономічних показників робочого місця діючим вимогам;

- шуми;

- статичні навантаження на кістково-м'язовий апарат;

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	5,4
Довжина	6
Висота	2,75

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого*

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	8,1
Обсяг, V	$м^3$	не менше 20.0	24,3

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

У зазначеному приміщенні працюють 4 людей. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об'єм приміщення у розрахунку на одно робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [63], але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [63] та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»). Таним чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості

руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату.

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Вологість,%	Швидкість повітря, м/с	Температура, °С	Вологість %	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-23	40-55	0,1
Тепла	23-25	50-70	0,1	24-25	50-65	0,11

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року, а в літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер HP 1100, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

З 2019 року діють Державні будівельні норми України “Природне і штучне освітлення” – ДБН В.2.5-28:2018 [59], у яких прописані вимоги до використання всіх освітлювальних приладів, у т.ч. світлодіодних.

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5-28:2018 [59], можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк. [59], Крім того все поле зору повинне бути освітлено достатньо рівномірно - ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення, яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.4 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

- розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;
- мікроклімат відповідає нормативному значенню;
- акустичні умови роботи не перевищують нормативних значеннь;

Таким чином можна припустити, що основною причиною можливого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга).

Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв, які працюють при напрузі вище 36 В.

Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

8.5 Розрахункова частина

Для захисного штучного заземлення застосовуються вертикальні електроди: металевий куток 50*50*5 мм. (*К=50 мм.=0,04 м.)* довжиною *L=3 м.,* та горизонтальний електрод — металева полоса з перетином 40*4 мм. Напруга — 220/380 В. Розрахункова схема розташування заземлюючих електродів.

Розрахунок проводиться за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару ґрунту — чорнозем, нижнього шару ґрунту — глина (питомий опір 40 Ом*м). Умовна товщина верхнього шару ґрунту: *Н=0,4 м*. Відстань між вертикальними заземлювачами (електродами) *А*=3 *м*. Глибина закладення горизонтального контуру заземлення *t= 0,7 м*. Опір заземлювача, який

нормується: $R_{ЗН} = 4$ *Ом*. Необхідно визначити необхідну кількість вертикальних заземлювачів та довжину полоси (горизонтального заземлювача).

Розрахунок захисного заземлення можна автоматизувати за допомогою програми, сирцевий код якої опублікований на 13-16 сторінках[64], або аналогічної.

Розрахунок.

Відстань від центра вертикального заземлювача до поверхні землі:

$T=t+L/2=0,7+3/2=2,2$ *м.*

Розрахунковий питомий опір ґрунту (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі ґрунту):

$\rho = \psi \rho_2 = 1,2*40=54,5$ *Ом*м.*

де $\psi = 1,2$ - табличне значення коефіцієнта сезонності для відповідної кліматичної зони у багатошаровому ґрунту [64];

$\rho_1 = 50$ *Ом*м.* - значення питомого опору верхнього шару ґрунту [66];

$\rho_2 = 40$ *Ом*м*. - значення питомого опору нижнього шару ґрунту [66].

Еквівалентний діаметр вертикального електрода (кутка) [66]:

$$Dв=0,95*K= 0,95*50 = 47,5 \text{ мм.} = 0,0475 \text{м.}$$

Опір розтіканню електричного струму одного електрода вертикального заземлювача (з врахуванням заглиблення заземлювача) [66]:

$$R_O = 0,366(\rho/L)\lg(4L/Dв)=0,366(54,5/3)\lg(4*3/0,0475)=16 \text{ Ом.}$$

Відношення $A/L=3/3=1,3$.

Визначаємо коефіцієнт екранування вертикальних електродів $Кев=0,79$ при орієнтовній кількості вертикальних електродів, яке дорівнює 4 [66].

Визначаємо необхідну кількість вертикальних заземлювачів (без врахування горизонтального заземлювача), при $R_{ЗН} = 4$ *Ом* :

$$N=R_O /(Кев\ R_{ЗН})= 16 / (0,79*4)=5,66 \approx 5 \text{ шт.}$$

Визначаємо довжину з'єднуючої полоси:

$$L_П = 1,05*A*N= 1,05*3*5=15,8 \approx 16 \text{ м.}$$

Опір розтіканню електричного струму з'єднуючої полоси [66]:

$$R_П = 0,366(\rho_2 * K_П / L_П) lg(2(L_П * L_П)/(К * t)) =$$

$$= 0,366(40*5/16) * [lg(2*16*16)/(0,04*0,7)] = 19,88 \text{ Ом}.$$

де $K_П = 5$ - табличне значення коефіцієнта сезонності для відповідної кліматичної зони з'єднуючої полоси: [66].

Загальний опір розтіканню електричного струму заземлювача [66]:

$$R = (R_О * R_П)/(R_О * \eta_П + N * R_П * Кев) =$$

$$= (5,62 * 12,88)/(5,62 * 0,75 + 1 * 12,88 * 0,79) = 3,5 \text{ Ом}.$$

де $\eta_П = 0,75$ - табличне значення коефіцієнта екранування з'єднуючої полоси [66].

Умова $R \leq R_{ЗН}$ виконується ($3,5 \leq 4$).

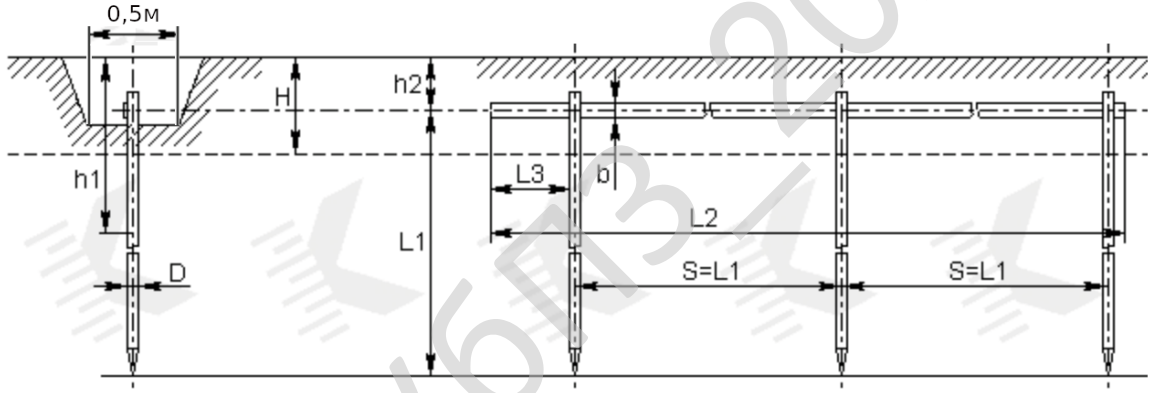

Рисунок 8.1 – Схема штучного заземлення.

8.6 Висновки

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Кваліфікаційна магістерська робота надала мені можливість здійснити своє бажання з приводу розробки власної покрокової стратегічної гри. Під час розробки програмного забезпечення, з використанням міжплатформного середовища Unity, що візуалізує гру, були розроблені концепція і методи, які будують її втілення. Після чого, для якісного представлення роботи системи, була здійснена розробка архітектури та побудовані схеми.

Під час виконання роботи, мною була розроблена низька управлінських менеджерів, що продемонстрували якісне управління ресурсами. Завдяки ним, теперішня побудова карти проходить майже автоматично, потрібно лише вказати низьку основних характеристик.

Низькорівневий штучний інтелект подарував мені низьку позитивних емоцій, особливо тоді, коли я зм іг втілити декілька модулів поведінки, хоча вони і відрізняються лише до того моменту, коли зустрінуть свого супротивника з ворожої фракції. Штучний інтелект, що був розроблений, має такі режими як агресивний, пасивний та тригерний. Гексагональна карта, що була побудована, завдяки менеджеру з генерації карти, зараз може генеруватися, а після і змінюватися. Дякуючи функціональності цього менеджеру, а не з використання методу формування пустих частин карти реалізована можливість прицільно знищувати гекси.

В подальшому, я планую продовжити втілювати в життя програмне забезпечення по візуалізації власної покрокової стратегічної гри, щоб розробити повноцінну гру, в якій буде відтворюватися життя кожного юніта, з прописуванням усіх їх можливих характеристик, навиків та талантів.

Під час виконання кваліфікаційної магістерської роботи я навчився більш якісно використовувати доступні ресурси та опановувати навички розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Multimedia . Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Multimedia
2. Донован, Тристан (2014). Играй! История видеоигр / Тристан Донован; пер. И. Воронина. Москва: Белое Яблоко. с. 131–132.
3. Комп'ютерна графіка : конспект лекцій для студентів усіх форм навчання спеціальностей 122 «Комп'ютерні науки» та 123 «Комп'ютерна інженерія» з курсу «Комп'ютерна графіка» / Укладач: Скиба О.П. – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2019. – 88 с
4. Маценко В.Г. Комп'ютерна графіка: Навчальний посібник. – Чернівці: Рута, 2009 – 343 с. ISBN 966-568-846-4
5. Новожилова М.В., Мироненко В.В. Комп'ютерна графіка. Частина1: Навчально-методичний посібник. – Х.: ХНУБА, 2015.– 60 с.
6. Конспект лекцій з навчальної дисципліни «Комп'ютерна графіка» спеціальності 051 «Економіка» / Укладач Стадник Ю.А., доцент., к.е.н., доцент Львівського національного університету імені Івана Франка факультету управління фінансами та бізнесу, затверджено на засіданні кафедри цифрової економіки та бізнес-аналітики протокол № 6 від "21" січня 2020 р. 5. Білуха М. Т. Методологія наукових досліджень: Підручник. — К.: АБУ, 2002. — 480 с.
7. M. Ehrgott and X. Gandibleux. Approximative Solution Methods for Multiobjective Combinatorial Optimization // TOP. — Sociedad de Estadística e Investigación Operativa, 2004. — Т. 12, вип. 1.
8. Michael Klappenbach Understanding and Optimizing Video Game Frame Rates // December 16, 2020 Режим доступу до ресурсу: https://www.lifewire.com/optimizing-video-game-frame-rates-811784

9. Heroes Might of Magic 3 Режим доступу до ресурсу: https://www.ubisoft.com/ru-ru/game/heroes-of-might-and-magic-3-hd

10. Крис Фадина Большое исследование про свет в играх // 10 августа 2020. Режим доступу до ресурсу https://cyber.sports.ru/tribuna/blogs/betterthansex/2812994.html

11. Улутшенная графика с трассировкой лучей и DLSS Режим доступу до ресурсу:: https://www.nvidia.com/ru-ru/geforce/rtx/

12. Владислав Миронович Что такое трассировка лучей и нужна ли она нам в играх // 01.05.2020 Режим доступу до ресурсу: https://blog.allo.ua/chto-takoe-trassirovka-luchej-i-nuzhna-li-ona-nam-v-igrah_2020-05-51/

13. Unity Toon Shader мультяшный шейдер (1): мультипликационная раскраска // Режим доступу до ресурсу: https://russianblogs.com/article/4312217817/

14. Toony Colors Pro 2. Режим доступу до ресурсу: https://assetstore.unity.com/packages/vfx/shaders/toony-colors-pro-2-8105

15. AutoCAD overview. Режим доступу до ресурсу: https://www.autodesk.ru/products/autocad/overview

16. Make something Unreal. Режим доступу до ресурсу: https://www.unrealengine.com/en-US/unreal

17. Встречайте Unity Gaming Services. Режим доступу до ресурсу: https://unity.com/ru

18. The Freedom to Create. Режим доступу до ресурсу: https://www.blender.org/about/

19. GODOT showcase. Режим доступу до ресурсу https://godotengine.org/showcase

20. C# and Shader Tutorials for the Unity Engine. Режим доступу до ресурсу: https://catlikecoding.com/unity/tutorials/

21. Turn-based RPG with Unity. Режим доступу до ресурсу: https://tbswithunity3d.wordpress.com/

22. Карти з шестикутників в Unity: частини 1-3. Режим доступу до ресурсу: https://habr.com/ru/post/424257/

23. Hexagonal Grids. Режим доступу до ресурсу: https://www.redblobgames.com/grids/hexagons/

24. Створення сіток шестикутників. Режим доступу до ресурсу: https://habr.com/ru/post/319644/

25. Про прямокутних координатах і гексагональних сітках. Режим доступу до ресурсу: https://habr.com/ru/post/147082/

26. Пошук шляху на гексагональної сітці. Режим доступу до ресурсу: https://habr.com/ru/post/125751/

27. Player.IO. Приклад покрокової гри. Режим доступу до ресурсу: http://www.ant-karlov.ru/PlayerIO-primer-poshagovoy-igri.html

28. Хокинг, Джозеф. Unity — в действии. Мультиплатформенная разработка на C# : [рус.]. — 2. — СПб : Питер, 2016. — 336 с. — ISBN 978-1617292323.

29. Торн, Алан. Искусство создания сценариев в Unity : [рус.]. - СПб : ДМК, 2016. - 362 с.

30. Джон Скит. C# для профессионалов: тонкости программирования, 3-е издание, новый перевод = C# in Depth, 3rd ed.. — М.: «Вильямс», 2014. — 608 с. — ISBN 978-5-8459-1909-0.

31. Кристиан Нейгел и др. C# 5.0 и платформа .NET 4.5 для профессионалов = Professional C# 5.0 and .NET 4.5. — М.: «Диалектика», 2013. — 1440 с. — ISBN 978-5-8459-1850-5.

32. А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд. Язык программирования C#. Классика Computers Science. 4-е издание = C# Programming Language (Covering C# 4.0), 4th Ed. — СПб.: «Питер», 2012. — 784 с. — ISBN 978-5-459-00283-6. Архивная копия от 10 октября 2011 на Wayback Machine

33. Э. Стиллмен, Дж. Грин. Изучаем C#. 2-е издание = Head First C#, 2ed. — СПб.: «Питер», 2012. — 704 с. — ISBN 978-5-4461-0105-4. (недоступная ссылка)

34. Эндрю Троелсен. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание = Pro C# 5.0 and the .NET 4.5 Framework, 6th edition. — М.: «Вильямс», 2013. — 1312 с. — ISBN 978-5-8459-1814-7.

35. Джозеф Албахари, Бен Албахари. C# 6.0. Справочник. Полное описание языка = C# 6.0 in a Nutshell: The Definitive Reference. — М.: «Вильямс», 2018. — 1040 с. — ISBN 978-5-8459-2087-4. — ISBN 978-1-491-92706-

36. Хокинг, Джозеф. Unity — в действии. Мультиплатформенная разработка на C# : [рус.]. — 2. — СПб : Питер, 2016. — 336 с. — ISBN 978-1617292323.

37. Grund- und Sicherheitsregeln für die Mensch-Maschine-Schnittstelle — Codierungsgrundsätze für Anzeigengeräte und Bedienteile. Europäische Norm 60073:2002, deutsche Fassung. VDE Verlag, Berlin 2003.

38. User Interface. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/User_interface

39. Luke Ahearn. 3D game environments: create professional 3D game worlds – Boca Raton, FL: CRC Press, Taylor & Francis Group, [2017]. – 331 p. – ISBN 9781138920026

40. Noor Shaker, Julian Togelius, Mark J. Nelson. Procedural Content Generation in Games. Computational Synthesis and Creative Sys-tems – Switzerland: Springer International Publishing, 2016. – 237 p. – ISBN 978-3-319-42714-0

41. Oliver Korn, Newton Lee. Game Dynamic. Best Practices in Procedural and Dynamic Game Content Generation. – AG: Springer Inter-national Publishing, 2017. 177 p. – ISBN 978-3-319-53087-1

42. Richard A. Hawley. Grome Terrain Modeling with Ogre3D, UDK, and Unity3D. – UK: Packt Publishing Ltd, 2013. 162 p. – ISBN 978-1-84969-939-6

43. Ryan Watkins. Procedural Content Generation for Unity Game Development. – UK: Packt Publishing Ltd, 2016. 260 p. – ISBN 978-1-78528-747-3

44. Tanya X. Short, Tarn Adams. Procedural generation in game design. – Boca Raton: Taylor & Francis, CRC Press, 2017. – 320 p. – ISBN 9781138743311

45. Могилев, А.В. Информатика. / Н.И. Пак, Е.К. Хённер – М.:Академия, 2004. – 848 с.

46. Шеллинг, Т. Стратегия конфликта. / М.: ИРИСЭН, 2007. – 376 с.

47. Диксит, А. Теория игр: искусство стратегического мышления в бизнесе и жизни/ Б. Нейлбафф, – М.: Манн, Иванов и Фербер, 2014. – 464 с.

48. Мураховский, В.И. Компьютерная графика – М.: АСТ-ПРЕСС 2002 – 638 с.

49. Корабельникова, Г.Б. Adobe Photoshop в теории и на практике. / Ю.А.Гурский – М.: Новое Знание, 2002. – 528 с.

50. Макарова, Н. В. Информатика / Л.А. Матвеев, В.Л. Бройдо – М.: Финансы и Статистика, 2002. – 768 с.

51. Пауэль, Т.А. Справочник программиста. / Т.А. Пауэль, Д. Уитворт – М.: АСТ, 2005. – 470 с.

52. Симонович, С.В. Практическая информатика. / Г.А. Евсеев – М.: АСТ-ПРЕСС КНИГА, 2005. – 480 с.

53. Морелос-Сарагоса, Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение – М.:ТЕХНОСФЕРА, 2006.– 320 с.

54. Блэкманн, С. Моделирование с помощью Unity – М.: АСТ- ПРЕСС, 2011. – 147 с.

55. Голдстон, В. Предметы первой необходимости для создания игр на Unity – М.: Академия, 2009 – 201 с.

56. Игошев, Б. ЭВТ: знакомимся, делаем, играем. / М. Галагузова, Д. Комский – М.:Молодая гвардия, 1989. – 156 с.

57. Лысовский, А. Как сибирские программисты заставили весь мир играть. / А.Егошин, А.Бочкарев - М.:Русский Репортер №10, 2012. - 56- 61с.

58. Мильчин, К. Школа доктора Пакмэна. Чему нас научили компьютерные игры - М.: "Русский Репортер" №38 2013 - 32с.

59. Державні будівельні норми України: ДБН В.2.5-28:2018. - Режим доступу до ресурсу: https://goo.su/9AkQ

60. . Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

61. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

62. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

63. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508

64. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. - Кіровоград: КІСМ, 1997. - 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358

65. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99

66. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : “Колос”, 1973. – 238 с.

67. Центр післядипломної освіти та підвищення кваліфікації. - Режим доступу до ресурсу: https://cpo.stu.cn.ua

Додаток А

Технічне завдання

Зміст

					ВКРМ -123.21.0004.00.00.ТЗ			
Вим	Арк	№ докум.	Підпис	Дата				
Розроб.		Железняк Б.Ю.			Дослідження та програмна реалізація системи візуалізації покрокової стратегічної гри	Літ.	Аркуш 1	Аркушів 7
Перев.		Дрєєв О.М.				Б		
Н.контр.		Гермак В.С.				ЦНТУ КІ-20 (М)		
Затв.		Смірнов О.А.						

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку компоненту програмного комплексу “Програмне забезпечення системи візуалізації покрокової стратегічної гри”.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (наказ № ____ від _____ 20__р.).

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є збір теоретичних відомостей про сучасні комп’ютерні ігри, жанри ігор, зосереджуючи увагу на стратегічних іграх та, за допомогою аналізу популярних ігор, розроблення архітектури своєї гри, концентруючись на бойовій частині, на основі впровадження нових інформаційних технологій і застосування сучасних засобів програмування.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є розробки, які ведуться спільнотою розробників комп’ютерних ігор, кафедрою кібербезпеки та програмного забезпечення і стосовні до теми бібліографічні джерела.

5 Технічні вимоги

5.1 Вміст проекту

Складовими розробки є:

– аналіз існуючих популярних аналогів покрокових стратегічних ігор;

– вибір і обґрунтування методики побудови додатків і засобів їхньої реалізації;

– генерація гексагонної глобальної карта, юнітів та системи фракцій, розробка механізму їхньої взаємодії, робочих форм і засобів;

– розробка програми, яка реалізує алгоритми роботи компоненту.

5.2 Показники призначення

Система повинна забезпечувати:

– систему взаємодії скриптів, моделей і параметрів, які перебувають на ігровій сцені;

– простий, інтуїтивно зрозумілий інтерфейс з гравцем;

– можливість тестування та подальшого розвитку.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно містити обмежень на адаптування та подальшого розробку.

5.4 Вимоги до архітектури

Компонент, що розробляється, повинен підтримувати роботу з системними компонентами ОС Windows.

5.5 Вимоги до надійності

Компонент повинен використати існуючі угоди по стандартним викликам процедур, функцій, засобів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Автоматизовані робочі місця користувачів системи повинні задовольняти наступним умовам експлуатації:

- температура повітря: 18-22^0 С;
- відносна вологість повітря при 20^0 С до 80%;
- атмосферний тиск 107 кПа.

5.7 Вимоги до складу і параметрів технічних засобів

Компонент повинен бути реалізований на ЕОМ типу IBM PC в операційному середовищі Windows 10 і орієнтований на сумісні з цією платформою зовнішні пристрої, мережне обладнання і прикладне програмне забезпечення.

5.8 Вимоги до інформаційної та програмної сумісності

Сумісність програмного забезпечення повинна бути забезпечена за рахунок його реалізації засобами Unity, міжплатформного середовища розробки комп'ютерних ігор, працюючого під управлінням ОС Windows 10.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

C# з використанням можливостей міжплатформного середовища розробки комп'ютерних ігор Unity.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена в вигляді опису структури даних, схем і описів алгоритмів, інструкції користувача, а також текстів вхідних модулів програмного забезпечення в відповідності з ЄСПД.

7 Економічні вимоги

7.1 Для програми ядра виконавчої системи моделі необхідно виробити функціонально-вартісний аналіз варіантів розробки

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 03. 12 2021 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд

небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з охорони праці.

9 Перелік документів, які необхідно розробити

- Структурна схема системи повна – 1 аркуш.
- Наукова новизка – 1 аркуш
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схеми алгоритму роботи програми – 1 аркуша.
- Блок схема алгоритму роботи підпрограм – 3 аркуша
- Пояснювальна записка – 114 аркуша.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

10.3 Розробка функціональних схем, блок-схем алгоритмів роботи програмного забезпечення компоненту.

10.4 Побудова схем взаємодії структур даних.

10.5 Створення прототипу компоненту. Створення програмного продукту.

10.6 Відлагодження компоненту, аналіз отриманих результатів.

10.7 Робота над питаннями охорони праці і техніки безпеки.

10.8 Розрахунки по техніко-економічному обґрунтуванню (остаточні).

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю і приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 03.12. 2021 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 22.12.2021 р.

Додаток Б

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за другим (магістерським) рівнем вищої освіти

_____ Дрєєв О. М. .

Дослідження та програмна реалізація системи візуалізації покрокової стратегічної гри

Лістинг програми

Код документу 12

Носій: DVD-диск

Загальна кількість аркушів: 61

Літера: РП

Кропивницький – 2021 року

Лістинг AbilityManager.cs

```
using UnityEngine;using System.Collections;using System.Collections.Generic;
using TBTK;
namespace TBTK{

	public class AbilityManager : MonoBehaviour {
		private static AbilityManager instance;
		public void Awake(){
			instance=this;}
		void Update(){
			if(!targetMode) return;
			if(Input.touchCount>0){
				if(Input.touchCount>=2) ExitAbilityTargetMode();}
				else{if(Input.GetMouseButtonDown(1))
			ExitAbilityTargetMode();}}
		private static bool targetMode=false;
		public static bool InTargetMode(){ return targetMode; }
		public delegate void TargetModeCallBack(Tile tile, int abIndex);
		private TargetModeCallBack targetSelectedCallBack;
		public delegate void ExitTargetModeCallBack();
		private ExitTargetModeCallBack exitTargetCallBack;
		private int abilityIndex=0;
		private int targetModeAOE;
		private _TargetType targetModeType;
		private List<Tile> targetModeTileList=new List<Tile>();
		private List<Tile> targetModeHoveredTileList=new List<Tile>();
		public void _ActivateTargetMode(int abIndex, int AOE, _TargetType
type, TargetModeCallBack sCallBack, ExitTargetModeCallBack eCallBack){
		GridManager.AbilityTargetSelectMode(this.AbilityTargetSelected,
this.SetTargetModeHoveredTile, this.ClearTargetModeHoveredTile);
			targetMode=true;
			abilityIndex=abIndex;
			targetModeAOE=AOE;
			targetModeType=type;
			targetSelectedCallBack=sCallBack;
			exitTargetCallBack=eCallBack;
			OverlayManager.DisableTileCursor();
			TBTK.OnAbilityTargetMode(true);}
		public void ActivateTargetModeFaction(int AOE, _TargetType type,
int abIndex, TargetModeCallBack sCallBack, ExitTargetModeCallBack eCallBack){
			TBTK.OnFactionABTargetMode(abIndex);
			_ActivateTargetMode(abIndex, AOE, type, sCallBack,
		eCallBack); }
			public void ActivateTargetModeUnit(Tile tile, UnitAbility
ability, int abIndex, TargetModeCallBack sCallBack, ExitTargetModeCallBack
eCallBack){
			TBTK.OnUnitABTargetMode(abIndex);
			_ActivateTargetMode(abIndex, ability.GetAOERange(),
ability.targetType, sCallBack, eCallBack);
			if(!ability.AttackInLine()){
				if(!ability.normalAttack){
					if(targetModeType==_TargetType.EmptyTile)
		targetModeTileList=GridManager.GetTilesWithinDistance(tile,
ability.GetRange(), true);
					else
		targetModeTileList=GridManager.GetTilesWithinDistance(tile,
ability.GetRange());}
				else{
					targetModeTileList=new List<Tile>();
					List<Tile>
tilesInRangeList=GridManager.GetTilesWithinDistance(tile,
ability.GetRange());
```

```
                        int sight=tile.unit.GetSight();
                              List<Unit>
allFriendlyUnitList=FactionManager.GetAllUnitsOfFaction(tile.unit.factionID);
                              for(int i=0; i<tilesInRangeList.Count; i++){
                                    Tile targetTile=tilesInRangeList[i];
                              if(!GameControl.EnableFogOfWar() &&
                        !GameControl.AttackThroughObstacle()){
                                          if(!FogOfWar.InLOS(tile, targetTile,
                                    0)) continue;}
                                    bool inSight=GameControl.EnableFogOfWar()
                                    ? false : true;
                                    if(GameControl.EnableFogOfWar()){
                              if(FogOfWar.InLOS(tile, targetTile) &&
                        GridManager.GetDistance(tile, targetTile)<=sight){
                                                inSight=true;}
                                          else if(!ability.requireDirectLOS){
                                                for(int n=0;
                                          n<allFriendlyUnitList.Count; n++){
      if(allFriendlyUnitList[n]==tile.unit) continue;
            if(GridManager.GetDistance(allFriendlyUnitList[n].tile,
targetTile)>allFriendlyUnitList[n].GetSight()) continue;
            if(FogOfWar.InLOS(allFriendlyUnitList[n].tile, targetTile)){
                              inSight=true;
                              break; }}}}
                                    if(inSight)
            targetModeTileList.Add(targetTile);}}}
                  else{
      OverlayManager.ShowAbilityRangeIndicator(targetModeTileList);
            }
            public static void ExitAbilityTargetMode(){
                  if(!targetMode) return;
                  TBTK.OnFactionABTargetMode();
                  TBTK.OnUnitABTargetMode();          //clear ability select
mode for UI
                  targetMode=false;
                  instance.abilityIndex=0;
                  instance.targetModeTileList=new List<Tile>();
                  OverlayManager.EnableTileCursor();
                  OverlayManager.ClearAbilityTargetIndicator();
                  OverlayManager.ClearAbilityRangeIndicator();
                  GridManager.ClearTargetSelectMode();
                  GameControl.ReselectUnit()
                  if(instance.exitTargetCallBack!=null)
instance.exitTargetCallBack();
                  TBTK.OnAbilityTargetMode(false);}
                  public void AbilityTargetSelected(Tile tile){
                  if(targetModeTileList.Count>0 &&
!targetModeTileList.Contains(tile)){
                        GameControl.DisplayMessage("Out of Range");
                        return;}
                  bool invalidFlag=false;
                  if(targetModeType==_TargetType.AllUnit){
                        invalidFlag=(tile.unit==null);}
                  else if(targetModeType==_TargetType.HostileUnit){
                  int currentFacID=FactionManager.GetSelectedFactionID();
                        invalidFlag=(tile.unit==null ||
tile.unit.factionID==currentFacID);}
                  else if(targetModeType==_TargetType.FriendlyUnit){
                  int currentFacID=FactionManager.GetSelectedFactionID();
                  invalidFlag=(tile.unit==null ||
tile.unit.factionID!=currentFacID); }
                  else if(targetModeType==_TargetType.EmptyTile){
                  invalidFlag=(tile.unit!=null || !tile.walkable);}
```

```
                else if(targetModeType==_TargetType.AllTile){ }
            if(invalidFlag) GameControl.DisplayMessage("Invalid target");
                else{
                    if(targetSelectedCallBack!=null)
targetSelectedCallBack(tile, abilityIndex);
                    TBTK.OnAbilityActivated();
                    ExitAbilityTargetMode();}}
            private void SetTargetModeHoveredTile(Tile tile){
                ClearTargetModeHoveredTile();
                if(targetModeTileList.Count>0 &&
!targetModeTileList.Contains(tile)) return;
                targetModeHoveredTileList=new List<Tile>();
                if(targetModeAOE>0)
targetModeHoveredTileList=GridManager.GetTilesWithinDistance(tile,
targetModeAOE);
                if(!targetModeHoveredTileList.Contains(tile))
targetModeHoveredTileList.Add(tile);
      OverlayManager.ShowAbilityTargetIndicator(targetModeHoveredTileList);}
            private void ClearTargetModeHoveredTile(Tile tile=null){
                OverlayManager.ClearAbilityTargetIndicator();}
            public static bool CanCastAbility(Tile tile){ return
instance.targetModeTileList.Contains(tile); }
            public static void ApplyAbilityEffect(Tile targetTile, Ability
ability, int type, Unit srcUnit=null){
      instance.StartCoroutine(instance._ApplyAbilityEffect(targetTile,
ability, type, srcUnit));}
            IEnumerator _ApplyAbilityEffect(Tile targetTile, Ability ability,
int type, Unit srcUnit=null){
                if(targetTile!=null && ability.effectObject!=null){
                    if(!ability.autoDestroyEffect)
ObjectPoolManager.Spawn(ability.effectObject, targetTile.GetPos(),
Quaternion.identity);
                    else ObjectPoolManager.Spawn(ability.effectObject,
targetTile.GetPos(), Quaternion.identity, ability.effectObjectDuration);}
                if(!ability.useDefaultEffect) yield break;
                if(ability.effectDelayDuration>0) yield return new
WaitForSeconds(ability.effectDelayDuration);
                if(type==1){
                    List<Tile> tileList=new List<Tile>();
                    if(targetTile!=null){
                        if(ability.aoeRange>0)
tileList=GridManager.GetTilesWithinDistance(targetTile, ability.aoeRange);
                        tileList.Add(targetTile);}
                    else{
                    if(ability.effTargetType==_EffectTargetType.Tile)
      tileList=GridManager.GetTileList();
                        else{
                        List<Unit> unitList=FactionManager.GetAllUnit();
                    for(int i=0; i<unitList.Count; i++)
                tileList.Add(unitList[i].tile);}}
                    if(ability.effTargetType==_EffectTargetType.AllUnit){
                        for(int i=0; i<tileList.Count; i++){
                            if(tileList[i].unit==null) continue;
      tileList[i].unit.ApplyEffect(ability.CloneEffect());
                            SpawnEffectObjTarget(ability,
tileList[i]);}}
                    else
if(ability.effTargetType==_EffectTargetType.HostileUnit){
                        for(int i=0; i<tileList.Count; i++){
                            if(tileList[i].unit==null) continue;
            if(tileList[i].unit.factionID==ability.factionID) continue;
                tileList[i].unit.ApplyEffect(ability.CloneEffect());
```

```
                                    SpawnEffectObjTarget(ability,
tileList[i]);}}
                  else
if(ability.effTargetType==_EffectTargetType.FriendlyUnit){
                              for(int i=0; i<tileList.Count; i++){
                                    if(tileList[i].unit==null) continue;
      if(tileList[i].unit.factionID!=ability.factionID) continue;
      tileList[i].unit.ApplyEffect(ability.CloneEffect());
                              SpawnEffectObjTarget(ability, tileList[i]);}}
                        else
if(ability.effTargetType==_EffectTargetType.Tile){
                              for(int i=0; i<tileList.Count; i++){
      tileList[i].ApplyEffect(ability.CloneEffect());
                        SpawnEffectObjTarget(ability, tileList[i]);}}}
                  else if(type==2){ //_AbilityType.SpawnNew
            Quaternion rot=srcUnit!=null ? srcUnit.thisT.rotation :
Quaternion.identity ;
                  GameObject
unitObj=(GameObject)Instantiate(ability.spawnUnit.gameObject,
targetTile.GetPos(), rot);
                  Unit unit=unitObj.GetComponent<Unit>();
                        unitObj.transform.position=targetTile.GetPos();
                        if(srcUnit!=null)
unitObj.transform.rotation=srcUnit.thisT.rotation;
                        else{
            Faction faction=FactionManager.GetFaction(ability.factionID);
                              if(faction!=null)
unitObj.transform.rotation=Quaternion.Euler(0, faction.spawnDirection, 0);}
                        FactionManager.InsertUnit(unit, ability.factionID);
                        unit.SetNewTile(targetTile);
                        if(GridManager.GetDistance(targetTile,
srcUnit.tile)<=srcUnit.GetMoveRange()) GameControl.SelectUnit(srcUnit);      }
                  else if(type==3){ //_AbilityType.ScanFogOfWar)
                        List<Tile>
targetTileList=GridManager.GetTilesWithinDistance(targetTile,
ability.GetAOERange());
                        targetTileList.Add(targetTile);
                        for(int i=0; i<targetTileList.Count; i++)
targetTileList[i].ForceVisible(ability.effect.duration);}
                  else if(type==4){ //_AbilityType.Overwatch
                        targetTile.unit.Overwatch(ability.CloneEffect());}
                  else if(type==5){ //_AbilityType.Teleport
                        Quaternion
wantedRot=Quaternion.LookRotation(targetTile.GetPos()-
ability.unit.tile.GetPos());
                        ability.unit.thisT.rotation=wantedRot;
                        GameControl.ClearSelectedUnit();
                        ability.unit.SetNewTile(targetTile);
                        GameControl.SelectUnit(srcUnit);}
                  else if(type==6){ //charge attack
                        List<Tile>
tileList=GridManager.GetTilesInALine(srcUnit.tile, targetTile,
ability.GetRange());
                        for(int i=0; i<tileList.Count; i++){
                              while(true){
                                    float
dist=Vector3.Distance(srcUnit.thisT.position, tileList[i].GetPos());
                                    if(dist<0.05f){
                                          if(tileList[i].unit!=null){
      tileList[i].unit.ApplyEffect(ability.CloneEffect());
                              SpawnEffectObjTarget(ability, tileList[i]);
                              break;}
```

```
            Quaternion wantedRot=Quaternion.LookRotation(tileList[i].GetPos()-
srcUnit.thisT.position);
            srcUnit.thisT.rotation=Quaternion.Slerp(srcUnit.thisT.rotation,
wantedRot, Time.deltaTime*srcUnit.moveSpeed*6);
      Vector3 dir=(tileList[i].GetPos()-srcUnit.thisT.position).normalized;
      srcUnit.thisT.Translate(dir*Mathf.Min(srcUnit.moveSpeed*2*Time.deltaTime
, dist), Space.World);
                                    yield return null;}}
                        srcUnit.tile.unit=null;
                        srcUnit.tile=tileList[tileList.Count-1];
                        tileList[tileList.Count-1].unit=srcUnit;
                        GameControl.ReselectUnit();}
                  else if(type==7){ //attack all unit in a straight line
                        List<Tile>
tileList=GridManager.GetTilesInALine(srcUnit.tile, targetTile,
ability.GetRange());
                        for(int i=0; i<tileList.Count; i++){
                              if(tileList[i].unit==null) continue;
      tileList[i].unit.ApplyEffect(ability.CloneEffect());
                              SpawnEffectObjTarget(ability, tileList[i]);}}
                  else{
                        Debug.LogWarning("Error, unknown ability type
("+type+")");}
                  yield return null;
            void SpawnEffectObjTarget(Ability ability, Tile tile){
                  if(ability.effectObjectTarget!=null){
                        if(!ability.autoDestroyEffectTgt)
ObjectPoolManager.Spawn(ability.effectObjectTarget, tile.GetPos(),
Quaternion.identity);
                        else
ObjectPoolManager.Spawn(ability.effectObjectTarget, tile.GetPos(),
Quaternion.identity, ability.effectObjectTgtDuration)}}}
```

Лістинг Tile.cs

```
using UnityEngine;using System.Collections;using System.Collections.Generic;
using TBTK;
namespace TBTK{

      public enum _TileType{Hex, Square}
      public enum _TextureTypeTile {Forest, Pasture}

      [System.Serializable]
      public class Wall{
            public bool init=false;
            public Tile neighbour;
            public Transform wallObjT;
            public float angle=90;
            public Wall(float ang, Transform wallT){ angle=ang;
wallObjT=wallT; }}
      public class Tile : MonoBehaviour {
            public int index=0;
            public _TextureTypeTile typetile;
            public _TileType type;
            public bool walkable=true;
            private bool visible=true;    //for fog of war, mark as false if
hidden in fog of war
            public bool IsVisible(){ return visible; }
            [HideInInspector] public GameObject fogOfWarObj;
            public void SetFogOfWarObj(Transform fogObjT){
                  fogOfWarObj=fogObjT.gameObject;
                  fogOfWarObj.transform.position=transform.position;
                  fogOfWarObj.transform.parent=transform.parent;
```

```
      fogOfWarObj.transform.localScale*=GridManager.GetGridToTileSizeRatio();}
            public List<CoverSystem.Cover> coverList=new
List<CoverSystem.Cover>();
            [Space(10)]
            public Unit unit=null;
            [Space(10)]
            public int spawnAreaID=-1;    //factionID of units that can be
generated on the tile, -1 means the tile is close
            public int deployAreaID=-1;   //factionID of units that can be
deploy on the tile, -1 means the tile is close
            [Space(10)]
            public float x=0;       //coordiate data for hex-tile
            public float y=0;
            public float z=0;
            public TileAStar aStar;
            public List<Tile> GetNeighbourList(bool walkableOnly=false){
return aStar.GetNeighbourList(walkableOnly); }
            public List<Tile> GetDCNeighbourList(){ return
aStar.GetDCNeighbourList(); }
            [HideInInspector]
            public int distance=0;  //for when the tile is in walkableTileList
for the selected unit, indicate the distance from selected unit
            private Transform thisT;
            public void Init(){
                  thisT=transform;}
            //stored for the current selected unit, or for AI algorithm
            [HideInInspector]
            public List<Tile> hostileInRangeList=new List<Tile>();
            public void SetHostileInRange(List<Tile> list){
hostileInRangeList=list; }
            public void ClearHostileInRange(){ hostileInRangeList=new
List<Tile>(); }
            public List<Tile> GetHostileInRange(){ return hostileInRangeList;}
            public List<Tile> SetupHostileInRange(Unit srcUnit=null){
                  hostileInRangeList=new List<Tile>();
                  if(srcUnit==null) srcUnit=unit;
                  if(srcUnit!=null)
GridManager.SetupHostileInRangeforTile(srcUnit, this);//unit.tile);
                  return hostileInRangeList;}
            //to set the visibility of the tile after checking fog-of-war
            public void SetVisible(bool flag){
                  if(forcedVisibleDuration.duration>0 && !flag) return;
                  visible=flag;
                  if(fogOfWarObj!=null) fogOfWarObj.SetActive(!visible);
                  if(!flag){
                        if(unit!=null){
      unit.gameObject.layer=TBTK.GetLayerUnitInvisible();
                              Utilities.SetLayerRecursively(unit.transform,
TBTK.GetLayerUnitInvisible());}}
                  else{
                        if(unit!=null){
                              unit.gameObject.layer=TBTK.GetLayerUnit();
                              Utilities.SetLayerRecursively(unit.transform,
TBTK.GetLayerUnit());}}}
            //for ability which reveal fog-of-war for certain duration
            [Space(10)]
            public TBDuration forcedVisibleDuration=new TBDuration();
            public void ForceVisible(int dur=1){
                  if(!GameControl.EnableFogOfWar()) return;
                  SetVisible(true);
                  if(forcedVisibleDuration.duration<dur){
                        forcedVisibleDuration.Set(dur);
                        EffectTracker.TrackVisible(this);}}
```

```
            //called by EffectTracker
            public void IterateForcedVisibleDuration(){
                  forcedVisibleDuration.Iterate();
                  if(forcedVisibleDuration.Due()){
                        SetVisible(FogOfWar.CheckTileVisibility(this));
                        EffectTracker.UntrackVisible(this);}}
            public Vector3 GetPos(){ return thisT==null ? transform.position :
thisT.position; }
            public Vector3 GetPosG(){
                  Vector3 pos=thisT==null ? transform.position :
thisT.position;
                  return new Vector3(pos.x, 0, pos.z);}
            public Tile GetNeighbourFromAngle(float angle){
                  List<Tile> neighbourList=aStar.GetNeighbourList();
                  for(int n=0; n<neighbourList.Count; n++){
                        Vector3 dir=neighbourList[n].GetPos()-GetPos();
                        float angleN=Utilities.Vector2ToAngle(new
Vector2(dir.x, dir.z));
                        if(Mathf.Abs(angle-angleN)<2) return
neighbourList[n];}
                  return null;}
            [HideInInspector] public int hostileCount=0;
            [HideInInspector] public int coverScore=0;
            public float GetCoverRating(){ return hostileCount>0 ?
(float)coverScore/(float)hostileCount : 0 ; }
            //these section are related to obstacle and wall
            [Space(10)]
            public Transform obstacleT;
            public bool HasObstacle(){ return obstacleT==null ? false : true }
            public void AddObstacle(int obsType, float gridSize){
                  if(obstacleT!=null){
            if(obstacleT.gameObject.layer==TBTK.GetLayerObstacleHalfCover() &&
obsType==1) return;
            if(obstacleT.gameObject.layer==TBTK.GetLayerObstacleFullCover() &&
obsType==2) return;}
                  if(wallList.Count>0){
                        if(!Application.isPlaying){
                              Grid grid=GridManager.GetInstance().GetGrid();
                              while(wallList.Count>0)
RemoveWall(wallList[0].angle, grid.GetNeighbourInDir(this,
wallList[0].angle));}
                        else{
                              while(wallList.Count>0)
RemoveWall(wallList[0].angle, GetNeighbourFromAngle(wallList[0].angle));}}
Transform obsT=(
                        if(obstacleT!=null)
Undo.DestroyObjectImmediate(obstacleT.gameObject);
                        Undo.RecordObject(this, "Tile");
                        Undo.RecordObject(GetComponent<Renderer>(),
"TileRenderer");
                  walkable=true;
                  GetComponent<Renderer>().enabled=true;
      }           public void CheckDiagonal(){
                  LayerMask mask=1<<TBTK.GetLayerObstacleHalfCover() |
1<<TBTK.GetLayerObstacleFullCover();
                  List<Tile> neighbourList=new List<Tile>(
aStar.GetNeighbourList() );
                  for(int n=0; n<neighbourList.Count; n++){
                        Vector3 dir=neighbourList[n].GetPos()-GetPos();
                        float angleN=Utilities.Vector2ToAngle(new
Vector2(dir.x, dir.z));
                        if(angleN%90!=0){
```

```
                        if(Physics.Linecast(GetPos(),
neighbourList[n].GetPos(), mask)){
      aStar.DisconnectNeighbour(neighbourList[n]); }}}}
            public List<Wall> wallList=new List<Wall>();
            //used in edit mode only
            public void AddWall(float angle, Tile neighbour, int wallType=0){
                  if(neighbour==null) return;
                  if(angle>360) angle-=360;
                  if(IsWalled(angle)) return;
                  float
gridSize=GridManager.GetTileSize()*GridManager.GetGridToTileSizeRatio();
                  if(type==_TileType.Square) gridSize*=2;
                        Transform
wallT=(Transform)Instantiate(GridManager.GetWallObstacleT(wallType));
                  float wallTAngle=angle+90;
                  if(type==_TileType.Square) wallTAngle=360-(angle-90);
                  else if(type==_TileType.Hex) wallTAngle=360-(angle-90);
                  wallT.rotation=Quaternion.Euler(0, wallTAngle, 0);
                  wallT.position=(GetPos()+neighbour.GetPos())/2;
                  wallT.localScale*=gridSize;
                  wallT.parent=transform;
                  wallList.Add(new Wall(angle, wallT));
                  if((angle+=180)>=360) angle-=360;
                  neighbour.wallList.Add(new Wall(angle, wallT));}
            public void RemoveWall(float angle, Tile neighbour){
                  Debug.Log(angle+"    "+neighbour);
                  if(angle>360) angle-=360;
                  for(int i=0; i<wallList.Count; i++){
                        Debug.Log(wallList[i].angle+"     "+angle);
                        if(wallList[i].angle==angle){
                              wallList.RemoveAt(i);
                              break;}}
                  if(neighbour==null) return;
                  if((angle+=180)>=360) angle-=360;
                  for(int i=0; i<neighbour.wallList.Count; i++){
                        Debug.Log(neighbour.wallList[i].angle+"     "+angle);
                        if(neighbour.wallList[i].angle==angle){
                              neighbour.wallList.RemoveAt(i);
                              break;}}}
            public bool IsWalled(float angle){
                  for(int i=0; i<wallList.Count; i++){
if(wallList[i].angle==angle) return true; }
                  return false;}
            //called during grid initiation
            public void InitWall(){
                  if(wallList.Count==0) return;
                  for(int i=0; i<wallList.Count; i++){
                        Wall wall=wallList[i];
                        if(wall.init) continue;
                        Tile neighbour=GetNeighbourFromAngle(wall.angle);
                        if(neighbour!=null){
                              wall.init=true;
                              wall.neighbour=neighbour;
                              aStar.DisconnectNeighbour(neighbour);
                              neighbour.CreateWall(this, wall.angle+180);}}}
            //call by other tile in InitWall to create a wall instance, to
avoid duplication or running the same code twice
            public void CreateWall(Tile neighbour, float angle){
                  if(angle>360) angle-=360;
                  for(int i=0; i<wallList.Count; i++){
                        Wall wall=wallList[i];
                        if(wall.angle==angle){
                              wall.init=true;
```

```
                        wall.neighbour=neighbour;
                        aStar.DisconnectNeighbour(neighbour);
                        break;}}}
//for enable/disable wall in runtime, not being used
      public void DisableWall(Transform wallObjT){
            for(int i=0; i<wallList.Count; i++){
                  if(wallList[i].wallObjT==wallObjT){
                        Tile neighbour=wallList[i].neighbour;
                        aStar.ConnectNeighbour(neighbour);
                        for(int n=0; n<neighbour.wallList.Count; n++){
if(neighbour.wallList[n].wallObjT==wallObjT){
aStar.ConnectNeighbour(neighbour.wallList[n].neighbour);
                                    if(GameControl.EnableCover()){
                        for(int j=0; j<neighbour.coverList.Count; j++){
if(Mathf.Abs(coverList[j].angle-neighbour.wallList[n].angle)<1){
coverList[j].enabled=false;}}
      break;}}
      if(GameControl.EnableCover()){
      for(int m=0; m<coverList.Count; m++){
            if(Mathf.Abs(coverList[m].angle-wallList[i].angle)<1){
                  coverList[m].enabled=false;}}}
                        break;}}}
      public void EnableWall(Transform wallObjT){
            for(int i=0; i<wallList.Count; i++){
                  if(wallList[i].wallObjT==wallObjT){
                        Tile neighbour=wallList[i].neighbour;
                        aStar.DisconnectNeighbour(neighbour);
                        for(int n=0; n<neighbour.wallList.Count; n++){
if(neighbour.wallList[n].wallObjT==wallObjT){
aStar.DisconnectNeighbour(neighbour.wallList[n].neighbour);
                  if(GameControl.EnableCover()){
                        for(int j=0; j<neighbour.coverList.Count; j++){
if(Mathf.Abs(coverList[j].angle-neighbour.wallList[n].angle)<1){
coverList[j].enabled=true;}}}
                                    break;}}
      if(GameControl.EnableCover()){
            for(int m=0; m<coverList.Count; m++){
                  if(Mathf.Abs(coverList[m].angle-wallList[i].angle)<1){
                  coverList[m].enabled=true;}}}
                        break;}}}
      //end obstacle and wall related function
      //this section are related to collectible on tile
      public Collectible collectible;
      public void TriggerCollectible(Unit unit){
            if(collectible==null) return;
            collectible.Trigger(unit);
            collectible=null;}
      //end collectible
      //these section are related to effects on tile
      [Space(10)]
      public Effect fixedEffect=new Effect();
      public List<Effect> effectList=new List<Effect>();
      public void ApplyEffect(Effect eff){
            if(eff.duration<=0) return;
            eff.Init();
            effectList.Add(eff);
            UpdateActiveEffect();
            EffectTracker.Track(this);}
      public void IterateEffectDuration(){
            bool changed=false;
            for(int i=0; i<effectList.Count; i++){
                  effectList[i].Iterate();
                  if(effectList[i].Due()){
```

```
                        effectList.RemoveAt(i); i-=1;
                        changed=true;}}
                if(changed){
                      if(effectList.Count>0) UpdateActiveEffect();
                      else{
                            activeEffect=new Effect();
                            EffectTracker.Untrack(this);}}}
          public Effect activeEffect=new Effect();
          public void UpdateActiveEffect(){
                activeEffect=new Effect();
                AddToActiveEffect(fixedEffect);
                for(int i=0; i<effectList.Count; i++)
AddToActiveEffect(effectList[i]);}
          public void AddToActiveEffect(Effect eff){
                activeEffect.HPPerTurn+=eff.HPPerTurn;
                activeEffect.APPerTurn+=eff.APPerTurn;
                activeEffect.moveAPCost+=eff.moveAPCost;
                activeEffect.attackAPCost+=eff.attackAPCost;
                activeEffect.moveRange+=eff.moveRange;
                activeEffect.attackRange+=eff.attackRange;
                activeEffect.sight+=eff.sight;
                activeEffect.damage+=eff.damage;
                activeEffect.hitChance+=eff.hitChance;
                activeEffect.dodgeChance+=eff.dodgeChance;
                activeEffect.critChance+=eff.critChance;
                activeEffect.critAvoidance+=eff.critAvoidance;
                activeEffect.critMultiplier+=eff.critMultiplier;
                activeEffect.stunChance+=eff.stunChance;
                activeEffect.stunAvoidance+=eff.stunAvoidance;
                activeEffect.stunDuration+=eff.stunDuration;
                activeEffect.silentChance+=eff.silentChance;
                activeEffect.silentAvoidance+=eff.silentAvoidance;
                activeEffect.silentDuration+=eff.silentDuration;
                activeEffect.flankingBonus+=eff.flankingBonus;
                activeEffect.flankedModifier+=eff.flankedModifier;}
          public float GetHPPerTurn(){ return activeEffect.HPPerTurn; }
          public float GetAPPerTurn(){ return activeEffect.APPerTurn; }
          public float GetMoveAPCost(){ return activeEffect.APPerTurn; }
          public float GetAttackAPCost(){ return activeEffect.APPerTurn; }
          public int GetMoveRange(){ return activeEffect.attackRange; }
          public int GetAttackRange(){ return activeEffect.attackRange; }
          public int GetSight(){ return activeEffect.sight; }
          public float GetDamage(){ return activeEffect.damage; }
          public float GetHitChance(){ return activeEffect.hitChance; }
          public float GetDodgeChance(){ return activeEffect.dodgeChance; }
          public float GetCritChance(){ return activeEffect.critChance; }
          public float GetCritAvoidance(){ return
activeEffect.critAvoidance; }
          public float GetCritMultiplier(){ return activeEffect.attackRange;
          public float GetStunChance(){ return activeEffect.stunChance; }
          public float GetStunAvoidance(){ return
activeEffect.stunAvoidance; }
          public int GetStunDuration(){ return activeEffect.stunDuration; }
          public float GetSilentChance(){ return activeEffect.silentChance;}
          public float GetSilentAvoidance(){ return
activeEffect.silentAvoidance; }
          public int GetSilentDuration(){ return
activeEffect.silentDuration; }
          public float GetFlankingBonus(){ return
activeEffect.flankingBonus; }
          public float GetFlankedModifier(){ return
activeEffect.flankedModifier; }
          //end effect related function}
```

Лістинг AIManager.cs

```
using UnityEngine;using System.Collections;using
System.Collections.Generic;using TBTK;
namespace TBTK{

	public enum _AIMode{
		Passive,	//the unit wont move unless the there are hostile
within the faction's sight (using unit sight value even when Fog-Of-War is
not used)
		Trigger,	//the unit wont move unless it's being triggered, when
it spotted any hostile or attacked
		Aggressive, //the unit will be on move all the time, looking for
potential target}
	public class AIManager : MonoBehaviour {
		public _AIMode mode=_AIMode.Passive;
		public static _AIMode GetAIMode(){ return instance.mode; }
		public float unitInterval=0.1f;
		public bool untriggeredUnitMove=false;
		[Space(10)] public bool debugMode=false;
		private static AIManager instance;
		void Awake(){
			instance=this;	}
		//called in FactionManager to move the whole faction
		public static void MoveFaction(Faction faction){
instance._MoveFaction(faction); }
		public void _MoveFaction(Faction faction){
			StartCoroutine(FactionRoutine(faction)); }
		//called in GameControl when a AI unit is selected to move that
particular unit
		public static void MoveUnit(Unit unit){ instance._MoveUnit(unit);}
		public void _MoveUnit(Unit unit){
			StartCoroutine(SingleUnitRoutine(unit));}
		public void AIDebug(string msg){
			if(debugMode) Debug.Log("AI Debug - "+msg);}
		//move the whole faction, unit by unit
		IEnumerator FactionRoutine(Faction faction){
			GameControl.DisplayMessage("AI's Turn");
			yield return new WaitForSeconds(0.3f)
			//create a new list so no unit will be skipped is one of
them is somehow destroyed (by counter attack);
			List<Unit> unitList=new List<Unit>( faction.allUnitList );
			for(int i=0; i<unitList.Count; i++){
				AIDebug("Start moving unit
"+unitList[i].gameObject.name);
				yield return new WaitForSeconds(unitInterval);
				if(unitList[i].IsStunned()) continue;
				_AIMode activeAIMode=!faction.useDefaultAIMode ?
faction.aiMode : mode ;
				while(unitList[i].moveRemain>0 ||
unitList[i].CanAttack()){
			AIDebug("Moving unit "+unitList[i].gameObject.name+"
(move remain:"+(unitList[i].moveRemain>0)+", can
attack:"+unitList[i].CanAttack()+")");
			StartCoroutine(MoveUnitRoutine(unitList[i], activeAIMode));
					while(movingUnit) yield return null;
					if(unitList[i]==null) break;
					if(unitList[i].moveRemain>0 ||
unitList[i].CanAttack()) yield return new WaitForSeconds(.1f);}
		if(GameControl.GetGamePhase()==_GamePhase.Over) yield break; }
			GameControl.EndTurn();}
		//move a single unit only
		IEnumerator SingleUnitRoutine(Unit unit){
```

```
                GameControl.DisplayMessage("AI's Turn")
                yield return new WaitForSeconds(0.1f);
                if(!unit.IsStunned()){
            Faction faction=FactionManager.GetFaction(unit.factionID);
      _AIMode activeAIMode=!faction.useDefaultAIMode ? faction.aiMode : mode ;
                while(unit.moveRemain>0 || unit.CanAttack()){
                      StartCoroutine(MoveUnitRoutine(unit, activeAIMode));
                            while(movingUnit) yield return null;
                            if(unit==null) break;
                            if(unit.attackRemain>0 || unit.CanAttack())
yield return new WaitForSeconds(.1f);}}
                GameControl.EndTurn();}
            IEnumerator DelayEndTurn(){
                yield return new WaitForSeconds(1f);
                while(!TurnControl.ClearToProceed()) yield return null;
                GameControl.EndTurn();}
            private bool movingUnit=false;
            //set to true when a unit is being moved
            IEnumerator MoveUnitRoutine(Unit unit, _AIMode activeMode){
                TBTK.OnUnitSelected(unit);
                movingUnit=true;
            if(activeMode!=_AIMode.Aggressive && !unit.trigger){
                AIDebug("unit "+unit.gameObject.name+" is not triggered");
                      if(!untriggeredUnitMove){
                            StartCoroutine(EndMoveUnitRoutine());
                            unit.moveRemain=0;      unit.attackRemain=0;}
                else{
                      if(Random.value<0.5f){
                            StartCoroutine(EndMoveUnitRoutine());
                            unit.moveRemain=0;      unit.attackRemain=0;}
                            else{
                                  AIDebug("Randomly move unit
"+unit.gameObject.name+" anyway");
                            List<Tile>
walkableTilesInRange=GridManager.GetTilesWithinDistance(unit.tile,
Mathf.Min(1, unit.GetEffectiveMoveRange()/2), true);
                                  if(walkableTilesInRange.Count>0)
unit.Move(walkableTilesInRange[Random.Range(0,
walkableTilesInRange.Count)]);}}
                      AIDebug("End unit "+unit.gameObject.name+" turn");
                      StartCoroutine(EndMoveUnitRoutine());
                      yield break;}
                Tile targetTile=Analyse(unit, activeMode);
                if(CameraControl.CenterOnSelectedUnit()){
                      bool visible=false;
                      if(unit.tile.IsVisible()) visible=true;
                      else if(targetTile!=unit.tile){
                            List<Tile> path=unit.GetPathForAI(targetTile);
                            for(int i=0; i<path.Count; i++){
                                  if(path[i].IsVisible()){
                                        visible=true;
                                        break;}}
                            targetTile=path[path.Count-1];
                            Debug.DrawLine(unit.tile.GetPos(),
targetTile.GetPos(), Color.red, 2);}
                if(visible){
                      CameraControl.OnUnitSelected(unit, false);
                      while(CameraControl.IsLerping()) yield return null;}
                //first move to the targetTile
            if(targetTile!=unit.tile){
                unit.Move(targetTile);
                yield return new WaitForSeconds(.1f);    //wait until the
unit has moved into the targetTile
```

```
                  while(!TurnControl.ClearToProceed()){
            //AIDebug("waiting, unit "+unit.gameObject.name+" is moving");
                              AIDebug("waiting while unit is moving");
                              yield return null;}}
      if(unit==null || unit.HP<=0){ //in case unit is destroyed by overwatch
                        StartCoroutine(EndMoveUnitRoutine());
                        yield break;}
                  for(int i=0; i<targetTile.hostileInRangeList.Count; i++){
                        if(targetTile.hostileInRangeList[i].unit==null ||
targetTile.hostileInRangeList[i].unit.factionID==unit.factionID){
                  targetTile.hostileInRangeList.RemoveAt(i);i-=1}}
                  //if there's hostile within range, attack it
                  if(targetTile.hostileInRangeList.Count>0){
                        if(unit.CanAttack()){
            AIDebug("waiting, unit "+unit.gameObject.name+" is attacking");}
                              int rand=Random.Range(0,
targetTile.hostileInRangeList.Count);
      unit.Attack(targetTile.hostileInRangeList[rand].unit);}
                              else{if (unit.moveRemain>0) unit.moveRemain-=1;}}
                  elseif(unit.moveRemain<=0) unit.attackRemain=0;}}
                  AIDebug("End unit "+unit.gameObject.name+" turn");
                  StartCoroutine(EndMoveUnitRoutine());
                  yield return null;}
            //clear movingUnit flag so the next unit can be moved
            IEnumerator EndMoveUnitRoutine(){
                  while(!TurnControl.ClearToProceed()){
                        AIDebug("Waiting for all sequence to complete");
                        yield return null;}
                  yield return null;
                  movingUnit=false;}
            //analyse the grid to know where the unit should move to
            private Tile Analyse(Unit unit, _AIMode activeMode){
                  //get all wakable tiles in range first
                  List<Tile>
walkableTilesInRange=GridManager.GetTilesWithinDistance(unit.tile,
unit.GetEffectiveMoveRange(), true);
                  walkableTilesInRange.Add(unit.tile);
                  //get all visible hostile
                  List<Unit>
allHostileInSight=FactionManager.GetAllHostileUnit(unit.factionID);
                  if(GameControl.EnableFogOfWar()){
                        for(int i=0; i<allHostileInSight.Count; i++){
      if(!FogOfWar.IsTileVisibleToFaction(allHostileInSight[i].tile,
unit.factionID)){
                        allHostileInSight.RemoveAt(i);      i-=1;}
                  //if cover system is in used
                  if(GameControl.EnableCover()){
                        Tile tile=AnalyseCoverSystem(unit,
walkableTilesInRange, allHostileInSight);
                        if(tile!=null) return tile;}
                  //if there are hostile
                  if(allHostileInSight.Count>0){
                        //fill up the walkableTilesInRange hostile list
                        //then filter thru walkableTilesInRange, those that
have a hostile in range will be add to a tilesWithHostileInRange
                        List<Tile> tilesWithHostileInRange=new List<Tile>();
                        GridManager.SetupHostileInRangeforTile(unit,
walkableTilesInRange);
                        for(int i=0; i<walkableTilesInRange.Count; i++){
      if(walkableTilesInRange[i].GetHostileInRange().Count>0)
tilesWithHostileInRange.Add(walkableTilesInRange[i]);}
                        //if the tilesWithHostileInRange is not empty after
the process, means there's tiles which the unit can move into and attack
```

```
                        //return one of those in the tilesWithHostileInRange
so the unit can attack
                        if(tilesWithHostileInRange.Count>0){
//if the unit current tile is one of those tiles with hostile, just stay put
and attack
                        if(tilesWithHostileInRange.Contains(unit.tile)){
//randomize it a bit so the unit do move around but not stay in place all the
time                                    if(Random.Range(0f, 1f)>0.25f) return
unit.tile;}
return tilesWithHostileInRange[Random.Range(0,
tilesWithHostileInRange.Count)];}}
//if there's not potential target at all, check if the unit has any previous
attacker
//if there are, go after the last attacker
                  if(unit.lastAttacker!=null) return unit.lastAttacker.tile;
                  //for aggresive mode with FogOfWar disabled, try move
towards the nearest unit
            if(activeMode==_AIMode.Aggressive && Random.Range(0f, 1f)>0.25f){
                        List<Unit>
allHostile=FactionManager.GetAllHostileUnit(unit.factionID);
                        float nearest=Mathf.Infinity; int nearestIndex=0;
                        for(int i=0; i<allHostile.Count; i++){
                              float
dist=GridManager.GetDistance(allHostile[i].tile, unit.tile);
                              if(dist<nearest){
                                    nearest=dist;
                                    nearestIndex=i;}}
                        return allHostile[nearestIndex].tile;}
                  //if there's really no hostile to go after, then just move
randomly in one of the walkable
                  int rand=Random.Range(0, walkableTilesInRange.Count);
                  //clear in hostileInRange list for all moveable tile so,
just in case the list is not empty (hostileInRange dont clear after each
move)
                  //so the unit dont try to attack anything when it moves into
            the targetTile
                  walkableTilesInRange[rand].SetHostileInRange(new L
                        ist<Tile>());
                  return walkableTilesInRange[rand];}
            private Tile AnalyseCoverSystem(Unit unit, List<Tile>
walkableTilesInRange, List<Unit> allHostileInSight){
                  List<Tile> walkableTilesInRangeAlt=new List<Tile>(); //for
cover system, secondary tiles with less cover
                  List<Tile> halfCoveredList=new List<Tile>();   //a list for
all the tiles with half Cover
                  List<Tile> fullCoveredList=new List<Tile>();   //a list for
all the tiles with full Cover
                  if(allHostileInSight.Count==0)
fullCoveredList=walkableTilesInRange;
                  else{ //if there are hostile in sight
                  //loop through all the walkable, record their score based on
                        for(int i=0; i<walkableTilesInRange.Count; i++){
                              Tile tile=walkableTilesInRange[i];
                              tile.hostileCount=0;
                              tile.coverScore=0;
                              //iterate through all hostile, add the count,
and cover type to the tile, this will then be used in tile.GetCoverRating()
when this loop is complete
                              for(int n=0; n<allHostileInSight.Count; n++){
                              // if the hostile is out of range, ignore it
                        Int hostileRange =a llHostileInSight[n].GetMoveRange()
                  + allHostileInSight[n].GetAttackRange() ;
```

```
      if(GridManager.GetDistance(allHostileInSight[n].tile,
tile)>hostileRange) continue;
                                    tile.hostileCount+=1;
                                    CoverSystem._CoverType
coverType=CoverSystem.GetCoverType(allHostileInSight[n].tile,
walkableTilesInRange[i]);
                                    if(coverType==CoverSystem._CoverType.Half)
tile.coverScore+=1;
                                    else
if(coverType==CoverSystem._CoverType.Full) tile.coverScore+=2;}
                              //get cover rating for the tile
                              //if score is >=2, the tile has full cover from
all hostile, so add it to fullCoveredList
                              //if score is >=1 && <2, the tile has half cover
from all hostile, so add it to halfCoveredList
            //if anything <1, the tile is exposed to hostile in some manner
                  if(tile.GetCoverRating()>=2) fullCoveredList.Add(tile);
            else if(tile.GetCoverRating()>=1) halfCoveredList.Add(tile);}}
                  //if either of the CoveredList is not empty, replace
walkableTilesInRange with that since there's no need to consider to move into
tiles without cover
                  if(fullCoveredList.Count!=0){
                        walkableTilesInRange=fullCoveredList;
                        walkableTilesInRangeAlt=halfCoveredList;}
                  else if(halfCoveredList.Count!=0)
walkableTilesInRange=halfCoveredList;
                  //if there are hostile
                  if(allHostileInSight.Count>0){
                        //fill up the walkableTilesInRange hostile list
                        //then filter thru walkableTilesInRange, those that
have a hostile in range will be add to a tilesWithHostileInRange
                        List<Tile> tilesWithHostileInRange=new List<Tile>();
                        GridManager.SetupHostileInRangeforTile(unit,
walkableTilesInRange);
                        for(int i=0; i<walkableTilesInRange.Count; i++){

      if(walkableTilesInRange[i].GetHostileInRange().Count>0)
tilesWithHostileInRange.Add(walkableTilesInRange[i]);}
                        if(tilesWithHostileInRange.Count==0){
                              GridManager.SetupHostileInRangeforTile(unit,
walkableTilesInRangeAlt);
                        for(int i=0; i<walkableTilesInRangeAlt.Count; i++){
      if(walkableTilesInRangeAlt[i].GetHostileInRange().Count>0)
tilesWithHostileInRange.Add(walkableTilesInRangeAlt[i]);}}
                        //if the tilesWithHostileInRange is not empty after
the process, means there's tiles which the unit can move into and attack
                        //return one of those in the tilesWithHostileInRange
so the unit can attack
                        if(tilesWithHostileInRange.Count>0){
                              //if the unit current tile is one of those tiles
with hostile, just stay put and attack
                              if(tilesWithHostileInRange.Contains(unit.tile)){
                              //randomize it a bit so the unit do move around
but not stay in place all the time
                        if(Random.Range(0f, 1f)>0.25f) return unit.tile;}
                              return tilesWithHostileInRange[Random.Range(0,
tilesWithHostileInRange.Count)];}}
                  return null;}}}
```

Лістинг Collectible.cs

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

```
namespace TBTK{

	public class Collectible : MonoBehaviour {
		[HideInInspector] public int prefabID=-1;
		[Space(10)]
		public Sprite icon;
		public string itemName="Collectible";
		public string desp="";
		[Space(10)]
		public List<int> facAbilityIDList=new List<int>();
		[Space(10)]
		public Effect effect;
		[Space(10)]
		public GameObject triggerEffectObj;
		public bool destroyTriggerEffect;
		public float triggerEffectDuration;
		public void Trigger(Unit unit){
			if(!destroyTriggerEffect)
ObjectPoolManager.Spawn(triggerEffectObj, transform.position,
Quaternion.identity);
			else ObjectPoolManager.Spawn(triggerEffectObj,
transform.position, Quaternion.identity, triggerEffectDuration);
	if(facAbilityIDList.Count>0){
				int facAbilityID=facAbilityIDList[Random.Range(0,
facAbilityIDList.Count)];
				FactionAbility
ability=AbilityManagerFaction.GetFactionAbility(facAbilityID);
				if(ability!=null){
					if(!ability.requireTargetSelection)
AbilityManager.ApplyAbilityEffect(null, ability.Clone(), (int)ability.type);
					else
AbilityManager.ApplyAbilityEffect(unit.tile, ability.Clone(),
(int)ability.type);}}
			unit.ApplyEffect(CloneEffect());
			CollectibleManager.TriggerCollectible(this);
			Destroy(gameObject);}
		public Effect CloneEffect(){
			Effect eff=effect.Clone();
			eff.icon=icon;
			eff.name=itemName;
			eff.desp=desp;
			return eff;}}}
```

Лістинг CollectibleManager.cs

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

namespace TBTK{

	public class CollectibleManager : MonoBehaviour {
		public bool generateCollectibleOnStart=false;
		public int activeItemLimit=4;
		public bool generateInGame=false;
		public int spawnPerTurn=2;
		public float spawnChance=0.5f;
		public GameObject spawnEffect;
		public bool autoDestroySpawnEffect=true;
		public float spawnEffectDuration=2f;
		public List<int> unavailableIDList=new List<int>();
		private List<Collectible> itemList=new List<Collectible>();
		public List<Collectible> activeItemList=new List<Collectible>();
		private static CollectibleManager instance;
```

```
            public static CollectibleManager SetInstance(){      //called in
Inspector Editor
                  if(instance==null)
instance=(CollectibleManager)FindObjectOfType(typeof(CollectibleManager));
                  return instance;}
            public static CollectibleManager GetInstance(){ return instance; }
            void Awake(){}
            public void Init(){
                  if(instance==null) instance=this;
                  InitItem();
                  if(generateCollectibleOnStart) GenerateCollectible();
                  else{
                        for(int i=0; i<activeItemList.Count; i++){
                              if(activeItemList[i]==null){
                                    activeItemList.RemoveAt(i); i-=1;}}}
                  if(spawnEffect!=null) ObjectPoolManager.New(spawnEffect);}
            public void InitItem(){
                  itemList=new List<Collectible>();
                  List<Collectible> dbList=CollectibleDB.Load();
                  for(int i=0; i<dbList.Count; i++){
                        if(!unavailableIDList.Contains(dbList[i].prefabID)){
                              itemList.Add(dbList[i]);}}}
            public static void TriggerCollectible(Collectible item){
                  instance.activeItemList.Remove(item);}
            public static float NewTurn(){ return instance._NewTurn(); }
            public float _NewTurn(){
                  if(!generateInGame) return 0;
                  if(activeItemList.Count>=activeItemLimit) return 0;
                  bool spawned=false;
                  for(int i=0; i<spawnPerTurn; i++){
                        if(activeItemList.Count>=activeItemLimit) break;
                        if(Random.value>spawnChance) continue;
                        Tile tile=GetRandomTile();
                        if(tile==null) continue;
                        int rand=Random.Range(0, itemList.Count);
                        GameObject
itemObj=(GameObject)Instantiate(itemList[rand].gameObject);
                        PlaceItemAtTile(itemObj, tile);
                        spawned=true;
                        if(spawnEffect!=null){
                              if(!autoDestroySpawnEffect)
ObjectPoolManager.Spawn(spawnEffect, tile.GetPos(), Quaternion.identity);
                              else ObjectPoolManager.Spawn(spawnEffect,
tile.GetPos(), Quaternion.identity, spawnEffectDuration);}}
                  GameControl.DisplayMessage("New Collectible!");
                  return spawned ? 2 : 0;}
            public static void GenerateCollectible(){
                  if(instance==null) SetInstance();
                  instance._GenerateCollectible();}
            public void _GenerateCollectible(){
                  _ClearAllActiveItem();
                  InitItem();
            int itemCount=Random.Range(activeItemLimit/2, activeItemLimit+1);
                  int iterateCount=0;
                  while(itemCount>0){
                        iterateCount+=1;
                        if(iterateCount>10) break;
                        Tile tile=GetRandomTile();
                        if(tile==null) break;
                        int rand=Random.Range(0, itemList.Count);
                        PlaceItemAtTile(itemObj, tile);
                        itemCount-=1;}}
            public static void ClearAllActiveItem(){
```

```
                  if(instance==null) SetInstance();
                  instance._ClearAllActiveItem();}
            public void _ClearAllActiveItem(){
                  for(int i=0; i<activeItemList.Count; i++){
                        if(activeItemList[i]!=null)
DestroyImmediate(activeItemList[i].gameObject);}
                  activeItemList=new List<Collectible>();}
            private Tile GetRandomTile(){
                  Tile tile=null;
                  List<Tile> tileList=GridManager.GetTileList();
                  int iterateCount=0;
                  while(true){
                        iterateCount+=1;
                        if(iterateCount>10) break;
                        tile=tileList[Random.Range(0, tileList.Count)];
                        if(!tile.walkable) continue;
                        if(tile.unit!=null) continue;
                        if(tile.collectible!=null) continue;
                        break;}
                  return tile;}
            public void PlaceItemAtTile(GameObject itemObj, Tile tile){
                  float rotUnit=tile.type==_TileType.Hex ? 60 : 90 ;
                  Quaternion rotation=Quaternion.Euler(0, Random.Range(0,
6)*rotUnit, 0);
                  itemObj.transform.position=tile.GetPos();
                  itemObj.transform.rotation=rotation;
                  itemObj.transform.parent=tile.transform;
                  Collectible collectible=itemObj.GetComponent<Collectible>();
                  tile.collectible=collectible;
                  activeItemList.Add(collectible);}
            //called from editor to remove item
            public void RemoveItem(Collectible item){
                  activeItemList.Remove(item);}}}
```

Лістинг FactionManager.cs

```
using UnityEngine;using System.Collections;using System.Collections.Generic;
using TBTK;
namespace TBTK{

      public class FactionManager : MonoBehaviour {
            public bool generateUnitOnStart=false;
            public bool hasAIInGame=true;
            public List<int> playerFactionIDList=new List<int>();
            public static List<int> GetPlayerFactionID(){ return
instance.playerFactionIDList; }
            public int selectedFactionID=-1;
            public List<Faction> factionList=new List<Faction>();
            public static int GetTotalFactionCount(){ return
instance.factionList.Count; }
            public static List<Faction> GetFactionList(){ return
instance.factionList; }
            public static int GetSelectedFactionID(){ return
instance.selectedFactionID; }
            public static Faction GetCurrentFaction(){ return
instance.factionList[instance.selectedFactionID]; }
            [HideInInspector] public int selectedUnitID=-1;//only use in
UnitPerTurn
            [HideInInspector] public int totalUnitCount=0;
            public List<Unit> allUnitList=new List<Unit>();//all unit from
all faction
            public static int GetTotalUnitCount(){ return
instance.totalUnitCount; }
```

```
            private static FactionManager instance;
            public static FactionManager SetInstance(){ return instance!=null
? instance :
instance=(FactionManager)FindObjectOfType(typeof(FactionManager)); }
            public static FactionManager GetInstance(){ return instance; }
            public static Transform GetTransform(){ return instance!=null ?
instance.transform : null ; }
            void Awake(){if(instance==null) instance=this;}
            public static void GameOver(){ instance._GameOver(); }
            public void _GameOver(){
                  TBData.ClearEndData();
                  //save the faction back to data if it's loaded from data
                  for(int i=0; i<factionList.Count; i++){
                        Faction fac=factionList[i];
                        if(!fac.loadFromData) continue;
                        List<TBDataUnit>
startDataList=TBData.GetStartData(fac.dataID);
                        List<TBDataUnit> list=new List<TBDataUnit>();
                        for(int m=0; m<startDataList.Count; m++){
                              for(int n=0; n<fac.allUnitList.Count; n++){
                                    if(fac.allUnitList[n].GetDataID()==m){
                        list.Add(startDataList[m].Clone(fac.allUnitList[n]));
                                          fac.allUnitList.RemoveAt(n);
                                          break;}}}
                        TBData.SetEndData(fac.dataID, list);}
                  TBData.ClearStartData();}
            //called by GameControl to initiate the factions,
            //load from data when needed, spawn the startingUnit, initiate the
unit (abillities), check if unit deployment is required....
            public void Init(){
                  if(instance==null) instance=this;
                  if(generateUnitOnStart) GenerateUnit();
                  //setup all the unit in the game
                  for(int i=0; i<factionList.Count; i++){
                        for(int n=0; n<factionList[i].allUnitList.Count; n++){
                              if(factionList[i].allUnitList[n]==null){
factionList[i].allUnitList.RemoveAt(n);  n--; continue; }
                              factionList[i].allUnitList[n].InitUnitAbility();
factionList[i].allUnitList[n].isAIUnit=!factionList[i].isPlayerFaction;}}
                  Vector3 pos=new Vector3(0, 99999, 0);
                  Quaternion rot=Quaternion.identity;
                  for(int i=0; i<factionList.Count; i++){
                        Faction fac=factionList[i];
                        //if load from data, then load the list from data and
then put it to startingUnitList
                        if(fac.loadFromData){
                              fac.startingUnitList=new List<Unit>();
                              fac.dataList=TBData.GetStartData(fac.dataID);
                              if(fac.dataList==null){
            Debug.LogWarning("TBTK faction's data not setup properly", this);
                                    continue;}
                        Debug.Log("unit from data: "+fac.dataList.Count);
                              for(int n=0; n<fac.dataList.Count; n++)
fac.startingUnitList.Add(fac.dataList[n].unit);
                              //put the data list back into the end data
first, to save the current starting lineup for next menu loading
                              //in case the player didnt finish the level and
GameOver is not called
                        TBData.SetEndData(fac.dataID, fac.dataList);}
                        else{
                              //if using default startingUnitList, make sure
none of the element in startingUnitList is empty
                              for(int n=0; n<fac.startingUnitList.Count; n++){
```

```
                                    if(fac.startingUnitList[n]==null){
fac.startingUnitList.RemoveAt(n); n-=1; }}}
                        for(int n=0; n<fac.startingUnitList.Count; n++){
                              GameObject
unitObj=(GameObject)Instantiate(fac.startingUnitList[n].gameObject, pos,
rot);
      fac.startingUnitList[n]=unitObj.GetComponent<Unit>();
                              fac.startingUnitList[n].InitUnitAbility();
                  fac.startingUnitList[n].isAIUnit=!fac.isPlayerFaction;
                              unitObj.transform.parent=transform;
                              unitObj.SetActive(false);
                              if(fac.loadFromData)
//fac.startingUnitList[n].ModifyStatsToData(fac.dataList[n], n);
      fac.dataList[n].CopyStatsToUnit(fac.startingUnitList[n], n);}
                        if(fac.isPlayerFaction && fac.startingUnitList.Count>0
&& !requireDeployment){
                              if(deployingFactionID==-1) deployingFactionID=i;
                              requireDeployment=true;}}
                  if(!GameControl.EnableManualUnitDeployment()){
                        for(int i=0; i<factionList.Count; i++){
                        if(factionList[i].startingUnitList.Count<=0) continue;
                              AutoDeployFaction(i);
                              for(int n=0; n<deployedUnitList.Count; n++){
      deployedUnitList[n].factionID=factionList[i].ID;
            factionList[i].allUnitList.Add(deployedUnitList[n]);}
                              deployedUnitList=new List<Unit>();}}}
            //called by GameControl just before the game start to initiate all
the faction, after unit deployment is done
            //sort out the unit move order, reset trigger status and what not.
            public static void SetupFaction(){ instance._SetupFaction(); }
            void _SetupFaction(){
                  for(int i=0; i<factionList.Count; i++){
                        if(factionList[i].isPlayerFaction){
                              playerFactionIDList.Add(factionList[i].ID);
                        for(int n=0; n<factionList[i].allUnitList.Count; n++){
      UpdateHostileUnitTriggerStatus(factionList[i].allUnitList[n]);}}
                        if(TurnControl.GetTurnMode()!=_TurnMode.UnitPerTurn){
      if(TurnControl.GetMoveOrder()!=_MoveOrder.Random){

      factionList[i].allUnitList=ArrangeUnitListToMovePriority(factionList[i].
allUnitList);}}
      factionList[i].NewTurn(true); //pass true to call new turn on all unit
                        //for(int n=0; n<factionList[i].allUnitList.Count;
n++) factionList[i].allUnitList[n].ResetUnitTurnData();
      totalUnitCount+=factionList[i].allUnitList.Count;}
                  if(factionList.Count==playerFactionIDList.Count)
hasAIInGame=false;
                  Debug.Log("SetupFaction  "+TurnControl.GetTurnMode());
                  if(TurnControl.GetTurnMode()==_TurnMode.UnitPerTurn){
                        for(int i=0; i<factionList.Count; i++){
                              for(int n=0; n<factionList[i].allUnitList.Count;
n++) allUnitList.Add(factionList[i].allUnitList[n]);}
      allUnitList=ArrangeUnitListToMovePriority(allUnitList);
                        Debug.Log("SetupFaction  allUnitList");}}
            public static void StartUnitDeploymentPhase(){
                  GridManager.DeployingFaction(instance.deployingFactionID);
                  TBTK.OnUnitDeployment(true);}
            public static void OnUnitDestroyed(Unit unit){
instance._OnUnitDestroyed(unit); }
            public void _OnUnitDestroyed(Unit unit){
                  totalUnitCount-=1;
                  //assume isObjectUnit is the boolean flag
                  if(unit.isObjectUnit){
```

```
                        GameControl.GameOver(0);
                  //assume 0 is the player unit faction
                        return;}
            //remove unit from allUnitList so it no longers reserve a turn
                  if(TurnControl.GetTurnMode()==_TurnMode.UnitPerTurn){
                        int ID=allUnitList.IndexOf(unit);
                        if(ID<=selectedUnitID){
                              selectedUnitID-=1;}
                        allUnitList.Remove(unit);}
                  //remove the unit from the faction, and if the faction has
no active unit remain, the faction itself is remove (out of the game)
                  for(int i=0; i<factionList.Count; i++){
                        if(factionList[i].ID==unit.factionID){
                              factionList[i].RemoveUnit(unit);
                              if(factionList[i].allUnitList.Count==0){
                                    TurnControl.OnFactionDestroyed();  //to
track cd on ability and effect
            TBTK.OnFactionDestroyed(factionList[i].ID);
                                    factionList.RemoveAt(i);
                              if(selectedFactionID>i) selectedFactionID-=1;
                              else if(selectedFactionID==i)
                                          TurnControl.EndTurn();}
                              break;}}
                  //if there's only 1 faction remain (since faction with no
active unit will be removed), then faction has won the game
                  if(factionList.Count==1)
GameControl.GameOver(factionList[0].ID);}
            //called when a unit has its turn priority changed, to update the
move order
            public static void UnitTurnPriorityChanged(List<int> facIDList){
instance._UnitTurnPriorityChanged(facIDList); }
            public void _UnitTurnPriorityChanged(List<int> facIDList){
                  if(TurnControl.GetTurnMode()==_TurnMode.UnitPerTurn){
      allUnitList=ArrangeUnitListToMovePriority(allUnitList);}
                  else if(TurnControl.GetMoveOrder()!=_MoveOrder.Random){
                        for(int i=0; i<factionList.Count; i++){
                              if(facIDList.Contains(factionList[i].ID)){
      factionList[i].allUnitList=ArrangeUnitListToMovePriority(factionList[i].
allUnitList);}}}}
      //generic function to sort a unit-list based on the unit turn priority
      public static List<Unit> ArrangeUnitListToMovePriority(List<Unit> list){
                  List<Unit> newList=new List<Unit>();
                  while(list.Count>0){
                        float highest=0;
                        int highestID=0;
                        for(int i=0; i<list.Count; i++){
                              float priority=list[i].GetTurnPriority();
                              if(priority>highest){
                                    highest=priority;
                                    highestID=i;}}
                        newList.Add(list[highestID]);
                        list.RemoveAt(highestID);}
                  return newList;}
            //called to update AI unit trigger status whenever a player unit
moved to a new tile
            public static void UpdateHostileUnitTriggerStatus(Unit unit){
                  if(!instance.hasAIInGame) return;
                  List<Unit> unitList=GetAllHostileUnit(unit.factionID);
                  for(int i=0; i<unitList.Count; i++){
                        if(GridManager.GetDistance(unit.tile,
unitList[i].tile)<=unitList[i].GetSight()){
                              unitList[i].trigger=true;}}}
            //functional but not really required atm
```

```
            //moved unit to movedUnitList in faction
            public static void UnitMoveDepleted(Unit unit){
instance._UnitMoveDepleted(unit); }
            public void _UnitMoveDepleted(Unit unit){
                  if(TurnControl.GetTurnMode()!=_TurnMode.FactionPerTurn){
                        for(int i=0; i<factionList.Count; i++){
                              if(factionList[i].ID==unit.factionID){
                                    factionList[i].UnitMoveDepleted(unit);}}}}
            //used in FactionPerTurn mode only
            public static void EndTurn_FactionPerTurn(){
instance._EndTurn_FactionPerTurn(); }
            public void _EndTurn_FactionPerTurn(){
                  GameControl.ClearSelectedUnit();
                  selectedFactionID+=1;
                  if(selectedFactionID>=factionList.Count)
selectedFactionID=0;
                  factionList[selectedFactionID].NewTurn(true);
                  if(factionList[selectedFactionID].isPlayerFaction){  //if
it's a player's faction, select a unit
                        if(TurnControl.GetMoveOrder()==_MoveOrder.Free)
_SelectNextUnitInFaction_Free();
                        else _SelectNextUnitInFaction_NotFree();}
                  else{
                  //if it's a AI's faction, execute AI move
                        if(TurnControl.GetMoveOrder()==_MoveOrder.Free)
AIManager.MoveFaction(factionList[selectedFactionID]);
                        else _SelectNextUnitInFaction_NotFree();}}
            //used in FactionPerTurn mode only
            public static bool SelectNextUnitInFaction_Free(){ return
instance._SelectNextUnitInFaction_Free(); }
            public bool _SelectNextUnitInFaction_Free(){
                  //return true is there's unit available
                  return
factionList[selectedFactionID].SelectFirstAvailableUnit();}
            public static bool SelectNextUnitInFaction_NotFree(){ return
instance._SelectNextUnitInFaction_NotFree(); }
            public bool _SelectNextUnitInFaction_NotFree(){
//return true is there's unit available, false when the queue is depleted
                  bool allUnitCycled  = factionList
[selectedFactionID].SelectNextUnitInQueue(true);
                  return !allUnitCycled;}
            //used in FactionUnitPerTurn mode only
            public static void EndTurn_FactionUnitPerTurn(){
instance._EndTurn_FactionUnitPerTurn(); }
            public void _EndTurn_FactionUnitPerTurn(){
                  selectedFactionID+=1;
                  if(selectedFactionID>=factionList.Count)
selectedFactionID=0;
                  if(TurnControl.GetMoveOrder()==_MoveOrder.Free){
                        factionList[selectedFactionID].NewTurn(true);
      //enabled if there's no need to cycle through all unit

      factionList[selectedFactionID].SelectRandomAvailableUnit();}
            else factionList[selectedFactionID].SelectNextUnitInQueue();}
            //used in UnitPerTurn mode only, select the next unit in turn
            public static void EndTurn_UnitPerTurn(){
instance._EndTurn_UnitPerTurn(); }
            public void _EndTurn_UnitPerTurn(){
                  selectedUnitID+=1;
                  if(selectedUnitID>=allUnitList.Count) selectedUnitID=0;
                  selectedFactionID=allUnitList[selectedUnitID].factionID
                  bool isUnitActive=allUnitList[selectedUnitID].NewTurn();
      //in case unit is destroyed by damage over time effect
```

```
                if(!isUnitActive){
                      _EndTurn_UnitPerTurn();
                      return;}
                if(allUnitList[selectedUnitID].isAIUnit)
AIManager.MoveUnit(allUnitList[selectedUnitID]);
            else GameControl.SelectUnit(allUnitList[selectedUnitID], false);}
            public static bool IsPlayerTurn(){
                  return (instance.playerFactionIDList.Contains (
instance.selectedFactionID )) ? true : false;}
            public static bool IsPlayerFaction(int factionID){
return (instance.playerFactionIDList.Contains(factionID)) ? true : false;
            //get the number of active unit on the grid
            public static int GetAllUnitCount(){ return GetAllUnit().Count; }
            public static List<Unit> GetAllUnit(){ return
instance._GetAllUnit(); }
            public List<Unit> _GetAllUnit(){
                  if(TurnControl.GetTurnMode()==_TurnMode.UnitPerTurn) return
allUnitList;
                  else{
                        List<Unit> list=new List<Unit>();
                        for(int i=0; i<factionList.Count; i++){
                              for(int n=0; n<factionList[i].allUnitList.Count;
n++) list.Add(factionList[i].allUnitList[n]);}
                        return list;}}
            //get all units that is hostile based on a factionID (basically
all units that has different factionID)
            public static List<Unit> GetAllHostileUnit(int factionID){ return
instance._GetAllHostileUnit(factionID); }
            public List<Unit> _GetAllHostileUnit(int factionID){
                  List<Unit> list=new List<Unit>();
                  for(int i=0; i<factionList.Count; i++){
                        if(factionID==factionList[i].ID) continue;
                        for(int n=0; n<factionList[i].allUnitList.Count; n++)
list.Add(factionList[i].allUnitList[n])}
                  return list;}
            //get all units that belong to all player factions
            public static List<Unit> GetAllPlayerUnits(){ return
instance._GetAllPlayerUnits(); }
            public List<Unit> _GetAllPlayerUnits(){
                  List<Unit> list=new List<Unit>();
                  for(int i=0; i<factionList.Count; i++){
                        if(!factionList[i].isPlayerFaction) continue;
                        for(int n=0; n<factionList[i].allUnitList.Count; n++)
list.Add(factionList[i].allUnitList[n])}
                  return list;}
            //get all unit that belong to a certain faction based on the
factionID
            public static List<Unit> GetAllUnitsOfFaction(int factionID){
return instance._GetAllUnitsOfFaction(factionID); }
            public List<Unit> _GetAllUnitsOfFaction(int factionID){
                  for(int i=0; i<factionList.Count; i++){
                        if(factionID==factionList[i].ID) return
factionList[i].allUnitList;}
                  return new List<Unit>();}
            //get a certain faction based on factionID
            public static Faction GetFaction(int factionID){ return
instance._GetFaction(factionID); }
            public Faction _GetFaction(int factionID){
                  for(int i=0; i<factionList.Count; i++){
                        if(factionList[i].ID==factionID) return
factionList[i];}
                  Debug.LogWarning("Faction with ID: "+factionID+" doesnt
exist");
```

```
                  return null;}
            //how many player's faction are there in the game
            public static int GetPlayerFactionCount(){ return
instance.playerFactionIDList.Count; }
            public static void SelectUnit(){ instance._SelectUnit(); }
            public void _SelectUnit(){
                  if(!GameControl.EnableCover()) return;

                  for(int n=0; n<factionList.Count; n++){
                        for(int i=0; i<factionList[n].allUnitList.Count; i++){

      factionList[n].allUnitList[i].UpdateCoverStatus();}}}
            //insert a unit to the grid in the middle of the game
            public static void InsertUnit(Unit unit, int factionID=0){
instance._InsertUnit(unit, factionID); }
            public void _InsertUnit(Unit unit, int factionID=0){
                  unit.factionID=factionID;
                  unit.InitUnitAbility();
                  for(int i=0; i<factionList.Count; i++){
                        if(factionList[i].ID==factionID){
                              factionList[i].allUnitList.Add(unit);
                              unit.isAIUnit=!factionList[i].isPlayerFaction;}}
                  if(TurnControl.GetTurnMode()==_TurnMode.UnitPerTurn)
allUnitList.Add(unit);
                  _UnitTurnPriorityChanged( new List<int>{ unit.factionID} );
                  unit.UpdateVisibility()
                  //if(onInsertUnitE!=null) onInsertUnitE(unit);
                  TBTK.OnNewUnit(unit);}
      //these section are related to starting unit deployment of the factions
            private bool requireDeployment=false;    //set to true when user
need to manually deploy the starting units
            public static bool RequireManualUnitDeployment(){ return
GameControl.EnableManualUnitDeployment() & instance.requireDeployment; }

            [HideInInspector] public int deployingFactionID=-1;        //Index
of current faction which units are being deployed
            [HideInInspector] public int deployingUnitID=0;            //index
of the unit currently being deployed in the startingUnitList of the faction
            public static int GetDeployingFactionID(){ return
instance.deployingFactionID; }
            public static int GetDeployingUnitID(){ return
instance.deployingUnitID; }
            public List<Unit> deployedUnitList=new List<Unit>(); //a list of
unit store the deployed unit before the deployment is complete,
      //this is to keep track of which unit is deployed and which unit is not
so the deployed unit can be remove from the grid to be deployed as well
//the list will be cleared as soon as the deployment for the faction is done
            public static void PrevDeployingUnitID(){
instance._PrevDeployingUnitID(); } //rewind deployingUnitID
            public void _PrevDeployingUnitID(){
                  deployingUnitID-=1;
                  if(deployingUnitID<0)
deployingUnitID=factionList[deployingFactionID].startingUnitList.Count-1;}
            public static void NextDeployingUnitID(){
instance._NextDeployingUnitID(); } //next deployingUnitID
            public void _NextDeployingUnitID(){
                  deployingUnitID+=1;
      if(deployingUnitID>=factionList[deployingFactionID].startingUnitList.Cou
nt) deployingUnitID=0;}
            public static void SetDeployingUnitID(int ID){
instance._SetDeployingUnitID(ID); }
            public void _SetDeployingUnitID(int ID){
```

```
                  if(deployingUnitID<0 &&
deployingUnitID>=factionList[deployingFactionID].startingUnitList.Count)
Debug.LogWarning("error", null);
                  deployingUnitID=ID;}
            public static void DeployUnitOnTile(Tile tile){
instance._DeployUnitOnTile(tile); }
            public void _DeployUnitOnTile(Tile tile){
                  if(tile.deployAreaID!=factionList[deployingFactionID].ID)
return;
      if(factionList[deployingFactionID].startingUnitList.Count<=0) return;
                  Unit unit = factionList
                  [deployingFactionID].startingUnitList[deployingUnitID];
                  deployedUnitList.Add(unit);
                  tile.unit=unit;
                  unit.tile=tile;
                  unit.transform.position=tile.GetPos();
                  unit.transform.rotation=Quaternion.Euler(0,
factionList[deployingFactionID].spawnDirection, 0);
      unit.gameObject.SetActive(true);factionList[deployingFactionID].starting
UnitList.RemoveAt(deployingUnitID);
            if(factionList[deployingFactionID].startingUnitList.Count<=0)
deployingUnitID=0;
                  else{
      if(deployingUnitID>=factionList[deployingFactionID].startingUnitList.Cou
nt) deployingUnitID-=1;}
                  //if(onUnitDeployedE!=null) onUnitDeployedE(null);
                  TBTK.OnUnitDeployed(null);}
            public static void UndeployUnit(Unit unit){
instance._UndeployUnit(unit); }
            public void _UndeployUnit(Unit unit){
                  if(!deployedUnitList.Contains(unit)) return;
                  deployedUnitList.Remove(unit);
                  unit.tile.unit=null;
                  unit.tile=null;
                  unit.gameObject.SetActive(false);
                  if(deployingUnitID==0)
factionList[deployingFactionID].startingUnitList.Add(unit);
            else {factionList
[deployingFactionID].startingUnitList.Insert(deployingUnitID-1, unit);}
                  //if(onUnitDeployedE!=null) onUnitDeployedE(null);
                  TBTK.OnUnitDeployed(null);}
            //called to complete unit deployment of a particular faction
            //move on to next faction needs unit deployment
            public static void CompleteDeployment(){
instance._CompleteDeployment(); }
            public void _CompleteDeployment(){
                  if(!_IsDeploymentComplete()) return;
                  for(int i=0; i<deployedUnitList.Count; i++){
      deployedUnitList[i].factionID=factionList[deployingFactionID].ID;
      factionList[deployingFactionID].allUnitList.Add(deployedUnitList[i]);}
                  deployedUnitList=new List<Unit>();
                  deployingUnitID=-1;
                  GridManager.FactionDeploymentComplete();
      //iterate thru all faction, get the next faction needs unit deployment
                  for(int i=deployingFactionID+1; i<factionList.Count; i++){
                        if(factionList[i].startingUnitList.Count>0){
                              if(factionList[i].isPlayerFaction){
                                    deployingFactionID=i;
                                    deployingUnitID=0;      //set to >=0 to
indicate there's a player faction needs deployment
                                    break;}
                        else{ //if it's an AI faction, automatically deploy it
                                    AutoDeployFaction(i);
```

```
                              for(int n=0; n<deployedUnitList.Count; n++){
      //deployedUnitList[n].factionID=factionList[i].ID;
      factionList[i].allUnitList.Add(deployedUnitList[n]);}}}}
                  if(deployingUnitID==-1){
            //no more faction needs deloyment, start the game
                        GameControl.StartGame();
                        //if(onUnitDeploymentPhaseE!=null)
onUnitDeploymentPhaseE(false);
                        TBTK.OnUnitDeployment(false);}
                  else{                                    //another player's
faction needs deloyment, initiate the process
                        GridManager.DeployingFaction(deployingFactionID);
                        //if(onUnitDeploymentPhaseE!=null)
onUnitDeploymentPhaseE(true);
                        TBTK.OnUnitDeployment(true);}}
            //automatically deployed the unit for the current deploying
faction, used for both AI and player faction
            public static void AutoDeployCurrentFaction(){
instance.AutoDeployFaction(); }
            public void AutoDeployFaction(int factionID=-1){
                  if(factionID==-1) factionID=deployingFactionID;
                  bool setToInvisible=GameControl.EnableFogOfWar() &
!factionList[factionID].isPlayerFaction;
                  List<Unit> unitList=factionList[factionID].startingUnitList;
                  List<Tile>
tileList=GridManager.GetDeployableTileList(factionList[factionID].ID);
                  for(int i=0; i<tileList.Count; i++){
                        if(!tileList[i].walkable || tileList[i].unit!=null ||
tileList[i].obstacleT!=null){
                              tileList.RemoveAt(i);         i-=1;}}
                  int count=0;
                  for(int i=0; i<unitList.Count; i++){
                        if(tileList.Count==0) break;
                        Unit unit=unitList[i];
                        int rand=Random.Range(0, tileList.Count);
                        Tile tile=tileList[rand];
                        tileList.RemoveAt(rand);
                        tile.unit=unit;
                        unit.tile=tile;
                        unit.transform.position=tile.GetPos();
                        unit.transform.rotation = Quaternion.Euler(0,
factionList[factionID].spawnDirection, 0);
                        unit.gameObject.SetActive(true);
                        deployedUnitList.Add(unit);
                        if(setToInvisible && !tile.IsVisible()){
      unit.gameObject.layer=TBTK.GetLayerUnitInvisible();
                        Utilities.SetLayerRecursively(unit.transform,
TBTK.GetLayerUnitInvisible());}
                        count+=1;
                        unit.factionID=factionList[factionID].ID;}
                  for(int i=0; i<count; i++) unitList.RemoveAt(0);
                  TBTK.OnUnitDeployed(null);}
            //check if a faction has deployed all its startingUnitList,
            //the deployment is considered complete when eitehr the
startingUnitList is empty or there are no more tile available
            public static bool IsDeploymentComplete(){ return
instance._IsDeploymentComplete(); }
            public bool _IsDeploymentComplete(){
                  bool
flag1=factionList[deployingFactionID].startingUnitList.Count>0;
                  bool flag2=GridManager.GetDeployableTileListCount()>0;
                  if(flag1 && flag2) return false;
                  return true;}
```

```
            public static List<Unit> GetDeployingUnitList(){
                  return
instance.factionList[instance.deployingFactionID].startingUnitList;}
            //end faction deployment mode related function
            public static void GenerateUnit(){ instance._GenerateUnit(); }
            public void _GenerateUnit(){
                  for(int i=0; i<factionList.Count; i++)
factionList[i].Spawn();}
            public void ClearUnit(){
                  for(int i=0; i<factionList.Count; i++)
factionList[i].ClearUnit();}
            //use in edit mode only
            public void RecordSpawnTilePos(){
                  for(int i=0; i<factionList.Count; i++)
factionList[i].RecordSpawnTilePos();}
            //use in edit mode only
            public void SetStartingTileListBaseOnPos(float tileSize=1){
                  for(int i=0; i<factionList.Count; i++)
factionList[i].SetStartingTileListBaseOnPos(tileSize);}
            private float gizmoSize1=0.25f;
            private float gizmoSize2=0.35f;
            void OnDrawGizmos(){
                  for(int i=0; i<factionList.Count; i++){
                        Faction fac=factionList[i];
                        Gizmos.color=fac.color;
                        for(int n=0; n<fac.spawnInfoList.Count; n++){
                              for(int m=0;
m<fac.spawnInfoList[n].startingTileList.Count; m++){
      if(fac.spawnInfoList[n].startingTileList[m]==null) continue;
                                    Vector3
pos=fac.spawnInfoList[n].startingTileList[m].GetPos();
                                    Gizmos.DrawSphere(pos, gizmoSize1);
                                    Gizmos.DrawLine(pos,
pos+Quaternion.Euler(0, fac.spawnInfoList[n].spawnDirection,
0)*Vector3.forward*gizmoSize2*2);}}
                        for(int n=0; n<fac.deployableTileList.Count; n++){
                              if(fac.deployableTileList[n]==null) continue;
                              Vector3 pos=fac.deployableTileList[n].GetPos();
                              Gizmos.DrawLine(pos+new Vector3(gizmoSize2, 0,
gizmoSize2), pos+new Vector3(-gizmoSize2, 0, -gizmoSize2));
                              Gizmos.DrawLine(pos+new Vector3(-gizmoSize2, 0,
gizmoSize2), pos+new Vector3(gizmoSize2, 0, -gizmoSize2));}}}}}
```

Лістинг GridGeneretion.cs

```
using UnityEngine;using UnityEngine.Rendering;using System.Collections;using
System.Collections.Generic;using TBTK;
namespace TBTK{

      public class GridGenerator : MonoBehaviour {
            public static float spaceXHex=0.75f;
            public static float spaceZHex=0.865f;
            private static Quaternion rot=Quaternion.Euler(-90, 0, 0);
            public static Grid GenerateSquareGrid(int width=5, int length=5,
float gridSize=1, float GridToTileRatio=1, float unwalkableRate=0,
GridManager._GridColliderType colType=GridManager._GridColliderType.Master){
                  string loadText="";
                  if(colType==GridManager._GridColliderType.Individual)
loadText="ScenePrefab/SquareTile_Collider";
                  else if(colType==GridManager._GridColliderType.Master)
loadText="ScenePrefab/SquareTile";
                  Transform tilePrefab=Resources.Load(loadText,
typeof(Transform)) as Transform;
```

```
                List<Transform> tileTransformList=new List<Transform>();
                int unwalkableCount=0;
                Vector3 pos=Vector3.zero;
                Transform parentT=new GameObject("GridTemp").transform;
                float spaceX=gridSize*GridToTileRatio;
                float spaceZ=gridSize*GridToTileRatio;
                Grid grid=new Grid(_TileType.Square, width, length,
gridSize, GridToTileRatio);
                for(int i=0; i<width; i++){
                      for(int n=0; n<length; n++){
                            pos=new Vector3(i*spaceX, 0, n*spaceZ);
                            Transform
tileT=(Transform)Instantiate(tilePrefab, pos, rot);
                            tileTransformList.Add(tileT);
                            tileT.localScale*=gridSize;
                            tileT.parent=parentT;
                            tileT.gameObject.layer=TBTK.GetLayerTile();
                            Tile tile=tileT.gameObject.AddComponent<Tile>();
                            tile.type=_TileType.Square;
                            tile.x=i;
                            tile.y=0;
                            tile.z=n;
                            grid.tileList.Add(tile);}}
                grid.AssignIndex();
                if(unwalkableRate>0){
          unwalkableCount=(int)(grid.tileList.Count*unwalkableRate);
                      for(int i=0; i<unwalkableCount; i++){
                            int rand=Random.Range(0, grid.tileList.Count);
                            grid.tileList[rand].walkable=false;
                      grid.tileList[rand].gameObject.SetActive(false);}}
                float x=(width-1)*spaceX;     float z=(length-1)*spaceZ;
                parentT.position-=new Vector3(x/2, 0, z/2);
                Transform newParentT=new GameObject("SquareGrid").transform;
                for(int i=0; i<tileTransformList.Count; i++)
tileTransformList[i].parent=newParentT;
                DestroyImmediate(parentT.gameObject);
                grid.gridObj=newParentT.gameObject;
                newParentT.gameObject.layer=TBTK.GetLayerTile();
                if(colType==GridManager._GridColliderType.Master){
                   BoxCollider
boxCol=newParentT.gameObject.AddComponent<BoxCollider>();
                boxCol.size=new Vector3(spaceX*width, 0, spaceZ*length);}
                //CombineMesh(newParentT);
                FitGridToTerrain(grid.tileList);
                return grid;}
          public static Grid GenerateHexGrid(int width=5, int length=5,
float gridSize=1, float GridToTileRatio=1, float unwalkableRate=0,
GridManager._GridColliderType colType=GridManager._GridColliderType.Master){
                string loadText="";
                if (colType == GridManager._GridColliderType.Individual)
                {loadText = "ScenePrefab/HexTile_Collider";}
                else if (colType == GridManager._GridColliderType.Master)
                {loadText = "ScenePrefab/HexTile";
                Transform tilePrefab = Resources.Load(loadText,
typeof(Transform)) as Transform;
                if (loadText == "ScenePrefab / HexTile")
                {Debug.Log("ScenePrefab == " + loadText);}
                List<Transform> tileTransformList=new List<Transform>();
                Vector3 pos=Vector3.zero;
                Transform parentT=new GameObject("GridTemp").transform;
                float spaceX=spaceXHex*gridSize*GridToTileRatio;
                float spaceZ=spaceZHex*gridSize*GridToTileRatio;
```

```
                  Grid grid=new Grid(_TileType.Hex, width, length, gridSize,
GridToTileRatio);
                  for(int i=0; i<width; i++){
                        float offsetZ=(i%2==1) ? 0 : (spaceZ/2);
                        int limit=i%2==1 ? length : length-1;
                        for(int n=0; n<limit; n++){
                              pos=new Vector3(i*spaceX, 0, n*spaceZ+offsetZ);
                              Transform
tileT=(Transform)Instantiate(tilePrefab, pos, rot);
                              tileTransformList.Add(tileT);
                              tileT.localScale*=gridSize;
                              tileT.parent=parentT;
                              tileT.gameObject.layer=TBTK.GetLayerTile();
                              Tile tile=tileT.gameObject.AddComponent<Tile>();
                              tile.type=_TileType.Hex;
                              tile.x=i;
                              tile.y=-((i+1)/2)+n;
                              tile.z=-(i/2)-n;
                              grid.tileList.Add(tile);}}
                  grid.AssignIndex();
                  if(unwalkableRate>0){
            int unwalkableCount=(int)(grid.tileList.Count*unwalkableRate);
                        for(int i=0; i<unwalkableCount; i++){
                              int rand=Random.Range(0, grid.tileList.Count);
                              grid.tileList[rand].walkable=false;
                        grid.tileList[rand].gameObject.SetActive(false);}}
                  float x=(width-1)*spaceX;    float z=(length-1)*spaceZ;
                  parentT.position-=new Vector3(x/2, 0, z/2);
                  Transform newParentT=new GameObject("HexGrid").transform;
                  for(int i=0; i<tileTransformList.Count; i++)
tileTransformList[i].parent=newParentT;
                  DestroyImmediate(parentT.gameObject);
                  grid.gridObj=newParentT.gameObject;
                  //newParentT.gameObject.AddComponent<CombineMesh>();
                  newParentT.gameObject.layer=TBTK.GetLayerTile();
                  if(colType==GridManager._GridColliderType.Master){
                        BoxCollider
boxCol=newParentT.gameObject.AddComponent<BoxCollider>();
                        boxCol.size=new Vector3(5000, 0, 5000);}
                  return grid;}
            public static void FitGridToTerrain(List<Tile> tileList, float
baseHeight=0){
                  LayerMask mask=1<<TBTK.GetLayerTerrain();
                  RaycastHit hit;
                  for(int i=0; i<tileList.Count; i++){
                        Vector3 castPoint=tileList[i].GetPos();
                        castPoint.y=50;//Mathf.Infinity;
                        if(Physics.Raycast(castPoint, Vector3.down, out hit,
Mathf.Infinity, mask)){
                              tileList[i].transform.position=hit.point+new
Vector3(0, 0.05f, 0);}}}
            public static void CombineMesh(Transform targetT){
                  MeshFilter[] meshFilters =
targetT.GetComponentsInChildren<MeshFilter>();
                  CombineInstance[] combine = new
CombineInstance[meshFilters.Length];
                  int i = 0;
                  while(i<meshFilters.Length) {
                        combine[i].mesh = meshFilters[i].sharedMesh;
                        combine[i].transform =
meshFilters[i].transform.localToWorldMatrix;
                        meshFilters[i].gameObject.SetActive(false);
                        i++;}
```

```
                  MeshRenderer mRenderer=targetT.GetComponent<MeshRenderer>();
                  if(mRenderer==null)
mRenderer=targetT.gameObject.AddComponent<MeshRenderer>();
      mRenderer.sharedMaterial=meshFilters[0].gameObject.GetComponent<MeshRend
erer>().sharedMaterial;
                  mRenderer.receiveShadows=false;
                  mRenderer.shadowCastingMode=ShadowCastingMode.Off;
                  MeshFilter mFilter=targetT.GetComponent<MeshFilter>();
                  if(mFilter==null)
mFilter=targetT.gameObject.AddComponent<MeshFilter>();
                  DestroyImmediate(mFilter.sharedMesh);
                  mFilter.sharedMesh = new Mesh();
                  mFilter.sharedMesh.CombineMeshes(combine);}}}
```

Лістинг GridManager.cs

```
using UnityEngine;using System.Collections;using System.Collections.Generic;
using TBTK;
namespace TBTK{

      public class GridManager : MonoBehaviour {
            public bool generateGridOnStart=false;
            public _TileType tileType=_TileType.Hex;
            public static _TileType GetTileType(){ return instance.tileType; }
            public enum _GridColliderType{ Master, Individual }
      public _GridColliderType gridColliderType=_GridColliderType.Master;
      public bool enableDiagonalNeighbour=false;     //for square tile only
            public static bool EnableDiagonalNeighbour(){ return
instance.enableDiagonalNeighbour; }
            public int width=5;
            public int length=5;
            public float tileSize=1;
      public static float GetTileSize(){ return instance.grid.GetTileSize(); }
            public float gridToTileRatio=1;
            public static float GetGridToTileSizeRatio(){ return
instance.grid.GetGridToTileRatio(); }
            public float unwalkableRate=0;
            //the prefab for obstacle
            public Transform obstacleWallH;
            public Transform obstacleWallF;
            public Transform obstacleHexF;
            public Transform obstacleHexH;
            public Transform obstacleSqF;
            public Transform obstacleSqH;
      public static Transform GetWallObstacleT(int type=1){ //1-half, 2-full
      return type==1 ? instance.obstacleWallH : instance.obstacleWallF ;}
            public static Transform GetObstacleT(int type=1){
                  if(instance.tileType==_TileType.Hex) return type==1 ?
instance.obstacleHexH : instance.obstacleHexF ;
                  if(instance.tileType==_TileType.Square) return type==1 ?
instance.obstacleSqH : instance.obstacleSqF ;
            return null;}
            //the prefab for cursor and indicators
            public Transform hexCursor;
            public Transform hexSelected;
            public Transform hexHostile;
            public Transform sqCursor;
            public Transform sqSelected;
            public Transform sqHostile;
            //active cursor and indicator in used during runtime
            private Transform indicatorCursor;
            private Transform indicatorSelected;
```

```
            //private List<Transform> indicatorHostileList=new
List<Transform>();
            //on grid overlays for cover
            public List<Transform> coverHOverlayList=new List<Transform>();
            public List<Transform> coverFOverlayList=new List<Transform>();
            //material for each individual tile
            public Material hexMatNormal;
            public Material hexMatSelected;
            public Material hexMatWalkable;
            public Material hexMatUnwalkable;
            public Material hexMatHostile;
            public Material hexMatRange;
            public Material hexMatAbilityAll;
            public Material hexMatAbilityHostile;
            public Material hexMatAbilityFriendly;
            public Material hexMatInvisible;
            public Material sqMatNormal;
            public Material sqMatSelected;
            public Material sqMatWalkable;
            public Material sqMatUnwalkable;
            public Material sqMatHostile;
            public Material sqMatRange;
            public Material sqMatAbilityAll;
            public Material sqMatAbilityHostile;
            public Material sqMatAbilityFriendly;
            public Material sqMatInvisible;
            public static Material GetMatNormal(){       return
instance._GetMatNormal(); }
            public static Material GetMatSelected(){ return
instance._GetMatSelected(); }
            public static Material GetMatWalkable(){ return
instance._GetMatWalkable(); }
            public static Material GetMatUnwalkable(){ return
instance._GetMatUnwalkable(); }
            public static Material GetMatHostile(){       return
instance._GetMatHostile(); }
            public static Material GetMatRange(){         return
instance._GetMatRange(); }
            public static Material GetMatAbilityAll(){    return
instance._GetMatABAll(); }
            public static Material GetMatAbilityHostile(){ return
instance._GetMatABHostile(); }
            public static Material GetMatAbilityFriendly(){ return
instance._GetMatABFriendly(); }
            public static Material GetMatInvisible(){     return
instance._GetMatInvisible(); }
            public Material _GetMatNormal(){        return
tileType==_TileType.Hex ? hexMatNormal : sqMatNormal; }
            public Material _GetMatSelected(){ return tileType==_TileType.Hex
? hexMatSelected : sqMatSelected; }
            public Material _GetMatWalkable(){ return tileType==_TileType.Hex
? hexMatWalkable : sqMatWalkable; }
            public Material _GetMatUnwalkable(){ return
tileType==_TileType.Hex ? hexMatUnwalkable : sqMatUnwalkable; }
            public Material _GetMatHostile(){       return
tileType==_TileType.Hex ? hexMatHostile : sqMatHostile; }
            public Material _GetMatRange(){         return
tileType==_TileType.Hex ? hexMatRange : sqMatRange; }
            public Material _GetMatABAll(){         return
tileType==_TileType.Hex ? hexMatAbilityAll : sqMatAbilityAll; }
            public Material _GetMatABHostile(){     return
tileType==_TileType.Hex ? hexMatAbilityHostile : sqMatAbilityHostile; }
```

```
            public Material _GetMatABFriendly(){     return
tileType==_TileType.Hex ? hexMatAbilityFriendly : sqMatAbilityFriendly; }
            public Material _GetMatInvisible(){      return
tileType==_TileType.Hex ? hexMatInvisible : sqMatInvisible; }
            //the grid instance which contains the current grid in scene
            public Grid grid=null;
            public Grid GetGrid(){ return grid; }
            public static List<Tile> GetTileList(){ return
instance.grid.tileList; }
            public static GameObject GetGridObject(){ return
instance.grid.GetGridObject();
            //temporarily tile list for selected unit storing attackable and
walkable tiles, reset when a new unit is selected
            private List<Tile> walkableTileList=new List<Tile>();
            private List<Tile> attackableTileList=new List<Tile>();
            private static GridManager instance;
            //public static void SetInstance(){ if(instance==null)
instance=(GridManager)FindObjectOfType(typeof(GridManager)); }
      public void SetInstance(){ instance=this; }    //called in GridEditor
            public static GridManager GetInstance(){ return instance; }
            void Awake(){
                  if(instance==null) instance=this;}
            // initiate all the indicators and overlay
            void Start () {   ClearAllTile();}
            //called by GameControl at the start of a scene
            public void Init(){
                  if(instance==null) instance=this;
                  if(generateGridOnStart) GenerateGrid();
                  grid.Init();
                  //if(gridColliderType==_GridColliderType.Master)
GridGenerator.CombineMeshForGrid(grid.gridObj.transform);}
            //called by GameControl to setup the grid for Fog-of-war
            public static void SetupGridForFogOfWar(){
FogOfWar.InitGrid(instance.grid.tileList); }
            public static void OnUnitDestroyed(Unit unit){
      instance._OnUnitDestroyed(unit); }
            void _OnUnitDestroyed(Unit unit){
                  if(GameControl.GetSelectedUnit()==null) return;
                  Tile tile=unit.tile;
                  if(attackableTileList.Contains(tile)){
            attackableTileList.Remove(tile);    //remove from target tile
                        OverlayManager.RemoveHostileIndicator(tile);}
      int dist=GetDistance(tile, GameControl.GetSelectedUnit().tile, true);
                  if(dist>0 &&
dist<GameControl.GetSelectedUnit().GetMoveRange()){  //if within walkable
distance, add to walkable tile since the tile is now open
                        walkableTileList.Add(tile);
            OverlayManager.ShowMoveableIndicator(walkableTileList);}}
            public static Tile GetTileOnCursor(Vector3 cursorPos){
                  LayerMask mask=1<<TBTK.GetLayerTile() |
1<<TBTK.GetLayerTerrain();
                  Ray ray = Camera.main.ScreenPointToRay(cursorPos);
                  RaycastHit hit;
                  if(Physics.Raycast(ray, out hit, Mathf.Infinity, mask)){
                        //Debug.Log(hit.collider.gameObject);
                        return GetTileOnPos(hit.point);}
                  return null;}
            public Tile tile1;
            public Tile tile2;
            // Update is called once per frame
            private Tile hoveredTile;
      public static Tile GetHoveredTile(){ return instance.hoveredTile; }
            //private int cursorID=-1;
```

```
            private bool cursorDown;
            private bool invalidCursor;
            private Vector3 cursorPosition;
            void Update() {
                  invalidCursor=false;
                  //cursorID=-1;
                  cursorPosition=Input.mousePosition;
                  if(Input.touchCount>1) invalidCursor=true;
                  else if(Input.touchCount==1){
                        cursorPosition=TBTK.GetFirstTouchPosition();
                        //cursorID=0;}
                  if(!invalidCursor){
                        Tile newTile=GetTileOnCursor(cursorPosition);
                        if(newTile!=hoveredTile){
                  if(newTile==null && hoveredTile!=null) ClearHoveredTile();
                              else{
                                    ClearHoveredTile();
                                    NewHoveredTile(newTile);}}}}
            //for debug only
            public static void DebugDrawNeighbour(Tile tile){
                  List<Tile> tileList=tile.GetNeighbourList();
                  for(int i=0; i<tileList.Count; i++){
      Debug.DrawLine(tile.GetPos(), tileList[i].GetPos(), Color.white, 1f);}
                  List<Tile> dcTileList=tile.GetDCNeighbourList();
                  for(int i=0; i<dcTileList.Count; i++){
      Debug.DrawLine(tile.GetPos(), dcTileList[i].GetPos(), Color.red, 1f);}}
            public static void OnCursorDown(int cursorID=-1){
instance._OnCursorDown(cursorID); }
            public void _OnCursorDown(int cursorID=-1){
                  //Debug.Log("_OnCursorDown");
                  if(hoveredTile==null) return
                  if(TBTK.IsCursorOnUI(cursorID)) return;
                  if(!TurnControl.ClearToProceed()) return;
                  if(GameControl.GetGamePhase()==_GamePhase.Over) return;
                  if(GameControl.GetGamePhase()==_GamePhase.UnitDeployment){
                        if(hoveredTile.unit==null)
FactionManager.DeployUnitOnTile(hoveredTile);
                        else if(hoveredTile.unit!=null)
FactionManager.UndeployUnit(hoveredTile.unit);
                        return;}
                  if(AbilityManager.InTargetMode()){
                        Debug.Log("AbilityTargetSelected");
                        targetModeTargetSelected(hoveredTile);}
                  else OnTile(hoveredTile);}
            //call when cursor just hover over a new tile
            public static void NewHoveredTile(Tile tile){
instance._NewHoveredTile(tile); }
            void _NewHoveredTile(Tile tile){
                  hoveredTile=tile;
                  if(AbilityManager.InTargetMode()){
                        SetTargetModeHoveredTile(tile);
                        return;}
                  bool isWalkable=walkableTileList.Contains(tile);
bool isSelectedUnitTile=GameControl.GetSelectedUnit()==null ? false : true;
                  if(isSelectedUnitTile)
isSelectedUnitTile=GameControl.GetSelectedUnit().tile!=tile ? false : true ;
                  //show cover overlay if cover-system is enabled
                  if(GameControl.EnableCover() && (isWalkable ||
isSelectedUnitTile)){
                        OverlayManager.ShowCoverOverlay(tile);}
      //highlight potential target for the unit to be moved into this tile
                  if(isWalkable && GameControl.GetSelectedUnit().CanAttack()){
                        SetWalkableHostileList(tile);}
```

```
                  TBTK.OnHoverTile(tile);}
            //cleared the tile which has just been hovered over by the cursor
            public static void ClearHoveredTile(){
instance._ClearHoveredTile(); }
            void _ClearHoveredTile(){
                  if(hoveredTile!=null){
                        ClearWalkableHostileList();}
                  ShowHostileIndicator(attackableTileList);
                  hoveredTile=null;
                  OverlayManager.HideCoverOverlay();
      if(AbilityManager.InTargetMode()) ClearTargetModeHoveredTile();
                  TBTK.OnHoverTile(null);}
            //for when hover over a walkable tile, show the potential target
if move into that tile
            void SetWalkableHostileList(Tile tile){
                  ClearHostileIndicator();
                  List<Tile> tempAttackableTileList=tile.GetHostileInRange();
                  for(int i=0; i<tempAttackableTileList.Count; i++){
                        if(!tempAttackableTileList[i].IsVisible()){
                              tempAttackableTileList.RemoveAt(i); i-=1;}}
                  ShowHostileIndicator(tempAttackableTileList);}
            void ClearWalkableHostileList(){ ClearHostileIndicator(); }
//these section are related to target tile selecting when using abilities
            public delegate void TargetModeCB(Tile tile);
            private TargetModeCB targetModeSelectCallBack;
            private TargetModeCB targetModeHoverCallBack;
            private TargetModeCB targetModeExitCallBack;
            public static void AbilityTargetSelectMode(TargetModeCB sCallBack,
TargetModeCB hCallBack, TargetModeCB eCallBack){
                  //instance.targetMode=true;
                  instance.targetModeSelectCallBack=sCallBack;
                  instance.targetModeHoverCallBack=hCallBack;
                  instance.targetModeExitCallBack=eCallBack;
                  ClearAllTile();}
            public static void ClearTargetSelectMode(){
                  //instance.targetMode=false;
                  instance.targetModeSelectCallBack=null;
                  instance.targetModeHoverCallBack=null;
                  instance.targetModeExitCallBack=null;
                  OverlayManager.EnableTileCursor();}
            private void SetTargetModeHoveredTile(Tile tile){
                  if(targetModeHoverCallBack!=null)
targetModeHoverCallBack(tile);}
            private void ClearTargetModeHoveredTile(){
                  targetModeExitCallBack(null);}
            private void targetModeTargetSelected(Tile tile){
                  if(targetModeSelectCallBack!=null)
targetModeSelectCallBack(tile);}
            //end target mode related function
      //calculate the tile in the grid based on a position in the world space
            public static Tile GetTileOnPos(Vector3 pos){ //the static
function is only called by GridEditor
                  return instance!=null ? instance._GetTileOnPos(pos) : null;}
            public Tile _GetTileOnPos(Vector3 point){
                  Tile tile=null;
                  Vector3 cPoint=grid.GetGridCenterPoint();
                  int gridOffsetX=width/2;
                  int gridOffsetZ=length/2;
                  if(tileType==_TileType.Hex){
            float spaceX=GridGenerator.spaceXHex*tileSize*gridToTileRatio;
            float spaceZ=GridGenerator.spaceZHex*tileSize*gridToTileRatio;
            //for symetry hex-grid
```

```
            float offX=width%2==1 ? spaceX/2 : 0;                  //depends on
the with of the gird, set the offset of x-axis
      int column=(int)Mathf.Floor((point.x+offX-cPoint.x)/spaceX)+gridOffsetX;
            float offZ=column%2==1 ? spaceZ : spaceZ/2;
//depends on the column, introduce a offset of half a tile (odd number column
has more row)
                  if(length%2==1) offZ-=spaceZ/2;
      //depends on the length of the grid, modify the offset
                  int row=(int)Mathf.Floor((point.z-offZ-
cPoint.z)/spaceZ)+gridOffsetZ;
                        int tileID=column*length+row-column/2;
grid.tileList[tileID].GetPos()+new Vector3(.5f, 0, 0), Color.red, .5f);
            if(tileID<0 || tileID>=grid.tileList.Count) return null;
                        tile=grid.tileList[tileID];
                        //Debug.Log(Vector3.Distance(tile.GetPos(), point)+"
"+(GridGenerator.spaceZHex*tileSize));
                        if(Vector3.Distance(tile.GetPos(),
point)>GridGenerator.spaceZHex*tileSize*.5f) return null;}
                  else if(tileType==_TileType.Square){
                        float spaceX=tileSize*gridToTileRatio;
                        float spaceZ=tileSize*gridToTileRatio;
                        float offX=width%2==1 ? spaceX/2 : 0;    //depends on
the with of the gird, set the offset of x-axis
                        float offZ=length%2==1 ? spaceZ/2 : 0;   //depends on
the length of the grid, introduce a offset of half a tile
                        int column=(int)Mathf.Floor((point.x+offX-
cPoint.x)/spaceX)+gridOffsetX;
                        int row=(int)Mathf.Floor((point.z+offZ-
cPoint.z)/spaceZ)+gridOffsetZ;
                        int tileID=column*length+row;
                  if(tileID<0 || tileID>=grid.tileList.Count) return null;
                        tile=grid.tileList[tileID];
                        if(Vector3.Distance(tile.GetPos(),
point)>tileSize*.65f) return null;}
                  return tile;}
            public void GenerateGrid(){
                  Debug.Log("generate grid");
                  FactionManager factionManager=FactionManager.SetInstance();
                  if(factionManager!=null){
                        factionManager.ClearUnit();
                        factionManager.RecordSpawnTilePos();     //this is to
record the tile of the spawn and deploy area
                  }
                  if(grid!=null) grid.ClearGrid();
                  if(tileType==_TileType.Hex){
                        grid=GridGenerator.GenerateHexGrid(width, length,
tileSize, gridToTileRatio, unwalkableRate, gridColliderType);}
                  else if(tileType==_TileType.Square){
                        grid=GridGenerator.GenerateSquareGrid(width, length,
tileSize, gridToTileRatio, unwalkableRate, gridColliderType);}
                  if(grid.gridObj!=null)
grid.gridObj.transform.parent=transform.parent;
                  if(factionManager!=null){
      factionManager.SetStartingTileListBaseOnPos(tileSize*gridToTileRatio);
      //this is to set the tiles of the spawn and deploy area bsaed on the
stored info earlier
                  if(factionManager.generateUnitOnStart)
factionManager._GenerateUnit();}}
            //when player click on a particular tile
            public static void OnTile(Tile tile){ instance._OnTile(tile); }
            public void _OnTile(Tile tile){
                  if(!FactionManager.IsPlayerTurn()) return;
                  if(tile.unit!=null){
```

```
                        if(attackableTileList.Contains(tile))
GameControl.GetSelectedUnit().Attack(tile.unit);
                        else GameControl.SelectUnit(tile.unit);}
                  //if the tile is within the move range of current selected
unit, move to it
                  else if(walkableTileList.Contains(tile)){
                        ClearWalkableHostileList();   //in case the unit move
into the destination and has insufficient ap to attack
                        GameControl.GetSelectedUnit().Move(tile);}
      ClearHoveredTile();     //clear the hovered tile so all the UI overlay
will be cleared}
            //when player right-click on a particular tile
            //only used to set unit, facing
            public static void OnTileAlt(Tile tile){
instance._OnTileAlt(tile); }
            public void _OnTileAlt(Tile tile){
                  if(!FactionManager.IsPlayerTurn()) return;
                  //change the unit facing}
            //select a unit, setup the walkable, attackable tiles and what not
            public static void SelectUnit(Unit unit){
                  ClearAllTile();
                  //unit.tile.SetState(_TileState.Selected);
                  if(unit.CanMove()) instance.SetupWalkableTileList(unit);
                  if(unit.CanAttack()) instance.SetupAttackableTileList(unit);
                  //instance.indicatorSelected.position=unit.tile.GetPos();
                  OverlayManager.SelectUnit(unit);}
            //function to setup and clear walkable tiles in range for current
selected unit
            private void ClearWalkableTileList(){
                  for(int i=0; i<walkableTileList.Count; i++){
                        //walkableTileList[i].SetState(_TileState.Default);
                  walkableTileList[i].hostileInRangeList=new List<Tile>();
                        walkableTileList[i].distance=0;}
                  walkableTileList=new List<Tile>();
                  OverlayManager.ClearMoveableIndicator();}
            private void SetupWalkableTileList(Unit unit){
                  ClearWalkableTileList();
                  List<Tile> newList=GetTilesWithinDistance(unit.tile,
unit.GetEffectiveMoveRange(), true, true);
                  for(int i=0; i<newList.Count; i++){
                        if(newList[i].unit==null){
                              walkableTileList.Add(newList[i]);
                              //newList[i].SetState(_TileState.Walkable);}}
                  SetupHostileInRangeforTile(unit, walkableTileList);
                  OverlayManager.ShowMoveableIndicator(walkableTileList);}
            //function to setup and clear attackble tiles in range for current
selected unit
            private void ClearAttackableTileList(){
                  //for(int i=0; i<attackableTileList.Count; i++)
attackableTileList[i].SetState(_TileState.Default);
                  attackableTileList=new List<Tile>();}
            private void SetupAttackableTileList(Unit unit){
                  ClearAttackableTileList();
                  attackableTileList=unit.tile.SetupHostileInRange();
                  //for(int i=0; i<attackableTileList.Count; i++)
attackableTileList[i].SetState(_TileState.Hostile);
                  ShowHostileIndicator(attackableTileList);}
            //given a unit and a list of tiles, setup the attackable tiles
with that unit in each of those given tiles. the attackble tile list are
stored in each corresponding tile
            public static void SetupHostileInRangeforTile(Unit unit, Tile
tile){ SetupHostileInRangeforTile(unit, new List<Tile>{ tile }); }
```

```
            public static void SetupHostileInRangeforTile(Unit unit,
List<Tile> tileList){
                  List<Unit> allUnitList=FactionManager.GetAllUnit();
                  List<Unit> allHostileUnitList=new List<Unit>();
                  for(int i=0; i<allUnitList.Count; i++){
                        if(allUnitList[i].factionID!=unit.factionID)
allHostileUnitList.Add(allUnitList[i]);}
                  List<Unit> allFriendlyUnitList=new List<Unit>();
                  if(GameControl.EnableFogOfWar())
allFriendlyUnitList=FactionManager.GetAllUnitsOfFaction(unit.factionID);
                  int range=unit.GetAttackRange();
                  int rangeMin=unit.GetAttackRangeMin();
                  int sight=unit.GetSight();
                  for(int i=0; i<tileList.Count; i++){
                        Tile srcTile=tileList[i];
                        List<Tile> hostileInRangeList=new List<Tile>();
                        for(int j=0; j<allHostileUnitList.Count; j++){
                              Tile targetTile=allHostileUnitList[j].tile;
            if(GridManager.GetDistance(srcTile, targetTile)>range) continue;
      if(GridManager.GetDistance(srcTile, targetTile)<rangeMin) continue;
                              if(!GameControl.EnableFogOfWar() &&
!GameControl.AttackThroughObstacle()){
                  if(!FogOfWar.InLOS(srcTile, targetTile, 0)) continue;}
                  bool inSight=GameControl.EnableFogOfWar() ? false : true;
                  if(GameControl.EnableFogOfWar()){
                  if(FogOfWar.InLOS(srcTile, targetTile) &&
GridManager.GetDistance(srcTile, targetTile)<=sight){
                        inSight=true;}
                  else if(!unit.requireDirectLOSToAttack){
                  for(int n=0; n<allFriendlyUnitList.Count; n++){
                  if(allFriendlyUnitList[n]==unit) continue;
                  if(GridManager.GetDistance(allFriendlyUnitList[n].tile,
targetTile)>allFriendlyUnitList[n].GetSight()) continue;
      if(FogOfWar.InLOS(allFriendlyUnitList[n].tile, targetTile)){
                        inSight=true;
                              break;}}}}
                              if(inSight) hostileInRangeList.Add(targetTile);}

                        tileList[i].SetHostileInRange(hostileInRangeList);}}
            //reset all selection, walkablelist and what not
            public static void ClearAllTile(){
                  instance.ClearWalkableTileList();
                  instance.ClearAttackableTileList();
                  instance.ClearWalkableHostileList();
                  OverlayManager.ClearSelection();}
            public void ClearHostileIndicator(){
OverlayManager.ClearHostileIndicator(); }
            public void ShowHostileIndicator(List<Tile> list){
OverlayManager.ShowHostileIndicator(list); }
            //called to setup tile for player's faction unit deployment, they
need to be highlighted
            private List<Tile> currentDeployableTileList=new List<Tile>();
            public static void DeployingFaction(int factionID){
instance._DeployingFaction(factionID); }
            public void _DeployingFaction(int factionID){
                  currentDeployableTileList=_GetDeployableTileList(factionID);
                  //for(int i=0; i<currentDeployableTileList.Count; i++)
currentDeployableTileList[i].SetState(_TileState.Range);
      OverlayManager.ShowDeploymentIndicator(currentDeployableTileList);}
            public static void FactionDeploymentComplete(){
instance._FactionDeploymentComplete(); }
            public void _FactionDeploymentComplete(){
                  currentDeployableTileList=new List<Tile>();
```

```
                  //for(int i=0; i<currentDeployableTileList.Count; i++)
currentDeployableTileList[i].SetState(_TileState.Default);
                  OverlayManager.ClearDeploymentIndicator();}
            //get delployable tile list for certain faction
            public static List<Tile> GetDeployableTileList(int factionID){
return instance._GetDeployableTileList(factionID); }
            public List<Tile> _GetDeployableTileList(int factionID){
                  List<Tile> deployableTileList=new List<Tile>();
                  for(int i=0; i<grid.tileList.Count; i++){
                        if(grid.tileList[i].deployAreaID==factionID)
deployableTileList.Add(grid.tileList[i]);}
                  return deployableTileList;}
            public static int GetDeployableTileListCount(){ return
instance._GetDeployableTileListCount(); }
            public int _GetDeployableTileListCount(){
                  int count=0;
                  for(int i=0; i<currentDeployableTileList.Count; i++){
                        if(currentDeployableTileList[i].unit==null) count+=1;}
                  return count;}
            public static bool CanAttackTile(Tile tile){ return
instance.attackableTileList.Contains(tile); }
            public static bool CanMoveToTile(Tile tile){ return
instance.walkableTileList.Contains(tile); }
            public static bool CanCastAbility(Tile tile){ return
AbilityManager.CanCastAbility(tile); }
            public static bool CanPerFormAction(Tile tile){
                  return CanAttackTile(tile) | CanMoveToTile(tile) |
CanCastAbility(tile);}
      //to get all the tiles within certain distance from a particular tile
            //set distance is only used when called from SetupWalkableTileList
(to record the distance of each tile from the source tile)
            public static List<Tile> GetTilesWithinDistance(Tile tile, int
dist, bool walkableOnly=false, bool setDistance=false, int minDist=0){
                  return instance.grid.GetTilesWithinDistance(tile, dist,
walkableOnly, setDistance, minDist);}
            //get the distance (in term of tile) between 2 tiles,
            public static int GetDistance(Tile tile1, Tile tile2, bool
walkable=false){
                  if(!walkable) return instance.grid.GetDistance(tile1,
tile2);
      else return instance.grid.GetWalkableDistance(tile1, tile2);}
            public static float GetAngleFromTileToCursor(Tile targetTile){
                  float angle=0;
                  LayerMask mask=1<<TBTK.GetLayerTile();
                  Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
                  RaycastHit hit;
                  if(Physics.Raycast(ray, out hit, Mathf.Infinity, mask)){
                        Vector3 dir=(hit.point-targetTile.GetPos());
                        if(GetTileType()==_TileType.Hex) angle=90-
Utilities.VectorToAngle60(new Vector2(dir.x, dir.z));
                        if(GetTileType()==_TileType.Square) angle=90-
Utilities.VectorToAngle90(new Vector2(dir.x, dir.z));}
                  return angle;     }
            public static List<Tile> GetTilesInALine(Tile srcTile, Tile
tgtTile, int range, bool walkableOnly=true){
                  float angle=0;
                  if(tgtTile==null) angle=GetAngleFromTileToCursor(srcTile);
                  else{
                        Vector3 dir=(tgtTile.GetPos()-srcTile.GetPos());
                        if(GetTileType()==_TileType.Hex)
angle=Utilities.VectorToAngle60(new Vector2(dir.x, dir.z));
                        if(GetTileType()==_TileType.Square)
angle=Utilities.VectorToAngle90(new Vector2(dir.x, dir.z));}
```

```
                List<Tile> tileList=new List<Tile>();
                Tile nextTile=null;
                for(int i=0; i<range; i++){
                      nextTile=srcTile.GetNeighbourFromAngle(angle);
                      if(walkableOnly && !nextTile.walkable) break;
                      if(nextTile!=null ){
                            tileList.Add(nextTile);
                            srcTile=nextTile;}
                      else break;}
                return tileList;}}}
```

Лістинг OverlayManager.cs

```
using UnityEngine;using System.Collections;using System.Collections.Generic;
using TBTK;
namespace TBTK{

      public class OverlayManager : MonoBehaviour {
            private Transform thisT;
            private int maxUnitCount;
            private Transform cursorIndicatorF;
            private Transform cursorIndicatorH;
            private Transform selectIndicator;
            private List<Transform> deploymentIndicatorList=new
            List<Transform>();
            private List<Transform> moveableIndicatorList=new
            List<Transform>();
            private List<Transform> hostileIndicatorList=new
            List<Transform>();
            private List<Transform> abRangeIndicatorList=new
            List<Transform>();
            private List<Transform> abTargetIndicatorList=new
            List<Transform>();
            private List<Transform> movedIndicatorList=new List<Transform>();
            [Tooltip("Indicator to show which unit has depleted it's moved")]
            public Transform movedIndicator;
            [Header("Hex Indicator")]
            [Tooltip("For cursor hovered over friendly unit")]
            public Transform hexCursorF;
            [Tooltip("For cursor hovered over hostile unit")]
            public Transform hexCursorH;
            [Tooltip("To show on the tile with slected unit")]
            public Transform hexSelected;
            [Tooltip("To show on tiles available for unit deployment")]
            public Transform hexDeployment;
      [Tooltip("To show on empty tiles the selected unit can move into")]
            public Transform hexMoveable;
            [Tooltip("To show on tiles containing hostile the selected unit
can attack")]
            public Transform hexHostile;
            [Tooltip("To show on tiles within the range of an ability when
selecting a target for ability")]
            public Transform hexAbRange;
            [Tooltip("To show on tiles where the ability can be cast on when
selecting a target for ability")]
            public Transform hexAbTarget;
            [Header("Square Indicator")]
            [Tooltip("For cursor hovered over friendly unit")]
            public Transform sqCursorF;
            [Tooltip("For cursor hovered over hostile unit")]
            public Transform sqCursorH;
            [Tooltip("To show on the tile with slected unit")]
            public Transform sqSelected;
```

```
            [Tooltip("To show on tiles available for unit deployment")]
            public Transform sqDeployment;
      [Tooltip("To show on empty tiles the selected unit can move into")]
            public Transform sqMoveable;
            [Tooltip("To show on tiles containing hostile the selected unit
can attack")]
            public Transform sqHostile;
            [Tooltip("To show on tiles within the range of an ability when
selecting a target for ability")]
            public Transform sqAbRange;
            [Tooltip("To show on tiles where the ability can be cast on when
selecting a target for ability")]
            public Transform sqAbTarget;
            [Header("Fog Indicator")]
            public Transform hexFogObj;
            public Transform sqFogObj;
            [Header("Cover Indicator")]
            public Transform coverOverlayH;
            public Transform coverOverlayF;
            //on grid overlays for cover
            private List<Transform> coverOverlayHList=new List<Transform>();
            private List<Transform> coverOverlayFList=new List<Transform>();
            private static OverlayManager instance;
            void Awake(){
                  instance=this;
                  thisT=transform;}
            void Start(){
                  Init();}
            // Use this for initialization
            private bool init=false;
            public void Init(){
                  if(init) return;
                  init=true;
                  thisT=transform;
                  Transform deploymentIndicator=null;
                  Transform moveableIndicator=null;
                  Transform hostileIndicator=null;
                  Transform abRangeIndicator=null;
                  Transform abTargetIndicator=null;
                  if(GridManager.GetTileType()==_TileType.Hex){
                        deploymentIndicator=hexDeployment;
                        moveableIndicator=hexMoveable;
                        hostileIndicator=hexHostile;
                        abRangeIndicator=hexAbRange;
                        abTargetIndicator=hexAbTarget;
                        cursorIndicatorF=(Transform)Instantiate(hexCursorF);
                        cursorIndicatorH=(Transform)Instantiate(hexCursorH);
                        selectIndicator=(Transform)Instantiate(hexSelected);}
                  else if(GridManager.GetTileType()==_TileType.Square){
                        deploymentIndicator=sqDeployment;
                        moveableIndicator=sqMoveable;
                        hostileIndicator=sqHostile;
                        abRangeIndicator=sqAbRange;
                        abTargetIndicator=sqAbTarget;
                        cursorIndicatorF=(Transform)Instantiate(sqCursorF);
                        cursorIndicatorH=(Transform)Instantiate(sqCursorH);
                        selectIndicator=(Transform)Instantiate(sqSelected);}
                  cursorIndicatorF.parent=thisT;
                  cursorIndicatorH.parent=thisT;
                  selectIndicator.parent=thisT;
                  for(int i=0; i<20; i++){
deploymentIndicatorList.Add((Transform)Instantiate(deploymentIndicator));
            deploymentIndicatorList[i].gameObject.SetActive(false);
```

```
                        deploymentIndicatorList[i].parent=thisT;}
                  for(int i=0; i<20; i++){
      moveableIndicatorList.Add((Transform)Instantiate(moveableIndicator));
                        moveableIndicatorList[i].gameObject.SetActive(false);
                        moveableIndicatorList[i].parent=thisT;}
                  for(int i=0; i<10; i++){
      hostileIndicatorList.Add((Transform)Instantiate(hostileIndicator));
                        hostileIndicatorList[i].gameObject.SetActive(false);
                        hostileIndicatorList[i].parent=thisT;}
                  for(int i=0; i<20; i++){
      abRangeIndicatorList.Add((Transform)Instantiate(abRangeIndicator));
                        abRangeIndicatorList[i].gameObject.SetActive(false);
                        abRangeIndicatorList[i].parent=thisT;}
                  for(int i=0; i<20; i++){
      abTargetIndicatorList.Add((Transform)Instantiate(abTargetIndicator));
                        abTargetIndicatorList[i].gameObject.SetActive(false);
                        abTargetIndicatorList[i].parent=thisT;}
                  if(TurnControl.GetTurnMode()==_TurnMode.FactionPerTurn){
                        //create the moved indicator
                        for(int i=0; i<10; i++){
      movedIndicatorList.Add((Transform)Instantiate(movedIndicator));
            movedIndicatorList[i].gameObject.SetActive(false);
                        movedIndicatorList[i].parent=thisT; }}
                  if(GameControl.EnableFogOfWar()){
                        Transform fogObj=null;
      if(GridManager.GetTileType()==_TileType.Hex) fogObj=hexFogObj;
      else if(GridManager.GetTileType()==_TileType.Square) fogObj=sqFogObj;
                        List<Tile> tileList=GridManager.GetTileList();
                        for(int i=0; i<tileList.Count; i++){
                        //if(!tileList[i].walkable) continue;
            tileList[i].SetFogOfWarObj((Transform)Instantiate(fogObj));}}
                  if(GameControl.EnableCover()){
float scaleOffset=GridManager.GetTileType()==_TileType.Hex ? 0.5f : 0.8f ;
            float tileSize=GridManager.GetTileSize();
                        for(int i=0; i<5; i++){
      coverOverlayHList.Add((Transform)Instantiate(coverOverlayH));
      coverOverlayFList.Add((Transform)Instantiate(coverOverlayF));
      coverOverlayHList[i].localScale*=tileSize*scaleOffset;
      coverOverlayHList[i].parent=thisT;
      coverOverlayHList[i].gameObject.SetActive(false);
      coverOverlayFList[i].localScale*=tileSize*scaleOffset;
      coverOverlayFList[i].parent=thisT;
      coverOverlayFList[i].gameObject.SetActive(false);}}}
            public static void ShowCoverOverlay(Tile tile){
instance._ShowCoverOverlay(tile); }
            public void _ShowCoverOverlay(Tile tile){
                  for(int i=0; i<tile.coverList.Count; i++){
                        if(!tile.coverList[i].enabled) continue;
                        Transform overlayT=null;
      if(tile.coverList[i].type==CoverSystem._CoverType.Full)
overlayT=coverOverlayFList[i];
      if(tile.coverList[i].type==CoverSystem._CoverType.Half)
overlayT=coverOverlayHList[i];
                        overlayT.position=tile.coverList[i].overlayPos;
                        overlayT.rotation=tile.coverList[i].overlayRot;
                        overlayT.gameObject.SetActive(true);}}
            public static void HideCoverOverlay(){
instance._HideCoverOverlay(); }
            public void _HideCoverOverlay(){
                  for(int i=0; i<coverOverlayHList.Count; i++)
coverOverlayHList[i].gameObject.SetActive(false);
                  for(int i=0; i<coverOverlayFList.Count; i++)
coverOverlayFList[i].gameObject.SetActive(false);}
```

```
            void Update(){TileCursor();}
            private bool enableTileCursor=true;
            public static void EnableTileCursor(){
instance.enableTileCursor=true; }
            public static void DisableTileCursor(){
instance.enableTileCursor=false; }
            private void TileCursor(){
                  if(!enableTileCursor) return;
                  if(GridManager.GetHoveredTile()==null){
                        cursorIndicatorF.gameObject.SetActive(false);
                        cursorIndicatorH.gameObject.SetActive(false);}
                        else{cursorIndicatorF.position=GridManager.GetHovered
                  Tile().GetPos()+new Vector3(0, 0.05f, 0);
                        cursorIndicatorH.position=GridManager.GetHoveredTile()
                  .GetPos()+new Vector3(0, 0.05f, 0);
      bool attackable=GridManager.CanAttackTile(GridManager.GetHoveredTile());
                        cursorIndicatorH.gameObject.SetActive(attackable);// &
!selected);
                        cursorIndicatorF.gameObject.SetActive(!attackable);//
& !selected);}
            //new turn, clear all indicator
            private void OnNewTurn(){
            if(TurnControl.GetTurnMode()!=_TurnMode.FactionPerTurn) return;
                  for(int i=0; i<movedIndicatorList.Count; i++){
                        movedIndicatorList[i].gameObject.SetActive(false);}}
//when a unit completes it's move, put an unused indicator on the unit tile
            //this will only be called if the unit in question is player unit
            private void OnUnitMoveDepleted(Unit unit){
      if(TurnControl.GetTurnMode()!=_TurnMode.FactionPerTurn) return;
                  int index=-1;
                  for(int i=0; i<movedIndicatorList.Count; i++){
      if(movedIndicatorList[i].gameObject.activeInHierarchy){
                  index=i;
                  break;//continue;}}
                  if(index==-1){
                        Transform
moveIndicatorT=(Transform)Instantiate(movedIndicatorList[0]);
                        moveIndicatorT.parent=transform;
                        movedIndicatorList.Add(moveIndicatorT);
                        index=movedIndicatorList.Count-1;}
                  movedIndicatorList[index].position=unit.tile.GetPos();
                  movedIndicatorList[index].gameObject.SetActive(true);}
            public static void ClearDeploymentIndicator(){
instance._ClearDeploymentIndicator(); }
            public void _ClearDeploymentIndicator(){
                  for(int i=0; i<deploymentIndicatorList.Count; i++)
deploymentIndicatorList[i].gameObject.SetActive(false);}
            public static void ShowDeploymentIndicator(List<Tile> list){
instance._ShowDeploymentIndicator(list); }
            public void _ShowDeploymentIndicator(List<Tile> list){
                  while(deploymentIndicatorList.Count<list.Count){
                        Transform
deploymentIndicatorT=(Transform)Instantiate(deploymentIndicatorList[0]);
                        deploymentIndicatorT.parent=transform;
                        deploymentIndicatorList.Add(deploymentIndicatorT);}
            for(int i=0; i<list.Count; i++){
            deploymentIndicatorList[i].position=list[i].GetPos();
      deploymentIndicatorList[i].gameObject.SetActive(true);}}
            public static void ClearMoveableIndicator(){
instance._ClearMoveableIndicator(); }
            public void _ClearMoveableIndicator(){
                  for(int i=0; i<moveableIndicatorList.Count; i++)
moveableIndicatorList[i].gameObject.SetActive(false);}
```

```
            public static void ShowMoveableIndicator(List<Tile> list){
instance._ShowMoveableIndicator(list); }
            public void _ShowMoveableIndicator(List<Tile> list){
                  while(moveableIndicatorList.Count<list.Count){
                        Transform
moveableIndicatorT=(Transform)Instantiate(moveableIndicatorList[0]);
                        moveableIndicatorT.parent=transform;
                        moveableIndicatorList.Add(moveableIndicatorT);}
                  for(int i=0; i<list.Count; i++){
                        moveableIndicatorList[i].position=list[i].GetPos();
                        moveableIndicatorList[i].gameObject.SetActive(true);
                        ScaleTransformUp(moveableIndicatorList[i]);}}
            public static void ClearHostileIndicator(){
instance._ClearHostileIndicator(); }
            public void _ClearHostileIndicator(){
                  for(int i=0; i<hostileIndicatorList.Count; i++)
hostileIndicatorList[i].gameObject.SetActive(false);}
            public static void ShowHostileIndicator(List<Tile> list){
instance._ShowHostileIndicator(list); }
            public void _ShowHostileIndicator(List<Tile> list){
                  while(hostileIndicatorList.Count<list.Count){
                        Transform
hostileIndicatorT=(Transform)Instantiate(hostileIndicatorList[0]);
                        hostileIndicatorT.parent=transform;
                        hostileIndicatorList.Add(hostileIndicatorT);}
                  for(int i=0; i<list.Count; i++){
                        hostileIndicatorList[i].position=list[i].GetPos();
                        hostileIndicatorList[i].gameObject.SetActive(true);
                        ScaleTransformUp(hostileIndicatorList[i]);}}
            public static void RemoveHostileIndicator(Tile tile){
instance._RemoveHostileIndicator(tile); }
            public void _RemoveHostileIndicator(Tile tile){
                  for(int i=0; i<hostileIndicatorList.Count; i++){
                        if(hostileIndicatorList[i].position==tile.GetPos())
hostileIndicatorList[i].gameObject.SetActive(false);}}
            public static void ClearAbilityRangeIndicator(){
instance._ClearAbilityRangeIndicator(); }
            public void _ClearAbilityRangeIndicator(){
                  for(int i=0; i<abRangeIndicatorList.Count; i++)
abRangeIndicatorList[i].gameObject.SetActive(false);}
            public static void ShowAbilityRangeIndicator(List<Tile> list){
instance._ShowAbilityRangeIndicator(list); }
            public void _ShowAbilityRangeIndicator(List<Tile> list){
                  while(abRangeIndicatorList.Count<list.Count){
                  Transform
abRangeIndicatorT=(Transform)Instantiate(abRangeIndicatorList[0]);
                        abRangeIndicatorT.parent=transform;
                        abRangeIndicatorList.Add(abRangeIndicatorT);}
                  for(int i=0; i<list.Count; i++){
                        abRangeIndicatorList[i].position=list[i].GetPos();
                        abRangeIndicatorList[i].gameObject.SetActive(true);
                        ScaleTransformUp(abRangeIndicatorList[i])}}
      public static void ClearAbilityTargetIndicator(){
instance._ClearAbilityTargetIndicator(); }
            public void _ClearAbilityTargetIndicator(){
                  for(int i=0; i<abTargetIndicatorList.Count; i++)
abTargetIndicatorList[i].gameObject.SetActive(false);}
            public static void ShowAbilityTargetIndicator(List<Tile> list){
instance._ShowAbilityTargetIndicator(list); }
            public void _ShowAbilityTargetIndicator(List<Tile> list){
                  while(abTargetIndicatorList.Count<list.Count){
                        Transform
abTargetIndicatorT=(Transform)Instantiate(abTargetIndicatorList[0]);
```

```
                        abTargetIndicatorT.parent=transform;
                        abTargetIndicatorList.Add(abTargetIndicatorT);}
                  for(int i=0; i<list.Count; i++){
                        abTargetIndicatorList[i].position=list[i].GetPos();
                        abTargetIndicatorList[i].gameObject.SetActive(true);
                        ScaleTransformUp(abTargetIndicatorList[i]);}}
            public static void SelectUnit(Unit unit){
                  instance.selectIndicator.position=unit.tile.GetPos()+new
Vector3(0, 0.05f, 0);
                  instance.selectIndicator.gameObject.SetActive(true);}
      public static void ClearSelection(){
                  instance.selectIndicator.gameObject.SetActive(false);
                  instance._ClearMoveableIndicator();
                  instance._ClearHostileIndicator();}
            void ScaleTransformDown(Transform targetT, float duration=0.125f){
                  //StartCoroutine(ScaleTransformRoutine(targetT,
GridManager.GetTileSize(), 0, duration))}
            void ScaleTransformUp(Transform targetT, float duration=0.125f){
                  //StartCoroutine(ScaleTransformRoutine(targetT, 0,
GridManager.GetTileSize(), duration)); }
            IEnumerator ScaleTransformRoutine(Transform targetT, float
startVal, float endVal, float dur=0.25f){
                  targetT.gameObject.SetActive(true);
                  float duration=0;
                  Vector3 start=new Vector3(startVal, startVal, startVal);
                  Vector3 end=new Vector3(endVal, endVal, endVal);
                  targetT.localScale=start;
                  while(duration<1){
                        targetT.localScale=Vector3.Lerp(start, end, duration);
                        duration+=Time.deltaTime*(1f/dur);
                        yield return null;}
                  targetT.localScale=end;}}}
```

Лістинг TurnContor.cs

```
using UnityEngine;using System.Collections;using
System.Collections.Generic;using TBTK;
namespace TBTK{
      public enum _TurnMode{
            FactionPerTurn,
//each faction take turn to move all units in each round
            FactionUnitPerTurn,
//each faction take turn to move a single unit in each round
            //doesnt use move order
            UnitPerTurn,
//all units (regardless of faction) take turn to move according to the stats,
when all unit is moves, the round is completed}
      public enum _MoveOrder{
            Free,
            //unit switching is enabled
            Random,
            //random fix an order and follow the order throughout
            StatsBased
            //arrange the order based on unit's stats}
      public class TurnControl : MonoBehaviour{
            [HideInInspector] public _TurnMode turnMode;
            public static _TurnMode GetTurnMode(){ return instance.turnMode; }
            [HideInInspector] public _MoveOrder moveOrder;
      public static _MoveOrder GetMoveOrder(){ return instance.moveOrder; }
            //this is the flag/counter indicate how many action are on-going,
no new action should be able to start as long as this is not clear(>0)
            private static int actionInProgress=0;
private int currentTurnID=-1; //indicate how many turn has passed, not in used
```

```
private bool moved=false;     //to check if player has make a move on the
turn, used to lock unit switching in FactionUnitPerTurn-FreeMoveOrder mode
            public static void CheckPlayerMoveFlag(bool flag=true){
instance.moved=flag; }
            public static bool HasMoved(){ return instance.moved;
            public static TurnControl instance;
            void Awake(){if(instance==null) instance=this;}
            public void Init(){
                  if(instance==null) instance=this;
                  actionInProgress=0;
                  currentTurnID=-1;
      if(turnMode==_TurnMode.UnitPerTurn) moveOrder=_MoveOrder.StatsBased;}
            //call by unit when all action is depleted
            public static void SelectedUnitMoveDepleted(){
                  if(GameControl.GetGamePhase()==_GamePhase.Over) return;
                  if(instance.turnMode==_TurnMode.FactionPerTurn){
      //if not in free move order, cant switch to next unit without end turn
                        if(instance.moveOrder!=_MoveOrder.Free){
                              TBTK.OnAllUnitOutOfMove();
                              if(GameControl.GetSelectedUnit().HP<=0)
instance.StartCoroutine(instance.AutoEndTurn());}
                        else{
      if(!FactionManager.SelectNextUnitInFaction_Free())
TBTK.OnAllUnitOutOfMove();}}
                  else if(instance.turnMode==_TurnMode.FactionUnitPerTurn){
                        TBTK.OnAllUnitOutOfMove();
                        if(GameControl.GetSelectedUnit().HP<=0)
instance.StartCoroutine(instance.AutoEndTurn());}
                  else if(instance.turnMode==_TurnMode.UnitPerTurn){
                        TBTK.OnAllUnitOutOfMove();
                        if(GameControl.GetSelectedUnit().HP<=0)
instance.StartCoroutine(instance.AutoEndTurn());}}
            //for when selected unit is destroyed during counter attack
            public IEnumerator AutoEndTurn(){
                  while(!ClearToProceed()) yield return null;
                  yield return new WaitForSeconds(0.5f);
                  if(GameControl.GetGamePhase()!=_GamePhase.Over)
GameControl.EndTurn();}
            public static void StartGame(){ instance._StartGame(); }
            public void _StartGame(){
                  if(turnMode==_TurnMode.FactionPerTurn &&
moveOrder!=_MoveOrder.Free){
                        FactionManager.EndTurn_FactionPerTurn();
                        TBTK.OnNewTurn(IsPlayerTurn());}
                  else EndTurn();}
            //called in GameControl when endTurn button is pressed, move the
turn forward
            //also used when the game first started
            public static void EndTurn(){
instance.StartCoroutine(instance._EndTurn()); }
            public IEnumerator _EndTurn(){
                  if(GameControl.GetGamePhase()==_GamePhase.Over) yield break;
                  yield return new WaitForSeconds(0.2f);
                  currentTurnID+=1;
                  if(turnMode==_TurnMode.FactionPerTurn){
                        if(moveOrder==_MoveOrder.Free)
FactionManager.EndTurn_FactionPerTurn();
                        else{
                  if(FactionManager.SelectNextUnitInFaction_NotFree()){
                                    TBTK.OnNewTurn(IsPlayerTurn());
                                    yield break;}
                              else FactionManager.EndTurn_FactionPerTurn();}}
                  else if(turnMode==_TurnMode.FactionUnitPerTurn){
```

```
                        CheckPlayerMoveFlag(false);
                        FactionManager.EndTurn_FactionUnitPerTurn();}
                  else if(turnMode==_TurnMode.UnitPerTurn){
                        FactionManager.EndTurn_UnitPerTurn();
                  IterateEndTurn();
                  TBTK.OnNewTurn(IsPlayerTurn());}
            public void IterateEndTurn(){
                  List<Faction> facList=FactionManager.GetFactionList();
                  for(int i=0; i<facList.Count; i++){
                        Faction fac=facList[i]
                        for(int n=0; n<fac.abilityInfo.abilityList.Count; n++)
fac.abilityInfo.abilityList[n].currentCD.Iterate();
                        for(int n=0; n<fac.allUnitList.Count; n++){
                              Unit unit=fac.allUnitList[n];
                              for(int m=0; m<unit.abilityList.Count; m++)
unit.abilityList[m].currentCD.Iterate();}}
                  EffectTracker.IterateEffectDuration();}
            public static void OnFactionDestroyed(){
                  if(TurnControl.GetTurnMode()==_TurnMode.UnitPerTurn) return;
                  instance.IterateEndTurn();}
            public static void OnUnitDestroyed(){
                  if(TurnControl.GetTurnMode()!=_TurnMode.UnitPerTurn) return;
                  instance.IterateEndTurn();}
            public static bool IsPlayerTurn(){
                  return FactionManager.IsPlayerTurn();}
            //called by all to check if a new action can take place (shoot,
move, ability, etc)
            public static bool ClearToProceed(){
                  return (actionInProgress==0) ? true : false;}
            public static bool ClearToProceedFromOverWatch(){
                  return (actionInProgress<=1) ? true : false;}
            //called to indicate that an action has been started, prevent any
other action from starting
            public static void ActionCommenced(){
                  if(actionInProgress==0) TBTK.OnGameInAction(true);
                  actionInProgress+=1;}
            //called to indicate that an action has been completed
            public static void ActionCompleted(float delay=0){
instance.StartCoroutine(instance._ActionCompleted(delay)); }
            IEnumerator _ActionCompleted(float delay=0){
                  if(delay>0) yield return new WaitForSeconds(delay);
                  actionInProgress=Mathf.Max(0, actionInProgress-=1);
                  //yield return null;
                  if(actionInProgress==0) TBTK.OnGameInAction(false);}}}
```

Лістинг Unit.cs

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

namespace TBTK{

      public class Unit : MonoBehaviour {
            public delegate void ActionCamHandler(Unit unit, Vector3
targetPos, float chance);
            public static event ActionCamHandler onActionCamE;
      //listen by camera only
            [HideInInspector] public bool isObjectUnit=false;
            public int prefabID;
            public int instanceID;
            public int factionID;
            private int dataID=-1;//use to identify unit when using data
```

```
            public int GetDataID(){ return dataID; }
            public void SetDataID(int ID){ dataID=ID; }
            private int level=1;    //for data only, has no effect on game
            public int GetLevel(){ return level; }
            public void SetLevel(int lvl){ level=lvl; }
            public string unitName="Unit";
            public string desp="";
            public Sprite iconSprite;
            [HideInInspector] public bool trigger=false;
            [HideInInspector] public bool isAIUnit=false;
            [HideInInspector] public Unit lastAttacker;
            [HideInInspector] public Unit lastTarget;
            public int damageType=0;
            public int armorType=0;
            public Tile tile; //occupied tile
            public float hitThreshold=0.25f;
            public Transform targetPoint;
            public Transform GetTargetT(){ return targetPoint==null ? thisT :
targetPoint; }
            public bool requireDirectLOSToAttack=true;
            public int value=5;
            public float moveSpeed=10;
            [HideInInspector] private float rotateSpeed=3;
public bool IsStunned(){ return activeEffect.stun; }//>0 ? true : false; }
            public bool DisableAbilities(){ return activeEffect.silence; }
            public int silenced=0;
            public bool IsSilenced(){ return silenced>0 ? true : false; }
            [Header("Basic Stats")]
            public float defaultHP=10;
            public float defaultAP=10;
            public float HP=10;public float AP=10;
            public float HPPerTurn=0;public float APPerTurn=0;
            public float moveAPCost=0;
            public float attackAPCost=0;
            public float turnPriority=1;
            public int moveRange=3;
            public int attackRange=3;
            public int attackRangeMin=0;
            public int sight=6;
            public int movePerTurn=1;
            public int attackPerTurn=1;
            public int counterPerTurn=1;  //counter attack
            public int abilityPerTurn=1;       //ability
            [HideInInspector]
            public int moveRemain=1;
            [HideInInspector]
            public int attackRemain=1;
            [HideInInspector]
            public int counterRemain=1;
            [HideInInspector]
            public int abilityRemain=1;
            public float damageMin=3;
            public float damageMax=6;public float hitChance=.8f;
            public float dodgeChance=.15f;public float critChance=0.1f;
            public float critAvoidance=0  public float critMultiplier=2;
            public float stunChance=0;public float stunAvoidance=0;
            public int stunDuration=1    private Effect stunEffect;
            public float silentChance=0;public float silentAvoidance=0;
            public int silentDuration=1;private Effect silentEffect;
            public float flankingBonus=0;public float flankedModifier=0;
            public bool overwatching=false;          [Header("Hybrid Unit")]
            public bool isHybridUnit;
            public int meleeRange;
```

```
            public float damageMinMelee=3;
            public float damageMaxMelee=6;
            public float hitChanceMelee=.8f;
            public float critChanceMelee=0.1f;
            public int GetMeleeRange(){ return meleeRange; }
            //these section are functions that get active stats of unit
            public float GetFullHP(){ return
defaultHP*(1+GetEffHPBuff()+PerkManager.GetUnitHPBuff(prefabID)); }
            public float GetFullAP(){ return
defaultAP*(1+GetEffAPBuff()+PerkManager.GetUnitAPBuff(prefabID)); }
            public float GetHPPerTurn(){ return
HPPerTurn+GetEffHPPerTurn()+tile.GetHPPerTurn()+PerkManager.GetUnitHPPerTurn(
prefabID); }
            public float GetAPPerTurn(){ return
APPerTurn+GetEffAPPerTurn()+tile.GetAPPerTurn()+PerkManager.GetUnitAPPerTurn(
prefabID); }
            public float GetMoveAPCost(){ return (GameControl.UseAPForMove())
? Mathf.Max(0,
moveAPCost+GetEffMoveAPCost()+PerkManager.GetUnitMoveAPCost(prefabID)) : 0 ;}
            public float GetAttackAPCost(){ return
(GameControl.UseAPForAttack()) ? Mathf.Max(0,
attackAPCost+GetEffAttackAPCost()+PerkManager.GetUnitAttackAPCost(prefabID))
: 0 ; }
            public float GetCounterAPCost(){ return
GetAttackAPCost()*GameControl.GetCounterAPMultiplier(); }
            public float GetTurnPriority(){ return
turnPriority+GetEffTurnPriority()+PerkManager.GetUnitTurnPriority(prefabID);}
      public int GetMoveRange(){ return
moveRange+GetEffMoveRange()+tile.GetMoveRange()+PerkManager.GetUnitMoveRange(
prefabID); }
            public int GetAttackRange(){ return
attackRange+GetEffAttackRange()+tile.GetAttackRange()+PerkManager.GetUnitAtta
ckRange(prefabID); }
            public int GetAttackRangeMin(){ return attackRangeMin; }
            public int GetSight(){ return
sight+GetEffSight()+tile.GetSight()+PerkManager.GetUnitSight(prefabID); }
            public int GetMovePerTurn(){ return
movePerTurn+GetEffMovePerTurn()+PerkManager.GetUnitMovePerTurn(prefabID); }
      public int GetAttackPerTurn(){return attackPerTurn+GetEffAttackPerTurn()
+PerkManager.GetUnitAttackPerTurn(prefabID); }
public int GetCounterPerTurn(){ return counterPerTurn+GetEffCounterPerTurn()
+PerkManager.GetUnitCounterPerTurn(prefabID); }
      public float GetDamageMin(){ return damageMin*(1+GetEffDamage()+
tile.GetDamage()+PerkManager.GetUnitDamage(prefabID)); }
      public float GetDamageMax(){ return damageMax*(1+
GetEffDamage()+tile.GetDamage()+PerkManager.GetUnitDamage(prefabID)); }
      public float GetDamageMinMelee(){ return damageMinMelee*
(1+GetEffDamage()+ tile.GetDamage()+PerkManager.GetUnitDamage(prefabID)); }
            public float GetDamageMaxMelee(){ return damageMaxMelee*
(1+GetEffDamage()+tile.GetDamage()+PerkManager.GetUnitDamage(prefabID)); }
      public float GetHitChance(){ return hitChance+ GetEffHitChance()+
tile.GetHitChance()+PerkManager.GetUnitHitChance(prefabID); }
      public float GetDodgeChance(){ return dodgeChance+ GetEffDodgeChance( )
+ tile.GetDodgeChance()+PerkManager.GetUnitDodgeChance(prefabID); }
      public float GetHitChanceMelee(){ return hitChanceMelee+GetEffHitChance(
)+tile.GetHitChance()+PerkManager.GetUnitHitChance(prefabID); }
public float GetCritChanceMelee(){ return critChanceMelee+GetEffCritChance()+
tile.GetCritChance()+PerkManager.GetUnitCritChance(prefabID); }
      public float GetCritChance(){ return critChance+GetEffCritChance()
+tile.GetCritChance()+PerkManager.GetUnitCritChance(prefabID); }
public float GetCritAvoidance(){ return critAvoidance+GetEffCritAvoidance()
+tile.GetCritAvoidance()+PerkManager.GetUnitCritChance(prefabID); }
```

```
public float GetCritMultiplier(){ returncritMultiplier+GetEffCritMultiplier()
+tile.GetCritMultiplier()+PerkManager.GetUnitCritChance(prefabID); }
public float GetStunChance(){ return stunChance+GetEffStunChance()+
tile.GetStunChance()+PerkManager.GetUnitStunCh ance(prefabID); }
public float GetStunAvoidance(){ return stunAvoidance+GetEffStunAvoidance()
+tile.GetStunAvoidance()+PerkManager.GetUnitStunAvoidance(prefabID); }
public int GetStunDuration(){ return stunDuration+GetEffStunDuration()
+tile.GetStunDuration()+PerkManager.GetUnitStunDuration(prefabID); }   public
float GetSilentChance(){ return silentChance+GetEffSilentChance()+
tile.GetSilentChance()+PerkManager.GetUnitSilentChance(prefabID); }
public float GetSilentAvoidance(){ return silentAvoidance+
GetEffSilentAvoidance()+tile.GetSilentAvoidance()+PerkManager.GetUnitSilentAv
oidance(prefabID); }
public int GetSilentDuration(){ return
silentDuration+GetEffSilentDuration()+tile.GetSilentDuration()+PerkManager.Ge
tUnitSilentDuration(prefabID); }
      public float GetFlankingBonus(){ return flankingBonus+
GetEffFlankingBonus()+PerkManager.GetUnitFlankingBonus(prefabID); }
      public float GetFlankedModifier(){ return
flankedModifier+GetEffFlankedModifier()+PerkManager.GetUnitFlankedModifier(pr
efabID); }
            public float GetHPRatio(){
                  var full=GetFullHP();
                  return full<=0 ? 0 : HP/full;}
            public float GetAPRatio(){
                  var full=GetFullAP();
                  return full<=0 ? 0 : AP/full;}
            public int GetEffectiveMoveRange(){
                  if(movePerTurn==0) return 0;
                  float apCost=GetMoveAPCost();
                  int apAllowance=apCost==0 ? 999999 :
(int)Mathf.Abs(AP/apCost);
                  return Mathf.Min(GetMoveRange(), apAllowance); }
            public bool CanAttack(){
                  bool apFlag=GameControl.UseAPForAttack() ?
AP>=GetAttackAPCost() : true ;
                  return attackRemain>0 & GetAttackRange()>0 & !IsStunned() &
apFlag; }
            public bool CanMove(){ return moveRemain>0 &
GetEffectiveMoveRange()>0 & !IsStunned(); }
            public bool CanUseAbilities(){ return abilityRemain>0 &
!IsStunned() & !IsSilenced(); }
            //end get stats section
            //indicate the cover status to selected unit
            public CoverSystem._CoverType
coverStatus=CoverSystem._CoverType.None;
            public void UpdateCoverStatus(){
                  if(GameControl.GetSelectedUnit()==null)
coverStatus=CoverSystem._CoverType.None;
                  else{
                        if(GameControl.GetSelectedUnit().factionID==factionID)
coverStatus=CoverSystem._CoverType.None;
            else coverStatus=CoverSystem.GetCoverType(
GameControl.GetSelectedUnit().tile, tile);}}
            //these section are related to ability effects
            [Header("Misc")]
            public List<Effect> effectList=new List<Effect>();
            public List<Effect> GetEffectList(){ return effectList; }
            public Effect activeEffect=new Effect();
            public void ApplyEffect(Effect eff){
                  if(FactionManager.GetSelectedFactionID()!=factionID)
eff.duration+=1;
                  float HPVal=Random.Range(eff.HPMin, eff.HPMax);
```

```
                  if(HPVal<0) ApplyDamage(-
HPVal*DamageTable.GetModifier(armorType, eff.damageType));
                  else if(HPVal>0) RestoreHP(HPVal);
      AP=Mathf.Clamp(AP+Random.Range(eff.APMin, eff.APMax), 0, GetFullAP());
                  if(eff.duration<=0) return;
                  eff.Init();
                  effectList.Add(eff);
                  UpdateActiveEffect();
                  EffectTracker.Track(this);}
            public void IterateEffectDuration(){
                  bool changed=false;
                  for(int i=0; i<effectList.Count; i++){
                        effectList[i].Iterate();
                        if(effectList[i].Due()){
                              effectList.RemoveAt(i); i-=1;
                              changed=true;}}
                  if(changed){
                        if(effectList.Count>0) UpdateActiveEffect();
                        else{
                              activeEffect=new Effect();
                              EffectTracker.Untrack(this);}}}
            public void UpdateActiveEffect(){
                  activeEffect=new Effect();
                  for(int i=0; i<effectList.Count; i++){
            activeEffect.stun|=effectList[i].stun;
            activeEffect.silence|=effectList[i].silence;
            activeEffect.HP+=effectList[i].HP;
            activeEffect.AP+=effectList[i].AP;
            activeEffect.HPPerTurn+=effectList[i].HPPerTurn;
            activeEffect.APPerTurn+=effectList[i].APPerTurn;
            activeEffect.moveAPCost+=effectList[i].moveAPCost;
            activeEffect.attackAPCost+=effectList[i].attackAPCost;
            activeEffect.turnPriority+=effectList[i].turnPriority;
            activeEffect.moveRange+=effectList[i].moveRange;
            activeEffect.attackRange+=effectList[i].attackRange;
            activeEffect.sight+=effectList[i].sight;
            activeEffect.movePerTurn+=effectList[i].movePerTurn;
            activeEffect.attackPerTurn+=effectList[i].attackPerTurn;
            activeEffect.counterPerTurn+=effectList[i].counterPerTurn;
            activeEffect.damage+=effectList[i].damage;
            activeEffect.hitChance+=effectList[i].hitChance;
            activeEffect.dodgeChance+=effectList[i].dodgeChance;
            activeEffect.critChance+=effectList[i].critChance;
            activeEffect.critAvoidance+=effectList[i].critAvoidance;
            activeEffect.critMultiplier+=effectList[i].critMultiplier;
            activeEffect.stunChance+=effectList[i].stunChance;
            activeEffect.stunAvoidance+=effectList[i].stunAvoidance;
            activeEffect.stunDuration+=effectList[i].stunDuration;
            activeEffect.silentChance+=effectList[i].silentChance;
            activeEffect.silentAvoidance+=effectList[i].silentAvoidance;
            activeEffect.silentDuration+=effectList[i].silentDuration;
            activeEffect.flankingBonus+=effectList[i].flankingBonus;
            activeEffect.flankedModifier+=effectList[i].flankedModifier;}
            if(GameControl.GetSelectedUnit()==this && RequireReselect())
GameControl.ReselectUnit();}
            public bool RequireReselect(){
                  if(activeEffect.movePerTurn!=0) return true;
                  if(activeEffect.attackPerTurn!=0) return true;
                  if(activeEffect.moveRange!=0) return true;
                  if(activeEffect.attackRange!=0) return true;
                  if(activeEffect.sight!=0) return true;
                  return false;}
            float GetEffHPBuff(){ return activeEffect.HP; }
```

```
            float GetEffAPBuff(){ return activeEffect.AP; }
            float GetEffHPPerTurn(){ return activeEffect.HPPerTurn; }
            float GetEffAPPerTurn(){ return activeEffect.APPerTurn; }
            float GetEffMoveAPCost(){ return activeEffect.moveAPCost; }
            float GetEffAttackAPCost(){ return activeEffect.attackAPCost; }
            float GetEffTurnPriority(){ return activeEffect.turnPriority; }
            int GetEffMoveRange(){ return activeEffect.moveRange; }
            int GetEffAttackRange(){ return activeEffect.attackRange; }
            int GetEffSight(){ return activeEffect.sight; }
            int GetEffAttackPerTurn(){ return activeEffect.attackPerTurn; }
            int GetEffMovePerTurn(){ return activeEffect.movePerTurn; }
            int GetEffCounterPerTurn(){ return activeEffect.counterPerTurn; }
            float GetEffDamage(){ return activeEffect.damage; }
            float GetEffHitChance(){ return activeEffect.hitChance; }
            float GetEffDodgeChance(){ return activeEffect.dodgeChance; }
            float GetEffCritChance(){ return activeEffect.critChance; }
            float GetEffCritAvoidance(){ return activeEffect.critAvoidance; }
            float GetEffCritMultiplier(){ return activeEffect.critMultiplier;}
            float GetEffStunChance(){ return activeEffect.stunChance; }
            float GetEffStunAvoidance(){ return activeEffect.stunAvoidance; }
            int GetEffStunDuration(){ return activeEffect.stunDuration; }
            float GetEffSilentChance(){ return activeEffect.silentChance; }
            float GetEffSilentAvoidance(){ return
activeEffect.silentAvoidance; }
            int GetEffSilentDuration(){ return activeEffect.silentDuration; }
            float GetEffFlankingBonus(){ return activeEffect.flankingBonus; }
            float GetEffFlankedModifier(){ return
activeEffect.flankedModifier; }
            //end ability effect section and
            //these section are related to UnitAbilities
            public List<int> abilityIDList=new List<int>();
            public List<UnitAbility> abilityList=new List<UnitAbility>();
            //only used in runtime
            public List<UnitAbility> GetAbilityList(){ return abilityList; }
            public UnitAbility GetUnitAbility(int Index){ return
abilityList[Index]; }
            public void InitUnitAbility(){
      //get bonus abilityID from perk and add it to perkAbilityIDList
                  List<int>
perkAbilityIDList=PerkManager.GetUnitAbilityIDList(prefabID);
                  for(int i=0; i<perkAbilityIDList.Count; i++)
abilityIDList.Add(perkAbilityIDList[i]);
List<int> perkAbilityXIDList=PerkManager.GetUnitAbilityXIDList(prefabID);
                  for(int i=0; i<perkAbilityXIDList.Count; i++)
abilityIDList.Remove(perkAbilityXIDList[i]);abilityList=AbilityManagerUnit.Ge
tAbilityListBasedOnIDList(abilityIDList);
                  for(int i=0; i<abilityList.Count; i++)
abilityList[i].Init(this);}
            public void AddAbility(int abilityID, int index=-1){
                  UnitAbility
ability=AbilityManagerUnit.GetAbilityBasedOnID(abilityID);
                  if(ability==null) return;
                  ability.Init(this);
                  if(index>=0) index=Mathf.Min(index, abilityList.Count);
                  else index=abilityList.Count;
                  abilityIDList.Insert(index, abilityID);
                  abilityList.Insert(index, ability);}
            public void RemoveAbility(int index){
                  if(abilityIDList.Count>index) abilityIDList.RemoveAt(index);
                  if(abilityList.Count>index) abilityList.RemoveAt(index);}
            public string SelectAbility(int index){  //called from UI to
select an ability
                  AbilityManager.ExitAbilityTargetMode();
```

```
                  UnitAbility ability=abilityList[index];
                  string exception=ability.IsAvailable();
                  if(exception!="") return exception;
                  //requireTargetSelection=ability.requireTargetSelection;
            if(!ability.requireTargetSelection) ActivateAbility(tile, index);
                  else AbilityManagerUnit.ActivateTargetMode(tile,
abilityList[index], index, this.ActivateAbility, null);
                  return "";}
            //callback function for when a target tile has been selected for
current active ability
            private Tile abilityTargetedTile;
            public void ActivateAbility(Tile targetTile, int index){
                  if(targetTile==null) return;
                  UnitAbility ability=abilityList[index];
                  ability.factionID=factionID;
                  AP-=ability.GetCost();
                  ability.Use();
                  abilityTargetedTile=targetTile;
                  if(!ability.enableMoveAfterCast) moveRemain=0;
                  if(!ability.enableAttackAfterCast) attackRemain=0;
                  if(!ability.enableAbilityAfterCast) abilityRemain=0;
                  TurnControl.CheckPlayerMoveFlag();
                  if(ability.normalAttack){
                        OverlayManager.ClearSelection();
                        GameObject shootObj=ability.shootObject;
                        if(shootObj==null)
shootObj=GameControl.GetDefaultShootObject();
      if(onActionCamE!=null) onActionCamE(this, targetTile.GetPos(),
GameControl.GetAbilityActionCamFreq());
                        StartCoroutine(AttackRoutineAbility(targetTile,
targetTile.unit, shootObj, index));}
                  else{
                        float delay=AbilityHit(index);
                        StartCoroutine(DelayFinishAction(delay));}}
            //callback function when the shootObject of an ability which
require shoot hits it's target
            public float AbilityHit(int index){
                  if(Random.value>abilityList[index].chanceToHit){
                        abilityTargetedTile.GetPos();
      new TextOverlay(abilityTargetedTile.GetPos(), "missed", Color.white);
                  return 0}}
                  if(abilityList[index].applyToAllUnit)
abilityTargetedTile=null;
      AbilityManagerUnit.ApplyAbilityEffect(abilityTargetedTile,
abilityList[index], (int)abilityList[index].type, this);
                  return abilityList[index].effectDelayDuration;}
      //this is for ability teleport and spawnNewUnit to settle in new unit
            public void SetNewTile(Tile targetTile){
                  if(tile!=null) tile.unit=null;
                  tile=targetTile;targetTile.unit=this;
                  thisT.position=targetTile.GetPos();
                  FactionManager.UpdateHostileUnitTriggerStatus(this);
                  SetupFogOfWar();}
            public void Overwatch(Effect effect){
                  overwatching=true;
                  if(effect.duration==1)
                  effect.Init();effectList.Add(effect);
                  UpdateActiveEffect();
                  EffectTracker.Track(this);}//end UnitAbilities section
            [HideInInspector] public Transform thisT;
            [HideInInspector] public GameObject thisObj;
            void Awake(){
                  thisT=transform;
```

```
                  thisObj=gameObject;
                  for(int i=0; i<shootPointList.Count; i++){
                        if(shootPointList[i]==null){
                              shootPointList.RemoveAt(i);
                              i-=1;}}
                  if(shootPointList.Count==0) shootPointList.Add(thisT);
                  if(turretObject==null) turretObject=thisT;
                  SetDefaultRotation();}
            void Start () {
                  HP=GetFullHP();
                  if(GameControl.RestoreUnitAPOnTurn()) AP=GetFullAP();}
            public void PlaySelectAudio(){
                  if(unitAudio!=null) unitAudio.Select();}
            private bool rotating=false;
            private Quaternion facingRotation;
            public void Rotate(Quaternion rotation){
                  facingRotation=rotation;
                  if(!rotating) StartCoroutine(_Rotate());}
            IEnumerator _Rotate(){
                  rotating=true;
                  while(Quaternion.Angle(facingRotation,
thisT.rotation)>0.5f){
                        if(!TurnControl.ClearToProceed()) yield break;
                        thisT.rotation=Quaternion.Lerp(thisT.rotation,
facingRotation, Time.deltaTime*moveSpeed*2);
                        yield return null}
                  rotating=false;}
            public float GetMoveAPCostToTile(Tile targetTile){
                  List<Tile> path=AStar.SearchWalkableTile(tile, targetTile);
                  while(path.Count>GetEffectiveMoveRange())
path.RemoveAt(path.Count-1);
                  return GetMoveAPCost()*path.Count;}
            public List<Tile> GetPathForAI(Tile targetTile){
                  List<Tile> path=AStar.SearchWalkableTile(tile, targetTile);
                  while(path.Count>GetEffectiveMoveRange())
path.RemoveAt(path.Count-1);
                  return path;}
            public void Move(Tile targetTile){
                  if(moveRemain<=0) return;
                  moveRemain -=1;
                  Debug.Log("moving "+name+" to "+targetTile);
                  OverlayManager.ClearSelection();
                  TurnControl.CheckPlayerMoveFlag();
                  StartCoroutine(MoveRoutine(targetTile));
                  Defender(targetTile);}
            int def = 0;
            public MeshRenderer meshRenderer;
            public void Defender (Tile targetTile)
            {GridManager._GridColliderType colType =
GridManager._GridColliderType.Master;
                  string loadText = "";
                  if (colType == GridManager._GridColliderType.Master)
                  {loadText = "ScenePrefab/HexTile";
                        Transform tilePrefab = Resources.Load(loadText,
                        typeof(Transform)) as Transform;
                        def = 10;}}
            public IEnumerator MoveRoutine(Tile targetTile){
                  tile.unit=null;
                  GridManager.ClearAllTile();
                  List<Tile> path=AStar.SearchWalkableTile(tile, targetTile);
                  while(path.Count>GetEffectiveMoveRange())
path.RemoveAt(path.Count-1);
                  AP-=GetMoveAPCost()*path.Count;
```

```
      //smooth the path so the unit wont zig-zag along a diagonal direction
                  if(GridManager.GetTileType()==_TileType.Square &&
!GridManager.EnableDiagonalNeighbour()){
                        path.Insert(0, tile);
                        path=PathSmoothing.SmoothDiagonal(path);
                        path.RemoveAt(0);}
                  while(!TurnControl.ClearToProceed()) yield return null;
                  if(GameControl.EnableFogOfWar() && isAIUnit && path.Count>0
&& !tile.IsVisible()){
                        bool pathVisible=false;
                        for(int i=0; i<path.Count; i++){
                              if(path[i].IsVisible()){
                                    pathVisible=true;
                                    break;}}
                        if(!pathVisible){
                              thisT.position=path[path.Count-1].GetPos();
                              tile=path[path.Count-1];
                              tile.unit=this;
                              yield break;}}
                  TurnControl.ActionCommenced();
      }             if(path.Count!=0){
                        if(unitAnim!=null) unitAnim.Move();
                        if(unitAudio!=null) unitAudio.Move();}
                  while(path.Count>0){
                        while(true){
            float dist=Vector3.Distance(thisT.position, path[0].GetPos());
                        if(dist<0.05f) break;
      Quaternion wantedRot=Quaternion.LookRotation(path[0].GetPos()-
thisT.position);
      thisT.rotation=Quaternion.Slerp(thisT.rotation, wantedRot,
Time.deltaTime*moveSpeed*3);
      Vector3 dir=(path[0].GetPos()-thisT.position).normalized;
      thisT.Translate(dir*Mathf.Min(moveSpeed*Time.deltaTime, dist),
Space.World);
            yield return null;}
                        tile=path[0];
                        UpdateVisibility(path[0]);
                        FactionManager.UpdateHostileUnitTriggerStatus(this);
                        SetupFogOfWar();
                        int triggerCount=TriggerOverWatch();
                        if(triggerCount>0){
      while(!TurnControl.ClearToProceedFromOverWatch()) yield return null;
                        if(HP<=0){  //if the unit get destroyed there
and then by overwatch
                              StartCoroutine(Dead());
                              if(!isAIUnit)
TurnControl.SelectedUnitMoveDepleted();
                              TurnControl.ActionCompleted(0.25f);
                              yield break;}}
                        path.RemoveAt(0);}
                  if(unitAnim!=null) unitAnim.StopMove();
                  if(unitAudio!=null) unitAudio.StopMove();
                  tile.unit=this;
                  thisT.position=tile.GetPos();
                  tile.TriggerCollectible(this);
                  TurnControl.ActionCompleted(0.15f);
                  FinishAction();}
            public int TriggerOverWatch(){
                  List<Unit> unitList=FactionManager.GetAllUnit();
                  int triggerCount=0;
                  for(int i=0; i<unitList.Count; i++){
                        if(unitList[i]==this) continue;
                        if(unitList[i].CheckOverWatch(this)) triggerCount+=1;}
```

```
                return triggerCount;}
            public bool CheckOverWatch(Unit unit){
                if(unit==null) return false;
                if(!overwatching) return false;
                if(unit.factionID==factionID) return false;
                if(GridManager.GetDistance(unit.tile,
tile)>GetAttackRange()) return false;
                if(GameControl.EnableFogOfWar()){
            if(!FogOfWar.IsTileVisibleToUnit(unit.tile, this)) return false;}
                overwatching=false;
                AttackOverWatch(unit);
                return true;}//for ability
            public IEnumerator AttackRoutineAbility(Tile targetTile, Unit
targetUnit, GameObject shootObj, int abilityIndex){
                Debug.Log("AttackRoutineAbility");
                while(!TurnControl.ClearToProceed()) yield return null;
                TurnControl.ActionCommenced();
                aiming=true;
                while(!Aiming(targetTile, targetUnit,
shootObj.GetComponent<ShootObject>())) yield return null;
                aiming=false;//play animation
                float animDelay=PlayAttackAnimationNAudio(false);
                if(animDelay>0) yield return new WaitForSeconds(animDelay);
                waitingForAbilityHit=true;
                Vector3 targetPos=targetUnit==null ? targetTile.GetPos() :
targetUnit.GetTargetT().position;
                float hitThreshold=targetUnit==null ? 0.2f : Mathf.Max(0.2f,
targetUnit.hitThreshold);
                for(int i=0; i<shootPointList.Count; i++){
                    Shoot(shootObj, shootPointList[i], targetPos,
hitThreshold, this.AbilityHitCallback);
                    if(delayBetweenShootPoint>0) yield return new
WaitForSeconds(delayBetweenShootPoint);}
                while(waitingForAbilityHit) yield return null;
                float abilityEffectDelay=AbilityHit(abilityIndex);
                TurnControl.ActionCompleted(abilityEffectDelay);
                FinishAction();
                if(turretObject!=null || barrelObject!=null){
while(!RotateTurretToOrigin() && !aiming) yield return null; }}
            private bool waitingForAbilityHit=false;
            public void AbilityHitCallback(){ waitingForAbilityHit=false; }
            public void AttackOverWatch(Unit targetUnit){
                TurnControl.ActionCommenced();
                GameControl.DisplayMessage("overwatch");
                if(onActionCamE!=null) onActionCamE(this,
targetUnit.tile.GetPos(), GameControl.GetAttackActionCamFreq());
                StartCoroutine(AttackRoutine(targetUnit, false, true));}
            public void Attack(Unit targetUnit, bool isCounter=false){
                if(!isCounter){
                    if(attackRemain==0) return;
                    if(AP<GetAttackAPCost()) return;
                    if(!GameControl.EnableActionAfterAttack()){
                        moveRemain=0;
                        abilityRemain=0;}
                    if(onActionCamE!=null) onActionCamE(this,
targetUnit.tile.GetPos(), GameControl.GetAttackActionCamFreq());}
                attackRemain-=1;
                AP-=GetAttackAPCost();
                OverlayManager.ClearSelection();
                TurnControl.CheckPlayerMoveFlag();
                StartCoroutine(AttackRoutine(targetUnit, isCounter));}
            public IEnumerator AttackRoutine(Unit targetUnit, bool
isCounter=false, bool isOverWatch=false){
```

```
                if(!isCounter && !isOverWatch){
                while(!TurnControl.ClearToProceed()) yield return null;
                      TurnControl.ActionCommenced();}
                bool isMelee=false;
                if(isHybridUnit && GridManager.GetDistance(tile,
targetUnit.tile)<=meleeRange) isMelee=true;
                AttackInstance attInstance=GetAttackInfo(targetUnit,
isCounter, isOverWatch, isMelee);
                ShootObject so=!isMelee ? shootObject : shootObjectMelee ;
                GameObject shootObj=so!=null ? so.gameObject : null ;
                if(shootObj==null)
shootObj=GameControl.GetDefaultShootObject();
                aiming=true;
                while(!Aiming(targetUnit.tile, targetUnit,
shootObj.GetComponent<ShootObject>())) yield return null;
                aiming=false;//play animation
                float
animDelay=PlayAttackAnimationNAudio(attInstance.isMelee);
                if(animDelay>0) yield return new WaitForSeconds(animDelay);
                waitingForAttackHit=true;//shoot
                for(int i=0; i<shootPointList.Count; i++){
                      Shoot(shootObj, shootPointList[i],
targetUnit.GetTargetT().position, Mathf.Max(0.2f, targetUnit.hitThreshold),
this.AttackHitCallback);
                      if(delayBetweenShootPoint>0) yield return new
WaitForSeconds(delayBetweenShootPoint);}
                while(waitingForAttackHit) yield return null;
                targetUnit.ApplyAttack(attInstance, isOverWatch);    //pass
overwatch flag to stop the unit from being destroyed instantly
//multiple attack may happen at once, we need to wait until all attack has
been completed
//after all the attack done, destoy would be call when MoveRoutine is resumed
//if this is a normal attack and the target can counter attack, call the
attack function
//the responsible to call TurnControl.ActionCompleted() is then passed on the
the countner attacking unit
                if(!isCounter && !isOverWatch && !attInstance.destroyed &&
targetUnit.CanCounter(this)) targetUnit.Attack(this, true);
                else{
//in case we counter attack the selected player unit and destroy it
           if(isCounter && attInstance.destroyed && !targetUnit.isAIUnit){
                      TurnControl.SelectedUnitMoveDepleted();}
                      TurnControl.ActionCompleted(attInstance.destroyed ?
targetUnit.GetDestroyDuration() : 0.15f);}
                //if this is a normal attack, we need to finish the unit
action, to reselect and so on
                if(!isCounter && !isOverWatch){
                      while(!TurnControl.ClearToProceed()) yield return
null; //wait a bit in case the target unit is still counter attackinging
                      FinishAction();}
                //if(turretObject!=null && turretObject!=thisT){
while(!RotateTurretToOrigin() && !aiming) yield return null; }
                if(turretObject!=null || barrelObject!=null){
while(!RotateTurretToOrigin() && !aiming) yield return null; }}
           private bool waitingForAttackHit=false;
           public void AttackHitCallback(){ waitingForAttackHit=false; }
           public AttackInstance GetAttackInfo(Unit targetUnit, bool
isCounter=false, bool isOverWatch=false, bool isMelee=false){
                AttackInstance attInstance=new AttackInstance(this,
targetUnit, isCounter, isOverWatch, isMelee);
                attInstance.Process();
                return attInstance;}
           public float PlayAttackAnimationNAudio(bool isMelee=false){
```

```
                  float delay=0;
                  if(unitAnim!=null){
                        if(!isMelee) unitAnim.Attack();
                        else unitAnim.AttackMelee();
            delay=isMelee ? unitAnim.attackDelayMelee: unitAnim.attackDelay;}
                  if(unitAudio!=null){
                        if(!isMelee) unitAudio.Attack();
                        else unitAudio.AttackMelee();}
                  return delay;}
            public float GetDestroyDuration(){
                  Debug.Log("fix this");
                  return 1.5f;}
            public bool rotateTurretOnly=false;
            public bool rotateTurretAimInXAxis=true;
            public Transform turretObject;
            public Transform barrelObject;
            public ShootObject shootObject;
            public ShootObject shootObjectMelee;
            public List<Transform> shootPointList=new List<Transform>();
            public float delayBetweenShootPoint=0;
            private bool aiming=false;    //to avoid aiming and rotate back to
origin at the same time
            bool Aiming(Tile tile, Unit targetUnit=null, ShootObject so=null){
                  Quaternion wantedRotY=Quaternion.LookRotation(tile.GetPos()-
thisT.position);
                  Vector3 targetPos=(targetUnit==null) ? tile.GetPos() :
targetUnit.GetTargetT().position;
                  //rotate body
                  if(!rotateTurretOnly && thisT!=turretObject){
                        thisT.rotation=Quaternion.Slerp(thisT.rotation,
wantedRotY, Time.deltaTime*rotateSpeed);}
                  if(rotateTurretAimInXAxis){
                        float offsetX=so!=null ?
so.GetProjectileShootAngle(shootPointList[0].position, targetPos) : 0;
                        if(barrelObject!=null){
      turretObject.rotation=Quaternion.Slerp(turretObject.rotation,
wantedRotY, Time.deltaTime*rotateSpeed);
Quaternion wantedRot2=Quaternion.LookRotation(targetPos-
barrelObject.position);
Quaternion barrelRot=barrelDefaultRot*
Quaternion.Euler(wantedRot2.eulerAngles.x-offsetX, 0, 0);
      barrelObject.localRotation=Quaternion.Slerp(barrelObject.localRotation,
barrelRot, Time.deltaTime*rotateSpeed);
float angle1=Quaternion.Angle(turretObject.rotation, wantedRotY);
float angle2=Quaternion.Angle(barrelObject.localRotation, barrelRot);
                              if(angle1<0.5f && angle2<0.5f) return true;}
                        else{
                              Quaternion
wantedRot=Quaternion.LookRotation(targetPos-
turretObject.position)*Quaternion.Euler(-offsetX, 0, 0);
      turretObject.rotation=Quaternion.Slerp(turretObject.rotation, wantedRot,
Time.deltaTime*rotateSpeed);
                              float
angle=Quaternion.Angle(turretObject.rotation, wantedRot);
                              if(angle<.5f) return true;}}
                  else{
      turretObject.rotation=Quaternion.Slerp(turretObject.rotation,
wantedRotY, Time.deltaTime*rotateSpeed);
                        if(Mathf.Abs(turretObject.rotation.eulerAngles.y-
wantedRotY.eulerAngles.y)<1f) return true;}
                  return false;}
bool RotateTurretToOrigin(){
                  float angle=0;
```

```
                if(turretObject!=null && turretObject!=thisT){
      turretObject.localRotation=Quaternion.Lerp(turretObject.localRotation,
turretDefaultRot, Time.deltaTime*rotateSpeed);
      angle=Quaternion.Angle(turretObject.localRotation, turretDefaultRot);}
                if(barrelObject!=null){
      barrelObject.localRotation=Quaternion.Slerp(barrelObject.localRotation,
barrelDefaultRot, Time.deltaTime*rotateSpeed);
                      float
angleAlt=Quaternion.Angle(barrelObject.localRotation, barrelDefaultRot);
                      return (angle>1 || angleAlt>1) ? false : true ;}
                else return angle>1 ? false : true ;}
            private Quaternion turretDefaultRot;
            private Quaternion barrelDefaultRot;
            void SetDefaultRotation(){
                turretDefaultRot=turretObject.localRotation;
                if(barrelObject!=null)
barrelDefaultRot=barrelObject.localRotation;}
            private void Shoot(GameObject shootObj, Transform sp, Vector3
targetPos, float hitTH, ShootObject.HitCallback callback){
                GameObject sObjInstance=(GameObject)Instantiate(shootObj,
sp.position, sp.rotation);
                ShootObject
soInstance=sObjInstance.GetComponent<ShootObject>();
                soInstance.Shoot(targetPos, hitTH, callback);}
            IEnumerator DelayFinishAction(float delay){
                yield return new WaitForSeconds(delay);
                FinishAction();}
            void FinishAction(){
                if(isAIUnit) return;
            //check hp in case unit trigger a collectible and killed itself
                if(IsAllActionCompleted() || HP<=0){
                      FactionManager.UnitMoveDepleted(this);
                      TurnControl.SelectedUnitMoveDepleted();}
                else GameControl.ReselectUnit();}
            public Effect GetStunEffect(){
                stunEffect=new Effect(); stunEffect.name="Stun";
                stunEffect.desp="Stunned by "+unitName+"'s attack";
                stunEffect.duration=stunDuration;
                return stunEffect;}
            public Effect GetSilentEffect(){
                silentEffect=new Effect();silentEffect.name="Silent";
                silentEffect.desp="Silenced by "+unitName+"'s attack";
                silentEffect.duration=silentDuration;
                return silentEffect;}
            public void ApplyAttack(AttackInstance attInstance, bool
dontDestroyUnit=false){
                attInstance.srcUnit.lastTarget=this;
                lastAttacker=attInstance.srcUnit;
                if(isAIUnit && !trigger) trigger=true;
                if(attInstance.missed){
            new TextOverlay(GetTargetT().position, "missed", Color.white);
                      return;}
                if(unitAudio!=null) unitAudio.Hit();
                if(unitAnim!=null) unitAnim.Hit();
                ApplyDamage(attInstance.damage, attInstance.critical,
dontDestroyUnit);
                if(attInstance.stunned)
ApplyEffect(attInstance.srcUnit.GetStunEffect());
                if(attInstance.silenced)
ApplyEffect(attInstance.srcUnit.GetSilentEffect());}
      public void ApplyDamage(float dmg, bool critical=false, bool
dontDestroyUnit=false){
```

```
                  if(!critical) new TextOverlay(GetTargetT().position,
dmg.ToString("f0"), Color.white);
                  else new TextOverlay(GetTargetT().position,
dmg.ToString("f0")+" Critical!", new Color(1f, .6f, 0, 1f));
                  HP-=dmg;
                  if(HP<=0){
                        HP=0;
                        if(!dontDestroyUnit) StartCoroutine(Dead());}}
            public GameObject destroyEffectObj;
            public bool autoDestroyEffect=false;
            public float destroyEffectDuration=1.5f;
            IEnumerator Dead(){
                  if(destroyEffectObj!=null){
                        if(!autoDestroyEffect)
ObjectPoolManager.Spawn(destroyEffectObj, GetTargetT().position,
Quaternion.identity);
                  else ObjectPoolManager.Spawn(destroyEffectObj,
GetTargetT().position, Quaternion.identity, destroyEffectDuration);}
                  float delay=0;
                  if(unitAudio!=null) delay=unitAudio.Destroy();
                  if(unitAnim!=null) delay=Mathf.Max(delay,
unitAnim.Destroy());
                  if(isAIUnit) TBTK.OnAIUnitDestroyed(this);
                  else TBTK.OnPlayerUnitDestroyed(this);
                  FactionManager.OnUnitDestroyed(this);
                  GridManager.OnUnitDestroyed(this);
      TurnControl.OnUnitDestroyed();     //to track cd on ability and effect
                  ClearVisibleTile();
                  tile.unit=null;
                  yield return new WaitForSeconds(delay);
                  Destroy(thisObj); }
            public void RestoreHP(float val){
                  HP=Mathf.Min(HP+val, GetFullHP());
new TextOverlay(GetTargetT().position, val.ToString("f0"), Color.green);}
            //called when a unit just reach it's turn
            public bool NewTurn(){
                  //Debug.Log("ResetUnitTurnData");
                  moveRemain=GetMovePerTurn();
                  attackRemain=GetAttackPerTurn();
                  counterRemain=GetCounterPerTurn();
                  abilityRemain=abilityPerTurn;
                  //disableAbilities=0;
                  if(moveRange==0) moveRemain=0;
                  if(attackRange==0) attackRemain=0;
                  if(GameControl.RestoreUnitAPOnTurn()) AP=GetFullAP();
                  else AP=Mathf.Min(AP+GetAPPerTurn(), GetFullAP());
                  HP=Mathf.Min(HP+GetHPPerTurn(), GetFullHP());
                  if(HP<=0){
                        StartCoroutine(Dead());
                        return false;}
                  return true;}
            public bool CanCounter(Unit unit){
                  if(!GameControl.EnableCounter()) return false;
                  if(IsStunned()) return false;
                  if(counterRemain<=0) return false;
                  if(GetCounterAPCost()>AP) return false;
                  float dist=GridManager.GetDistance(unit.tile, tile);
                  if(dist>GetAttackRange()) return false;
                  if(GameControl.EnableFogOfWar()){
                        if(requireDirectLOSToAttack &&
!FogOfWar.InLOS(unit.tile, tile)) return false;
                        if(!FogOfWar.IsTileVisibleToFaction(unit.tile,
factionID)) return false;}
```

```
                  return true;}
            public bool IsAllActionCompleted(){
                  if(IsStunned()) return true;
                  if(attackRemain>0 && AP>=GetAttackAPCost()) return false;
                  if(moveRemain>0 && AP>=GetMoveAPCost()) return false;
                  //if(CanUseAbilities()) return false;
                  return true;}
            //these section are related to FogOfWar
            [HideInInspector] private List<Tile> visibleTileList=new
List<Tile>();     //a list of tile visible to the unit
            //called whenever the unit moved into a new tile
            //when ignoreFaction is set to true,  even AI faction will run the
function, this is for optimization since the function will be called in
MoveRoutine very often
            public void SetupFogOfWar(bool ignoreFaction=false){
                  if(!GameControl.EnableFogOfWar()) return;
                  if(!ignoreFaction && isAIUnit) return;
                  StartCoroutine(FogOfWarNewTile(ignoreFaction));}
            //add new tiles within sight to visibleTileList and remove those
that are out of sight
            public IEnumerator FogOfWarNewTile(bool ignoreFaction){
                  List<Tile> newList=GridManager.GetTilesWithinDistance(tile,
sight, false);
                  newList.Add(tile);
                  ClearVisibleTile(newList);
                  for(int i=0; i<newList.Count; i++){
                        if(!visibleTileList.Contains(newList[i])){

      newList[i].SetVisible(FogOfWar.CheckTileVisibility(newList[i]));}}
                  visibleTileList=newList;
                  yield return null;}
            //set visible tile that becomes invisible to invisible
            public void ClearVisibleTile(List<Tile> newList=null){
                  for(int i=0; i<visibleTileList.Count; i++){
                        if(visibleTileList[i]==tile)
visibleTileList[i].SetVisible(true);

                        bool
flag=FogOfWar.CheckTileVisibility(visibleTileList[i]);
                        visibleTileList[i].SetVisible(flag);
            //called just before a unit start moving into a new tile, for AI
unit to show/hide itself as it move in/out of fog-of-war
      //also called when a unit is just been placed on a grid in mid-game
            public void UpdateVisibility(Tile newTile=null){
                  if(!GameControl.EnableFogOfWar()) return;
                  if(!isAIUnit) return;
                  if(newTile==null) newTile=tile;
                  if(newTile.IsVisible()){
                        thisObj.layer=TBTK.GetLayerUnit();
            Utilities.SetLayerRecursively(thisT, TBTK.GetLayerUnit());}
                  else{
                        thisObj.layer=TBTK.GetLayerUnitInvisible();
                        Utilities.SetLayerRecursively(thisT,
TBTK.GetLayerUnitInvisible());}}
      //end FogOfWar section
            [HideInInspector] private UnitAudio unitAudio;
            public void SetAudio(UnitAudio unitAudioInstance){
unitAudio=unitAudioInstance; }
            [HideInInspector] private UnitAnimation unitAnim;
            public void SetAnimation(UnitAnimation unitAnimInstance){
unitAnim=unitAnimInstance; }
            public void DisableAnimation(){ unitAnim=null; }}}
```