Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра_____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти - другий (магістерський)

Дослідження оптимізаційних моделей та методів з метою створення розвинутого інтелекту для комп'ютерних ігор в жанрі "Tower Defense"
(тема)

Виконав: студент 2 курсу, групи ІПЗм-18-2___
_____Топчій О.Д._____
(прізвище, ініціали)

спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

Освітньо-наукової програми___
(тип програми)

___Інженерія програмного забезпечення
(повна назва освітньої програми)

Керівник _____доц. Мазурова О.О._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2020 р.

Харківський національний університет радіоелектроніки

Факультет_Комп’ютерних наук_____

Кафедра_Програмної інженерії_____

Рівень вищої освіти - другий (магістерський)

Спеціальність _121-Інженерія програмного забезпечення_____
(код і повна назва)

Тип програми освітньо-наукова програма_____

Освітня програма_Інженерія програмного забезпечення_____

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові _Топчію Олександру Дмитровичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи _Дослідження оптимізаційних моделей та методів з метою створення розвинутого інтелекту для комп'ютерних ігор в жанрі "Tower Defense"_____

затверджена наказом університету від “_27_” __03__2020 р № _473Ст_
заповнюється вручну після отримання наказу

2. Термін подання студентом роботи до екзаменаційної комісії
11 травня 2020 р.

3. Вихідні дані до роботи *електронні ресурси за обраною тематикою, вимоги до функціональності програми, методи теорії прийняття рішень, платформа розробки Unity та .NET, середовища розробки Jetbrains Rider*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, дослідження оптимізаційних моделей теорії прийняття рішень, методи згорток із теорії корисності, методи лінійного програмування із теорії дослідження операцій*

5 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	доц. Мазурова О.О.		11.05.20

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз проблемної області	27.01.20 – 10.02.20	виконано
2.	Постановка задачі	11.02.20 – 13.02.20	виконано
3.	Дослідження існуючих методів та моделей	14.02.20 – 19.02.20	виконано
4.	Розробка математичних моделей та алгоритмів	20.02.20 – 3.03.20	виконано
5.	Розробка бази даних та архітектури	4.03.20 – 7.03.20	виконано
6.	Програмна реалізація	8.03.20 – 25.03.20	виконано
7.	Експериментальне дослідження розроблених алгоритмів	26.03.20 – 30.03.20	виконано
8.	Підготовка пояснювальної записки	1.04.20 – 29.04.20	виконано
9.	Підготовка презентації та доповіді	30.04.20 – 3.05.20	виконано
10.	Попередній захист	9.05.20	виконано
11.	Нормоконтроль, рецензування	4.05.20 – 10.05.20	виконано
12.	Занесення диплома в електронний архів	11.05.20	виконано
13.	Допуск до захисту у зав. кафедри	11.05.20	виконано

Дата видачі завдання _____ _____2020 р.

Студент ___ _____ _____Топчій О.Д._____
(підпис)

Керівник роботи _____ _____доц. Мазурова О.О._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить: 106 стор., 12 рис., 8 таблиць, 23 джерела.

ІГРОВА СИСТЕМА, ШТУЧНИЙ ІНТЕЛЕКТ, TOWER DEFENSE, ЛІНІЙНЕ ПРОГРАМУВАННЯ, МЕТОД ЗГОРТКИ, ЖАДІБНИЙ АЛГОРИТМ, ОПТИМІЗАЦІЙНА ЗАДАЧА, МАТЕМАТИЧНА МОДЕЛЬ, C#, ASP.NET, WEBAPI, UNITY 3D, MYSQL.

Метою роботи є дослідження оптимізаційних моделей теорії прийняття рішень для розробки штучного інтелекту для керування стратегією комп’ютера в грі в жанрі «Tower Defense» та розробка ігрової системи за допомогою розроблених моделей.

Методи проектування та розробки базуються на технології ASP.NET Core WebApi, ігровому рушії Unity 3D, базі даних MySQL, шини повідомлень Rabbit MQ.

В результаті роботи досліджено оптимізаційні моделі та методи, на базі яких розроблено математичні моделі штучного інтелекту для ігрової системи в жанрі Tower Defense. Програмно реалізовано ігрову систему та проведено експериментальне дослідження розроблених моделей та алгоритмів.

GAMING SYSTEM, ARTIFICIAL INTELLIGENCE, TOWER DEFENSE, LINEAR PROGRAMMING, CONVOLUTION METHOD, GREEDY ALGORITHM, OPTIMIZATION PROBLEM, MATHEMATICAL MODEL, C#, ASP.NET, WEBAPI, UNITY 3D, MYSQL.

The purpose of this work is to investigate optimization models of decision theory for the development of artificial intelligence to manage the computer strategy in the game in the Tower Defense genre and to develop the game system with the help of the developed models.

Design and development methods are based on ASP.NET Core WebApi technology, Unity 3D game engine, MySQL database, Rabbit MQ message bus.

As a result, optimizational models and methods are investigated. On this basis have been developed mathematical models of artificial intelligence for the gaming system in the Tower Defense genre. A game system was implemented, an experimental research of developed models was conducted.

ЗМІСТ

Вступ……..…….....……...…..…….8
1 Аналіз проблемної області та постановка задачі……..…….……..…...10
1.1 Аналіз проблемної області розробки ігрових систем…..……...….10
1.2 Аналіз існуючих аналогів……...…….....….....…..15
1.3 Постановка задачі……...….....…...……..18
2 Перелік вимог до програмної системи.....20
2.1 Концепт ігрового додатку ……...…….....….20
2.2 Функціональні вимоги……....…….....…....…..21
2.3 Нефункціональні вимоги …….....….....……...23
3 Опис дослідження та прийнятих проектних рішень.....…….25
3.1 Розробка механіки та принципів гри……...……..….....……...25
3.2 Аналіз оптимізаційних моделей та методів з метою створення штучного інтелекту..…….…….……...26
3.3 Моделювання штучного інтелекту на базі теорії корисності..…….……...…...…….32
3.4 Моделювання штучного інтелекту на базі задач лінійного програмування ……...…….……...……..…..….....….35
3.5 Розробка бази даних для зберігання інформації з розроблених моделей …….……42
3.6 Розробка алгоритмів для вирішення промодельованих оптимізаційних задач.....…….…..…….46
3.7 Проектування архітектури програмного забезпечення.....50
4 Опис програмної реалізації……..…..…….....…….....…...53
4.1 Реалізація серверної частини ……...…….....…….....…..53
4.2 Реалізація клієнтської частини……...……...…….....…...…....….55
4.3 Реалізація фізичної моделі бази даних……...…….....…….....….57

5 Експериментальне дослідження розроблених моделей та методів.....58
5.1 Планування експериментів.....58
5.2 Експериментальне дослідження.....60
Висновки…….....….65
Перелік джерел посилання.....…….....…….67
Додаток А Слайди презентації…….……..…….…….69
Додаток Б Лістинг коду…….……...81
Додаток В Тези доповіді на молодіжному форумі «Радіоелекторніка та молодь у XXI ст»..……..……..……...……...92
Додаток Г Стаття для науково-технічного семінару «CMIS-2020»…….….93

ВСТУП

Складно уявити сучасний світ без комп’ютерних ігор, що зайняли значну частину ринку розваг та дозвілля. Серед різноманіття видів відеоігор насамперед виділяються стратегії у реальному часі. В таких стратегіях перед користувачем ставляться доволі складні задачі, тому що він повинен своєчасно реагуючі на ті чи інші події продумувати свою стратегію на гру для того, щоб досягти позитивного результату. Одним з найпопулярніших підвидів таких стратегій є стратегії у жанрі Tower Defence [1].

На даний момент існує велика кількість ігор даного жанру, проте більша частина з них являються іграми для одного користувача, та не підтримують можливості грати з іншими людьми, та змагатися з ними, граючи при цьому за різні сторони. Крім того, більшість ігор в даному жанрі є іграми для мобільних телефонів, тому користувачі потребують сучасну версію гри в представленому жанрі для персональних комп’ютерів.

На даний момент основні ігри в даному жанрі, що представлені на ринку, пропонують гравцям достатньо прямолінійні дії з боку штучного супротивника, які як правило є запрограмованими на певні рішення зазделегіть, та не реагують на дії гравця у реальному часі. Гравці потребують супротивника, який би міг приймати різноманітні рішення різного рівня складності, непередбачуваності. Тобто, такі ігри потребують використання з боку комп’ютеру розвинутого штучного інтелекту.

Метою роботи є дослідження оптимізаційних моделей теорії прийняття рішень для розробки штучного інтелекту для керування стратегією комп’ютера в грі в жанрі «Tower Defense» та розробка ігрової системи за допомогою розроблених моделей. Також для реалізації поведінки штучного інтелекту гри з боку монстрів або захисників необхідно було розробити математичні моделі оптимізаційних задач про призначення юнітів на шляхи.

Також для розробки алгоритмів переміщення ігрових об’єктів повинна бути розроблена графова модель шляхів на полі. Треба проаналізувати та обрати за основу існуючі алгоритми пошуку найкоротших шляхів на карті та методи багатокрітерійного аналізу для знаходження оптимальних шляхів.

В ході атестаційної роботи магістра було:

– розроблено механіку та принципи гри в жанрі "Tower Defence";

– проведено аналіз опитізаційних моделей та методів з метою створення штучного інтелекту;

– розроблено математичні моделі поведінки ШІ у грі в жанрі Tower Defense на базі оптимізаційних задач теорії прийняття рішень;

– розроблено алгоритми вирішення оптимізаційних задач на базі розроблених моделей;

– виконано програмну реалізацію ігрової системи;

– були спланувані та проведені експериментальні дослідження розроблених методів та моделей;

– за результатами експериментів були розроблені рекомендації стосовно використання обраного підходу для створення розвинутого ШІ в іграх у жанрі.

За результатами атестаційної роботи магістра було розроблено презентацію (див. додаток А). Лістинг коду наведено в додатку Б. За результатами роботи створено тези доповіді на молодіжному форумі «Радіоелекторніка та молодь у XXI ст.» (див. додаток В) та доповідь посіла 3-те місце на форумі.

Була написана та подана до опублікування до науково-технічного семінару «The Third International Workshop on Computer Modeling and Intelligent Systems (CMIS-2020)» стаття «A Study of Optimization Models for Creation of Advanced Artificial Intelligence for the Computer Game in the Tower Defense Genre», що наведена у додатку Г. Усі матеріали наведено на диску.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної області розробки ігрових систем

У сучасному світі індустрія розробки ігор займає значну частину індустрії інформаційних технологій. Комп'ютерні відеоігри набули великої популярності в останні роки та зайняли значне місце на ринку розваг та дозвілля. Індустрія комп'ютерних ігор (також індустрія інтерактивних розваг) - сектор економіки, пов'язаний з розробкою, просуванням і продажем комп'ютерних ігор. В неї входить велика кількість спеціальностей, за якими працюють тисячі людей по всьому світу.

Створення повноцінної комп'ютерної гри не може бути виконано без використання ігрового движку. Ігровий рушій - основний програмний компонент комп'ютерних відеоігор або інших інтерактивних додатків з графікою, яка обробляється з рендерингом в реальному часі. Він надає основні технологічні можливості, спрощує розробку і часто дає грі можливість запускатися на декількох платформах.

Більшість великих компаній створюють власні ігрові движки для своїх ігор, які оптимізовані та спеціалізовані для їх потреб. Проте розробники мають змогу використовувати загально доступні движки, серед яких є як платні так і безплатні версії.

В основному комп'ютерні ігри класифікуються за жанрами, а також за кількістю гравців. Так як, критерії приналежності гри до того чи іншого жанру не визначені однозначно, класифікація комп'ютерних ігор недостатньо систематизована, і в різних джерелах дані про жанр конкретного проекту можуть розрізнятися.

Одним з основних жанрів комп'ютерних ігр є жанр стратегій. Стратегії (англ. Strategy) – комп'ютерні ігри, що вимагають від гравців планування і вироблення певної стратегії для досягнення якоїсь конкретної мети, наприклад,

перемоги у військовій операції. Гравець управляє не одним персонажем, а цілою кількістю [2].

Розрізняють покрокові стратегічні ігри (Turn-Based Strategy, TBS), в яких гравці по черзі роблять ходи, і кожному гравцеві відводиться необмежений або обмежений (в залежності від типу гри) час на хід, та стратегічні ігри в реальному часі (Real Time Strategy, RTS), в яких всі гравці виконують свої дії одночасно, і хід часу не переривається.

Більшість стратегічних комп'ютерних ігор тим або іншим чином пов'язана з картою. Карта відіграє значну роль в ігровому процесі, так як основні дії протекають в тому або іншому місці карти. Відносно карти задаються та визначаються позиції ігрових об'єктів.

Серед всього різноманіття комп’ютерних ігор доволі високий інтерес визивають стратегічні ігри у реальному часі. В таких стратегіях перед гравцем ставляться доволі складні задачі: гравець повинен своєчасно реагуючі на ті чи інші події, продумувати свою стратегію для того, щоб обіграти свого супротивника. Супротивником людини в таких іграх виступає штучний інтелект, який для забезпечення певного інтересу повинен реагувати на дії гравця в реальному часі, приймати різноманітні та непередбачуванні рішення.

Одним з найпопулярніших підвидів стратегічних ігор є ігри-стратегії у жанрі Tower Defense. Для ігр в цьому жанрі присутні такі загальні риси:

– є мапа, на який присутні одна, чи декілька точок входу та одна чи декілька точок виходу; між цими точками пролягають звивисті шляхи;

– гравці оперують на карті ігровими об’єктами, які можна розділити на два класи: нападники (наприклад, монстри) та захисники;

– основною метою нападників є пройти від входу до виходу (або до цілі) по шляхах на мапі;

– основною метою захисників є створення перешкод на шляху нападників, за допомогою будування атакуючих веж уздовж шляхів; ігрові об’єкти (вежі та нападники) відрізняються за своїми характеристиками.

Ігрові персонажі, що переміщюються по карті повинні шукати оптімальні шляхи. Можуть бути різні критерії оптимальності шляху. Так одним із крітерієв є довжина шляху. Проте у рамках стратегічної гри довжина шляху далеко не завжди є оптимальним, або навіть значущим критерієм. Тому оптімальність шляху краще визначати на основі багатьох критеріїв з різними ступінями значущостю.

На данний момент серед розробників комп’ютерних ігор остаточно не вирішено питання, який підхід до розробки штучного інтелекту є більш ефективним: заздалегідь запрограмовані стратегії, тобто відсутність інтелекту, або розвинутий штучний інтелект (ШІ), де дії ШІ залежать від конкретних дій гравця [3, 4].

Дослідники поділяють ШІ в іграх у загальному випадку на три категорії: базовий ШІ, високореактивний ШІ (або середій), розвинутий ШІ (РШІ) [5]. Ігри з базовим ШІ є найменьш вимогливими до обчислюваних ресурсів комп’ютера, проте з точки зору ігрового досвіду гравця вони не є дуже інтерактивними та ніяк не реагують на дії гравця.

Ігри з високореактивним ШІ потребують достатньо обчислюваних ресурсів, проте вони задовільняють варіативний ігровий досвід, та адаптуються до дій ігрока у процесі геймплею. Ігри із високорозвиненим ШІ потребують найбільше обчислюваних потужностей, але й в процесі ігри вони використовують більш прораховані, оптимальні стратегії, навчаються та приспособлюються під стиль гравця [6].

Потрібно відзначити, що й досі тривають дебати про те, у яких іграх які види ШІ використовувати, як залежить популярність ігор з різним інтелектом серед гравців.

Деякі дослідники вважають, що використання РШІ далеко не завжди є доцільним, та у деяких випадках доцільніше використання високореактивного ШІ, або навіть базового.

Деякі статистики показують, що ігри з розвинутим ШІ не мають достатьої популярності серед користувачів [7] у зв’язку з використанням складних

математичних апаратів, що призводять до значних обчислювальних затрат. На цьому фоні більш популярними є ігри із базовим ШІ.

Приверженці розвинутого штучного інтелекту пропонують використовувати для його створення різноманітні математичні моделі [8], нейронні мережі [9] та диференціальні рівняння [10].

На сьогоднішній день існують різноманітні дослідженя у області розробки математичних моделей штучного інтелекту для ігор в жанрі Tower Defense.

Проте здебільшого у іграх цього жанру переважає заздалегіть запрограмована стратегія, тобто використовується нерозвинутий штучний інтелект. Це обгрунтовується в першу чергу економією ресурсів комп’ютера та тим, що як правило в таких іграх непотрібна велика варіотивність з боку комп’ютера.

Деякі дослідники розглядають РШІ, як інтелект, що здатний приймати рішення [11]. Отже, можна підійти до моделювання РШІ шляхом використання моделей теорії прийняття рішень (ТПР) [12].

Серед різноманіття підходів ТПР насамперед потрібно визначити ресурсоекономний багатокрітеріальний аналіз та різноманітні згортки [13], що дозволяють оцінити корисність того чи іншого рішення, враховуючи різноманітні критерії якості таких рішень.

В комп’ютерній грі, як правило, за такі критерії приймають максимальне здоров'я, швидкість, місце розташування та форму пересування ігрових юнітів, діапазон, втрату, швидкість стрільби тощо [14].

Згорточні методи багатокрітеріального аналізу можут бути використані при розробці алгорітмів стратегічної гри не тільки для знаходження оптимальних шляхів, але й для інших потреб, зокрема для знаходження оптимальної цілі серед декількох. На базі такого підходу може бути побудовано середній ШІ.

Достатньо популярним напрямом в ТПР є теорія операцій [15] та її оптимізаційні задачі математичного програмування, що дозволяють моделювати реальні процеси в різноманітних предметних областях.

Задачі математичного програмування розподільчого характеру [16] дозволяють адекватно промоделювати проблеми, де існують так звані роботи та ресурси, які можна розподілити для виконання таких робіт.

В комп’ютерних іграх для розподілу ігрових юнітів за ігровими шляхами можна використовувати подібні розподільчі задачі.

Такий підхід на базі побудованих математичних моделей, визначених функціях оптимізації й наборах заданих обмежень, дозволяє отримати оптимальні рішення розподільчих задач та дозволяє, таким чином, побудувати більш розвинутий штучний інтелект.

Важливим аспектом розробки стратегії такого жанру є спосіб програмного уявлення ігрової карти та сбежерення в пам'яті комп'ютера. Так ігрове поле повинне зберігатись як масив кліток, які мають певні координати та можуть містити в собі один чи декілька ігрових об'єктів, причому ігрові об’єкту можуть змінюватись.

Ігрова карта може містити гіллясту мережу шляхів. Для простоти уявлення та швидкості роботи з шляхами може бути використана теорія графів. Теорія графів є розділом математики, що вивчає властивості графів. Граф можна уявити як геометричну конфігурацію, яка складається з точок (вершини) сполучених лініями (ребрами) [17]. Таким чином ігрове поле можна уявити як граф із вершинами та ребрами.

Для зберігання даних користувачів, даних бою та різних характеристих в комп’ютерних іграх використовуються СУБД, зазвичай реляційні, але нерідко й нереляційні.

Крім того, в базах даних можна зберігати програмне уявлення ігрових об’єктів в даний момент часу, для цього краще використовувати нереляційну базу даних, що зберігає дані в оперативній пам’яті.

Розробка комп’ютерної гри починається з розробки механіки та основних принципів гри. Саме на цьому в подальшому базується проектування та реалізація.

1.2 Аналіз існуючих аналогів

На сьогоднішній день існує велика кількість різноманітних ігр у жанрі Tower Defence.

Кожна з цих засобів має свою специфіку, особливості, призначення, переваги та недоліки. Для ігр в цьому жанрі присутні такі загальні риси. Є карта, на який присутні одна, чи декілька точок входу та одна чи декілька точок виходу. Між цими точками пролягають звивисті коридори. Гравці оперують на карті ігровими об’єктами, які можна розділити на два класи: нападники та захисники. Основною метою нападників є пройти від входу до виходу (або до цілі) по коридорах мапи. Основною метою захисників є створення перешкод на шляху нападників, за допомогою будування атакуючіх веж уздовж коридорів. Ігрові об’єкти (вежі та нападники) відрізняються за своїми характеристиками.

Розглянемо декілька з таких ігр, що є популярними серед користувачів та пропонують різні особливості геймплею, такі як Defense Zone 2 HD, Kingdom Rush Origins,Tower Storm, Anomaly 2 [18].

Гра Defence Zone 2 HD є найбільш близькою до загально прийнятого шаблону ігор даного жанру (див. рис. 1.1).

Рисунок 1.1 - Геймплей ігри Defence Zone 2 HD

В даній грі користувач може грати лише за вежі, які обороняють базу від загарбників. Особливістю даної гри є той факт, що ігрові об'єкти не є квадратними, тобто усі шляхи на мапі є вигадливими, також на мапі нема виділених місць під вежі, вони розміщуються де завгодно окрім дороги. Крім того, в грі представлена можливість оптимізувати поведінку веж, шляхом задання для них пріоритету знаходження цілей.

Недоліками гри є відсутність можливості грати за атакуючих та неможливість грати з іншими користувачами, крім того вона є достатньо прямолінійною у діях комп’ютера.

Гра Anomaly 2 (див. рис. 1.2) схожа на класичні ігри в цьому жанрі, але в неї є особливість.

Рисунок 1.2 - Геймплей Anomaly 2

В цій грі користувач керує не захисниками, а атакуючими. Успіх користувача в грі залежить у першу чергу від якості планування ігрової стратегії. При розробці стратегії потрібно робити вибір атакуючої техніки, порядок її переміщення та прокладанню маршруту. треба також відзначити, що в даній грі наявний режим мультіплеєру, тобто гравці мають можливість змагатись між собою, граючи при цьому як за нападників, так і за захисників. Але при цьому гравець не може грати за захисників, коли він грає з компьютером.

Недоліком цієї гри є той факт, що гравець повинен контролювати дії своїх військ. Тому ігрові об'єкти не володіють достатнім рівнем штучного інтелекту, якій би дав їм змогу приймати рішення по найбільш ефективному маршруту.

Також потрібно відзначити серед ігр даного жанру таку гру як Tower Storm, яка представлена на рисунку 1.3.

Рисунок 1.3 - Геймплей Tower Storm

Дана гра відрізняється серед інших тим, в ній гравець має можливість змагатися з іншими гравцями, крім того кожен має можливість контролювати як наступаючих, так і захисників одночасно. Для цього ігрова мапа розподілена на дві незалежні половини. Проте потреба контролювати як нападаючих так і захисників може бути також недоліком. тому що гравець не може сконцентруватись на якійсь одній дії та продумати свою стратегію більш детально. Крім того ігрова мапа є достатньо короткою та нехитрою.

1.3 Постановка задачі

Метою роботи є дослідження оптимізаційних моделей теорії прийняття рішень для розробки штучного інтелекту для керування стратегією комп'ютера в грі в жанрі «Tower Defense».

Практичною задачею роботи є розробка ігрової системи у жанрі Tower Defence, основною особливістю якої будуть можливість двох гравців змагатися між собою онлайн з урахуванням інтелектуальних алгоритмів прийняття рішень.

Архітектура ігрової системи буде складатися з трьох основних компонентів [19]:

– ігрового клієнту, який буде містити в собі інтерфейс користувачів, а також логіку відображення ігрових подій;

– ігрового серверу, який буде містити в собі логіку обробки та зберігання даних користувачів, логіку ігри;

– бази даних, яка буде зберігати дані користувачів та зберігати тимчасові стани усіх битв у реальному часі.

Загальна структура системи з її основними компонентами представлена на рисунку 1.4.

Рисунок 1.4 – Схематичне надання архітектури системи

Отже, була поставлена задача провести математичне моделювання штучного інтелекту для гри-стратегії у жанрі Tower Defense на базі оптимізаційних моделей теорії прийняття рішень та провести їх дослідження з метою створення ефективного РШІ.

Для досягнення поставленої мети необхідно:

– розробити механіку та принципи гри в жанрі "Tower Defence";

– провести аналіз опитізаційних моделей та методів з метою створення штучного інтелекту;

– розробити математичні моделі поведінки ШІ у грі в жанрі Tower Defense на базі оптимізаційних задач теорії прийняття рішень;

– розробити алгоритми вирішення оптимізаційних задач на базі розроблених моделей;

– виконати програмну реалізацію ігрової системи;

– спланувати та провести експериментальне дослідження розроблених методів та моделей;

– розробити рекомендації стосовно використання обраного підходу для створення розвинутого ШІ в іграх-стратегіяї у жанрі Tower Defense.

2 ПЕРЕЛІК ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Концепт ігрового додатку

Ігровий додаток повинен відповідати наступним вимогам щодо технологій:

– гра для персональних комп’ютерів під управлінням ОС Windows;

– керування за допомогою клавіатури (контроль камери) та миші;

– тривимірна графіка;

– онлайн гра за технологією клієнт-сервер;

– підтримка режиму мультіплеєру.

Основна механіка гри - Tower Defense. Грають 2 гравця через інтернет. Один гравець грає за вежі (захищається), другий гравець за монстрів (атакує). Гравці змагаються між собою. Стратегія в реальному часі.

Битва відбувається на ігровому полі розбитому на клітини. На поле є дороги, по яких ходять монстри і земля, на якій будуються вежі. Всі дороги починаються з одного входу, розгалужуються, і ведуть до замку (кінця).

Мета атакуючого гравця - зруйнувати за відведений час замок противника. Для цього він повинен вибирати монстрів, яких потрібно запустити на поле. Монстри рухаються і знаходять шлях автоматично, рухаються від входу до замку.

Мета гравця, що захищається , не дати атакуючим зруйнувати замок за відведений час. Для цього він будує вежі на будь-яких клітинах землі (не на дорозі). Вежі атакують і знищують монстрів, цілі знаходять автоматично, без участі користувача.

Вежі й монстри купуються за ресурси (монети). Атакуючий заробляє ресурси за довжину шляху, який пройшли його монстри. Той, що захищається за характеристики убитих його вежами монстрів.

Бій триває 30 хвилин. Ігрове поле генерується кожен раз. За битви гравці отримують досвід. При цьому вони досягають певних рівнів.

У грі присутні 3 типи об'єктів на ігровому полі:

– вежі. Будуються гравцем в певній позиції на землі, нерухомі. Після завершення будівництва залишаються в позиції до кінця битви, не можуть бути зруйновані. Атакують монстрів, яких вони можуть дістати;

– монстри. Запускаються гравцем на ігрове поле. З'являються на старті і поступово переміщуються до замку. При досягненні замку завдають йому шкоди і пропадають з поля битви. Можуть бути знищені вежами. Для переміщення до замку вибирають один із шляхів в залежності від пріоритету;

– замок. Розташовується в кінці шляхів ігрового поля. Володіє рівнем здоров'я, рівним 100 одиниць. При руйнуванні замку гра закінчується перемогою атакуючих (монстрів), при не руйнуванні за відведений час - перемогою захисників (веж).

Весь прорахунок ігровий логіки і синхронізація дій відбувається на ігровому сервері, на ігровому клієнті відбувається лише графічне відображення дій.

2.2 Функціональні вимоги

Функціональні вимоги описують сервіси, що надаються програмної системою, її поведінку в певних ситуаціях, реакцію на ті чи інші вхідні дані і дії, які система дозволить виконувати користувачам.

Система повинна підтримувати процедури авторизації та аутентифікації. Якщо користувач не є зареєстрованим у системі, система повинна надавати можливість зареєструватися, тобто створити нового користувача. Персональні дані, які користувач вводить під час реєстрації повинні бути провалідовані на відповідність вимогам.

Після того, як користувача було авторизовано, він автоматично має перейти до системи. Якщо користувача не було успішно авторизовано, то він отримає відповідне повідомлення на екрані.

Авторизований користувач має обмежений час для того, щоб знаходитися у сесії. Якщо цей час пройшов, то при першій дії користувача, система зобов’язана перенаправити користувача на сторінку із формою авторизації.

Система зобов’язана надавати користувачеві інформацію про його поточний прогрес у грі. Користувач має бачити свій поточний рівень, кількість одиниць досвіду, та кількість одиниць досвіду, які він потребує для переходу до наступного рівня.

Користувач повинен мати можливість розпочати пошук супротивника для участі у бою. Система повинна підібрати для користувача супротивника, який також шукає битву та самостійно обрати сторону, за яку буде виступати кожен гравець. Якщо супротивник для гравця не буде знайдений на протязі однієї хвилини, система повинна повідомити гравцю про те що в даний момент не може знайти супротивника, та завершити пошук.

Якщо пара супротивників для бою знайдена, система повинна створити новий бій з ігровим полем та видати його обом гравцям. В залежності від сторони, за яку грає гравець, йому повинні бути відображені елементи управління. Кожен з гравців повинен бачити час, що залишився до закінчення бою, поточну кількість здоров’я замку, кількість доступних ресурсів для придбання нових ігрових об’єктів. Кожному гравцю повинен бути показаний список ігрових об’єктів (монстрів чи веж), доступних до придбання. При цьому, якщо в гравця недостатньо ресурсів, об’єкт повинен бути неактивним. При виборі об’єкту зі списку у інформаційномі вікні повинні відображуватись характеристики.

Результат битви повинен бути прорахований в залежності від розташування об’єктів, їх кількості, взаємодії, характеристик та прийнятих рішень. Рішення ігрові об’єкти приймають самостійно, без участі гравця.

При перемозі однієї з сторін гравці повинні отримати повідомлення про це, та отримати пропозицію про вихід з битви. Якщо один з гравців вийде з битви раніше, другий повинен отримати повідомлення про це, та пропозицію про вихід з битви. В такому разі він буде вважатися переможцем.

Гравці повинні мати можливість перегляду таблиці рейтингу всіх гравців системи, де кожному гравцю відповідає місце, та вони повинні мати змогу шукати себе у рейтингу.

2.3 Нефункціональні вимоги

До системи ставляться наступні нефункціональні вимоги:

а) доступність. Серверна частина має працювати постійно, незалежно від завантаженості. Ігровий додаток має працювати виключно при безпосередньому використанні;

б) вимоги до зберігання даних. Усі дані мають бути збережені у базі даних MySQL та бути винесені до міграцій. Час зберігання необмежений. Дані, що зберігаються у пам’яті клієнтських додатків, мають бути видалені одразу ж після синхронізації із сервером;

в) масштабованість. Система має підтримувати як горизонтальну масштабованість через створення нових кластерів із додатком, так і вертикальну, що має підсилити її продуктивність;

г) вимоги до зручності використання. Інтерфейс користувача має бути адаптивним, тобто бути зручним у використанні через ігровий додаток на різних комп’ютерах;

д) вимоги до безпеки. Дані, що передаються від ігрових клієнтів до серверу та від серверу до ігрових клієнтів повинні бути зашифровані. Під час обробки кожного запиту до серверу, запитувач має бути авторизований у системі через використання унікального токену, що має зберигатися на клієнті та видаватись сервером під час авторизації. Для генерації токену повинен бути використаний алгоритм шифрування AES. У разі відсутності токену у сховищі, користувач отримує помилку 401 Unauthorized. Пароль користувача не повинен зберігатися у базі у первинному вигляді, він повинен бути записаний до бази у

зашифрованому вигляді, та звірятися із хешом паролю, який вводить користувач під час авторизації у системі;

е) вимоги до конфігурування. Серверна частина повинна мати файл Web.config, де мають знаходитися рядки із конфігурацією доступу до бази даних;

ж) обмеження. Середовище розгортання серверної частини має підтримувати технологію .NET 4.5, та мати ОС Windows із встановленою програмою Visual Studio 2015. У системі має бути установлений Nuget, сервер IIS, база даних MySQL. В свою чергу, ігровий додаток має запускатися на ОС Windows різних версій;

з) вимоги до можливості та простоти локалізації. Система має підтримувати лише англійську мову для інтерфейсу користувача. Інформація, що зберігається у базі даних не обмежена у мові;

и) вимоги до можливості тестування. Інтерфейс користувача та основні ігрові механіки під час ігрового процесу мають бути перевірений методом мануального тесутвання;

і) вимоги до синхронізації. Синхронизація ігрового стану повинна виконуватись кожен раз, коли один з гравців змінює стан битви.

3 ОПИС ДОСЛІДЖЕННЯ ТА ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

3.1 Розробка механіки та принципів гри

Основна механіка гри - Tower Defence. Грають 2 гравця через інтернет. Один гравець грає за вежі (захищається), другий гравець за монстрів (атакує). Гравці змагаються між собою.

Битва відбувається на ігровому полі розбитому на клітини. На поле є дороги, по яких ходять монстри і земля, на якій будуються вежі. Всі дороги починаються з одного входу, розгалужуються, і ведуть до замку (кінця).

Мета атакуючого гравця - зруйнувати за відведений час замок противника. Для цього він повинен вибирати монстрів, яких потрібно запустити на поле. Монстри рухаються і знаходять шлях автоматично, рухаються від входу до замку.

Мета гравця, що захищається , не дати атакуючим зруйнувати замок за відведений час. Для цього він будує вежі на будь-яких клітинах землі (не на дорозі). Вежі атакують і знищують монстрів, цілі знаходять автоматично.

Вежі й монстри купуються за ресурси (монети). Атакуючий заробляє ресурси за довжину шляху, який пройшли його монстри. Той, що захищається за характеристики убитих його вежами монстрів.

Бій триває 30 хвилин. Ігрове поле генерується кожен раз. За битви гравці отримують досвід. При цьому вони досягають певних рівнів.

У грі присутні 3 типи об'єктів на ігровому полі:

– вежі. Будуються гравцем в певній позиції на землі, нерухомі. Після завершення будівництва залишаються в позиції до кінця битви, не можуть бути зруйновані. Атакують монстрів, яких вони можуть дістати;

– монстри. Запускаються гравцем на ігрове поле. З'являються на старті і поступово переміщуються до замку. При досягненні замку завдають йому шкоди і пропадають з поля битви. Можуть бути знищені вежами. Для переміщення до замку вибирають один із шляхів в залежності від пріоритету;

– замок. Розташовується в кінці шляхів ігрового поля. Володіє рівнем здоров'я, рівним 100 одиниць. При руйнуванні замку гра закінчується перемогою атакуючих (монстрів), при не руйнуванні за відведений час - перемогою захисників (веж).

Весь прорахунок ігровий логіки і синхронізація дій відбувається на ігровому сервері, на ігровому клієнті відбувається лише графічне відображення дій.

3.2 Аналіз оптимізаційних моделей та методів з метою створення штучного інтелекту

На сьогоднішній день існує велика кількість різноманітних досліджень у області розробки математичних моделей штучного інтелекту для ігр в жанрі Tower Defense.

Далі описані основні тезиси, що описуються у останніх дослідженнях у предметній області.

Для оптимального знаходження шляхів між різними точками на ігровому полі була розроблена графова модель збереження шляхів на полі в пам’яті програмної системи у вигляді орієнтованого графа.

Такий підхід спрощення та прискорює обчислення маршруту переміщення ігрових сутностей.

Для знаходження найкоротшого шляху між двома точками на ігровому полі може бути застосований один з алгоритмів пошуку найкоротшого шляху на графі.

Схематичне зображення графової структури шляхів на полі, що складається з клітинок зображено на рисунку 3.1.

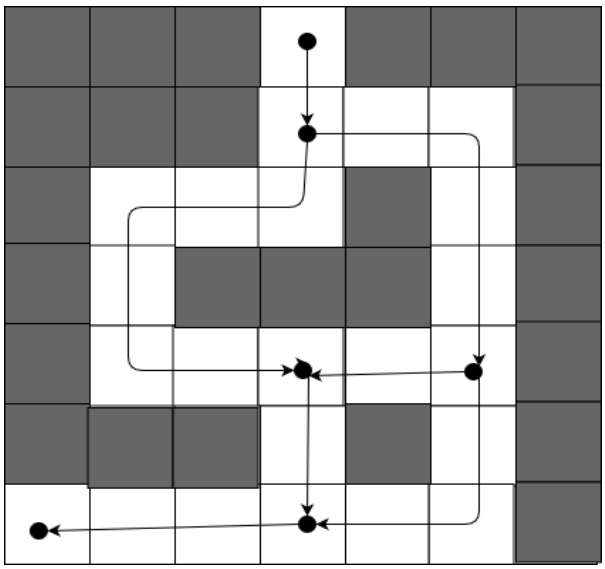

Рисунок 3.1 – Схематичне представлення шляхів на ігровому полі

Для розробки розвиненого ШІ для ком’ютерної гри в жанрі Tower Defense був проведений аналіз існуючих оптимізаційних моделей та методів. Поставлена задача може бути вірішена методами прийняття рішень в умовах визначенності, тому в роботі не було розглянено методи прийняття рішень в умовах невизначеності та ризику.

В якості методів прийняття рішень в умовах визначеності традиційно використовуються методи на базі теорії корисності та на базі теорії дослідження операцій.

Серед методів теорії прийняття рішень на базі теорії корисності насамперед потрібно визначити методи багатокрітеріального аналізу, тому що інші методи, такі як лінійне та нелінійне програмування використовуються при знаходженні рішення на основі одного крітерія. Серед методів багатокритеріального аналізу слід відзначити методи згорток.

Методи згорток надають найбільшої гнучкість при знаходженні оптимального рішення.

Аддитивна згортка є найкращім вибіром, коли зменшення оцінки за одним з критерієв компенсується збільшенням оцінки по одному з інших критеріев (або кількома критеріями).

Перераховані вище методи багатокрітеріального аналізу з теорії прийняття рішень можут бути використані при розробці алгорітмів стратегічної гри не тільки для знаходження оптимальних шляхів, але й для інших потреб, зокрема для знаходження оптимальної цілі серед декількох. Взагалі, згорткою називається процес приведення багатокритерійної задачі до однокритерійної.

Всі лінійні згортки грунтуються на принципі, що низька оцінка за одним критерієм може бути компенсована високою оцінкою по іншому. Самою простою є лінійна адитивна згортка. Дана згортка підраховує, скільки разів тий чи інший варіант був оптимальний. Проте вона не враховує кількісних показників значень критеріїв.

При грі комп'ютера на боці монстрів його метою є знищення замку супротивника. Для цього потрібно нанести йому максимальний можливий урон. У кожного юніту є кількість одиниць урону, які він наносить замку, у тому разі, якщо він до нього дійде. При цьому, не завжди важливо, щоб один окремо взятий юніт дістався замку, тому що він потенційно може допомогти іншим юнитам дістатися замку, наприклад відволікаючи атаку веж на себе, тобто, юніти залежать від інших юнитів.

Таким чином, для усіх юнитів, що можуть бути куплені у даний момент потрібно розрахувати урон для кожного з потенційних шляхів. З отриманих значень обираємо найбільше. Програмна система має змогу розрахувати потенційний урон, що може бути завданий замку, у разі запуску юнита на поле.

Задача вибору юніта та його шляху на ігровому полі може бути вирішена за допомогою методу лінійних адитивних згорток.

Лінійна адитивна згортка з ваговими коефіцієнтами обирає варіант, для якого максимальним є значення, що обчислюється по формулі:

$$Z = \sum_{j=1}^{n} \alpha_j \beta_j a_{ij},$$

де α_j - нормуючі множники,

β_j - вагові коефіцієнти, що залежать від відносного вкладу всіх критеріїв в загальний критерій,

a_{ij} - значення оцінки певного критерію для даного варіанту. При цьому, вагові коефіцієнти прийнято вказувати вже нормованими величинами.

Даний метод дозволяє найбільш ефективно обирати варіанти на основі багатьох критеріїв, тому саме цей метод був розглянутий для застосування в ігровій системі.

Для пошуку оптимального шляху ігрових об’єктів на ігровому адаптований алгоритм зваженої суми оцінок на підставі адитивної згортки з зваженими коефіцієнтами.

Серед методів прийняття рішень на базі теорії дослідження операцій потрібно відзначити задачі лінійного та нелінійного програмування. Математичне програмування використовується в задачах, де можливо пободувати однозначну математичну модель та існує один критерій оптимізації і набором обмежень на наведені змінні [20].

Методи знаходження оптимальних рішень досліджується в розділах теорії прийняття рішень, зокрема в теорії дослідження операцій. У ній досліджується вирішення задач математичного програмування (ЗМП).

Задачі математичного програмування застосовуються для: знаходження оптимального, єдино вірного і науково обгрунтованого рішення. Після знаходження рішення досліджується система, зовнішнє середовище і шукається адекватна модель.

Для побудови математичної моделі оптимізаційної ЗМП необхідно:

– визначити величини, для яких має бути побудована модель, тобто, як ідентифікувати шукані величини даної задачі;

– визначити обмеження змінних, щоб виконувалися умови, що характеризують систему яка моделюється;

– визначити ціль, для досягнення якої із усіх допустимих значень змінних необхідно обрати тільки ті, що задовільняють оптимальному розв’язанню задачі.

Використовують таку класифікацію ЗМП:

– задачі цілочисельного програмування, в яких невідомі набувають тільки цілочисельних значень;

– задачі лінійного програмування (ЗЛП), де всі функції в моделі лінійні;

– задачі нелінійного програмування (ЗЛП), де хоча б одна із функції в моделі нелінійна, до цього виду відносятьзадачі дискретного програмування, задачі квадратичного програмування, задачі опуклого програмування,

- задачі параметричного програмування – такі задачі, де цільова функція, тобто функція, яка визначає можливі зміни невідомих, залежать від деяких параметрів;

– задачі дрібно-лінійного програмування, в яких функції, що визначають область можливих змін невідомих є лінійні, а цільова функція є відношенням двох лінійних функцій;

– задачі динамічного програмування, в яких алгоритм вирішення є багатоетапним (багатокроковим);

– задачі стохастичного програмування, такі задачі, в яких утримуються випадкові величини для цільовій функції або функції, що визначає область можливих змін невідомих.

Лінійне програмування - широко поширений метод оптимізації використовування обмежених ресурсів.

Слід зазначите, що методи лінійного програмування можуть застосовуватись, якщо поставлена тільки одна ціль: максимізувати (наприклад, прибуток) чи мінімізувати (наприклад, витрати). Коли цілей декілька, використовується цільове програмування. Якщо ж задача ефективніше всього вирішується поетапно чи по часовим інтервалам, аналітику слід скористатися методом динамічного програмування. В ще більш складних задачах при вирішенні можуть стати потрібними інші варіанти даного методу, наприклад нелінійне, чи квадратичне програмування.

Такі задачі ще назиіають оптимізаційними задачами розподілу ресурсів. Їх застосовують тоді, коли існує послідовність дій, які потрібно виконати, із різними шляхами виконання. Крім того задача характеризується обмеженістю ресурсів, яких недостатньо для виконання всіх робіт.

Можно сформулювати задачу таким чином: знайти такий розподіл ресурсів за роботами, при якому сумарні витрати на виконання усіх видів робіт, були мінімальними, або одержуваний загальний прибуток був максимальним.

Таким чином, в оптимізаційній моделі використовуються:

– Сіj - витрати або прибуток, щодо виділення однієї одиниці ресурсу до роботи;

– невідомі Хіj, як правило це об’єм і-го ресурсу, необхідного для виконання j-ї роботи.

Загальні витрати, чи прибуток, на використання і-го ресурсу для виконання j-ї роботи обчислюється як Хіj*Сіj. Маємо маємо лінійну розподільну задачу, цільова функція:

$$F = \sum_{j=1}^{n} \sum_{i=1}^{m} C_{ij} * X_{ij} \rightarrow extremum$$

Система обмежень задається таким чином:

$$\sum_{j=1}^{n} X_{ij} \leq x_i, \ \ \forall i = \overline{1, m}$$

$$\sum_{i=1}^{m} X_{ij} \geq y_j, \ \ \forall j = \overline{1, n}$$

$$X_{ij} \geq 0$$

Важливо підкреслити, що на практиці до типового застосування методів лінійного програмування належать симплексний та транспортний методи, жадібний алгоритм, угорський алгоритм.

Для вирішення задачі методом лінійного програмування необхідно, щоб описана в ній ситуація відповідала п'яти основним умовам:

– пов'язана з обмеженими ресурсами;

– сформульована точна ціль (максимізація чи мінімізація);

– задача характеризується лінійністю;

– задача характеризується однородністю;

– задача характеризується дилимістю, тобто метод лінійного програмування будується на припущенні, що результати і ресурси можна поділити на долі, та якщо таке ділення не можливе, краще скористатися спеціальною модифікацією лінійного програмування — дискретним програмуванням.

Симплексний метод - це алгебраїчна процедура, в результаті якої послідовно наближається до знаходження оптимального рішення.

В теорії даний метод можна використовувати для вирішення задач, які включають будь-яку кількість обмежень і змінних, але якщо в них, більше чотирьох змінних чи обмежень, то обчислення стає досить складним.

Транспортний метод представляє собою спрощений варіант симплексного методу. Він отримав таку назву, тому що широко застосовується для вирішення задач транспортування продукції з різних локацій в різні пункти призначення.

Жадібний алгоритм — простий і прямолінійний евристичний алгоритм, який приймає найкраще рішення, виходячи з наявних на поточному етапі даних, не турбуючись про можливі наслідки, сподіваючись врешті-решт отримати оптимальне рішення. Легкий в реалізації і часто дуже ефективний за часом виконання. Багато задач не можуть бути розв'язані з його допомогою.

3.3 Моделювання штучного інтелекту на базі теорії корисності

Для моделювання ШІ взята поведінка комп'ютера на боці монстрів-нападників з метою знищення замку супротивника-людини. Для знищення, штучному інтелекту необхідно нанести максимальний можливий урон супротивнику. У кожного ігрового юніту є кількість одиниць урону, які він наносить замку у тому разі, якщо він до нього дійде. При цьому, не завжди важливо, щоб один окремо взятий юніт дістався замку, тому що він потенційно

може допомогти іншим юнитам дістатися замку, наприклад відволікаючи атаку веж на себе, тобто, юніти залежать від інших юнитів.

Таким чином, для усіх юнитів, що можуть бути куплені у даний момент (тобто на яких вистачає грошей) потрібно розрахувати урон для кожного з потенційних шляхів, при цьому враховуючи їх ціну (будемо ділити потенційний урон на ціну). З отриманих значень обираємо найбільше. ШІ потрібен розрахувати потенційний урон, що може бути завданий замку у разі запуску юнита на поле.

Для подальшого моделювання штучного інтелекту введемо такі математичні позначення для ігрових сутностей:

– $\{U_i\}_{i=1}^{m}$ – множина юнітів U_i, що доступні для запуску на поле в грі, всього доступно m юнітів;

– $P = \{P_i\}_{i=1}^{n}$ – множина шляхів P_i на ігровому полі;

– $T = \{T_i\}_{i=1}^{m}$ – список усіх веж, що доступні для запуску на поле в грі, всього доступно m веж;

– $P^* = \{P^*_{(x,y)}\}_{x=1,y=1}^{X,Y}$ – множина клітин на ігровому полі X×Y.

– X – ширина ігрового поля;

– Y – довжина ігрового поля;

– C_i – ціна і-го юніту або вежі у внутрішніх ресурсах;

– Q_i – сила атаки юніта і;

– H_i – кількість одиниць життя юніту і;

– V_i – сила атаки вежі і;

– M_M – кількість ресурсів гравця за монстрів;

– M_T – кількість ресурсів гравця за вежі.

Задача вибору юніта та його шляху на ігровому полі може бути промодельована за допомогою теорії корисності, а саме моделі лінійної адитивної згортки:

$$Z = \sum_{j=1}^{n} \alpha_j \beta_j K_j(P_i)$$

де α_j - нормуючі множники,

β_j - вагові коефіцієнти, що залежать від відносного вкладу всіх критеріїв в загальний критерій,

$K_j(P_i)$ – критеріальна оцінки за критеріїм K_j для варіанту шляху P_i.

При цьому, вагові коефіцієнти прийнято вказувати вже нормованими величинами [21].

Даний метод дозволяє найбільш ефективно обирати варіанти на основі багатьох критеріїв, тому саме цей метод був розглянутий для застосування в ігровій системі.

Для вирішення задачі призначення юніту до певного шляху в розробленій моделі (1) було запропоновано наступні критерії з ваговими коефіцієнтами:

– K_1 – загальна кількість захисних веж на шляху (ваговий коефіцієнт $\beta_1 = 0{,}1$);

– K_2 – безпека шляху: кількість одиниць шкоди, які потенційно можуть завдати нападнику вежі на шляху ($\beta_2 = 0{,}5$);

– K_3 – потенційна кількість ресурсів, які може отримати нападник, пройшовши по шляху або його частині ($\beta_3 = 0{,}3$);

– K_4 – відношення захисних веж на шляху з особливими ефектами при атаці до загальної кількості веж на шляху ($\beta_4 = 0{,}1$).

Такі обчислення при виборі юніта, якого потрібно додати на ігрове поле, потрібно виконувати для кожного юніту, та потім обирати варіант із найкращим значенням.

Для вирішення задачі призначення вежі до певної позиції на ігровому полі в розробленій моделі (1) було запропоновано наступні критерії з ваговими коефіцієнтами:

– K_1 – загальна кількість захисних веж на шляхах, що можуть бути атаковані вежею з позиції (ваговий коефіцієнт $\beta_1 = 0{,}3$);

– K_2 – кількість клітин на ігровому полі, що належить до множини шляхів та може бути атакована вежею з позиції ($\beta_2 = 0{,}3$);

– K_3 – близкість до середини ігрового поля ($\beta_3 = 0{,}2$);

– K_4 – загальна кількість нападників що в момент обчислення знаходяться на ігровому полі, та можуть бути потенційно атаковані вежею вежею з позиції ($\beta_4 = 0{,}2$).

Такі обчислення при виборі вежі, яку потрібно розмістити до певною позиції на ігровому полі, потрібно виконувати для кожного юніту, та потім обирати варіант із найкращим значенням.

3.4 Моделювання штучного інтелекту на базі задач лінійного програмування

Математичне програмування - широко поширений метод оптимізації використання обмежених ресурсів [20]. Конкретно методи лінійного програмування можуть застосовуватись, якщо поставлена тільки одна ціль: максимізувати (наприклад, прибуток) чи мінімізувати (наприклад, витрати). Якщо ж задача ефективніше всього вирішується поетапно чи по часовим інтервалам, слід скористатися методом динамічного програмування, а в ще більш складних задачах можуть стати потрібними інші варіанти даного підходу, наприклад нелінійне, чи квадратичне програмування.

Для моделювання ШІ на базі задач лінійного програмування була розглянута поведінка комп'ютера з боку обох супротивників:

– з боку нападника: під час запуску юніта (монстра) на ігрове поле по певному шляху;

– з боку захисника: під час розташування захисних веж на шляхах мапи.

Задача призначення юніту на той чи інший шлях може бути віднесена до оптимізаційних задач про призначення. Вона може бути сформульована наступним чином: розподілити (призначити) наявні на ігровому полі юніти на шляхи таким чином, щоб мінімізувати урон, який понесуть юнити під час

проходження цими шляхами. Табличне надання цієї оптимізаційної задачі надано в таблиці 3.1.

Таблиця 3.1 – Табличне надання оптимізаційної задача призначення юніта на шлях

Юніти	Шляхи				
	P_1	P_2	…	P_n	Обмеження за юнітами U_i
U_1 (C_1)	D_{11}, X_{11}	D_{12}, X_{12}	…	D_{1n}, X_{1n}	1
U_2 (C_2)	D_{21}, X_{21}	D_{22}, X_{22}	…	D_{2n}, X_{2n}	1
…	…	…	…	…	
U_m (C_m)	D_{m1}, X_{m1}	D_{m2}, X_{m2}	…	D_{mn}, X_{mn}	1
Обмеження за шляхами P_j	1	1		1	

Для моделювання задачі про призначення юнітів на шляхи введемо наступні позначення:

– $U = \{U_i\}_{i=1}^{m}$ – множина юнітів Ui, що доступні для запуску на поле в грі;

– Ci – ціна i-го юніту у внутрішніх ресурсах;

– $P = \{P_i\}_{i=1}^{n}$ – список усіх шляхів на ігровому полі;

– Dij – потенційний урон, що може бути завданий замку супротивника, у випадку, якщо юніт *i* буде запущений на шлях *j*.

Введемо змінну *Xij* , що символізує, чи був запущений *i*-й юніт по *j*-му шляху, змінна може приймати значення 1 (якщо юніт був запущений) або 0 (якщо не був).

Потенційний урон *Dij* розраховується за формулою:

$$D_{ij} = Q_i L_{ij} + \sum_{u=0}^{U} Q_u L_{uij}$$

де Q_i – сила атаки юніта i,

L_{ij} – факт того, дійде юніт i до кінця ігрового полю по шляху j, чи ні, може приймати значення 0 чи 1:

$$L_{ij} = \begin{cases} 0, H_i > T_{ij} \\ 1, H_i \leq T_{ij} \end{cases}$$

де H_i – кількість одиниць життя юніту *i*,

T_{ij} – кількість урону, що буде завдано юніту *i* на шляху *j*.

Для даної постановки задачі повинні виконуватися наступні умови:

– ціна *i*-го юніта повинна бути менша або дорівнювати кількості ресурсів у гравця за монстрів $Ci < MM$, де MM – кількість ресурсів, що є в наявності в гравця за монстрів;

– кожен юніт може бути призначений не більше ніж на один шлях, отже $\sum_{j=1}^{n} x_{ij} \leq 1,\ \forall_i = \overline{1,m}$;

– на кожен шлях може бути призначений лише один юніт, отже $\sum_{i=1}^{m} x_{ij} = 1,\ \forall_j = \overline{1,n}.$;

– i-й юніт може бути призначений або не призначений на j-ий шлях, отже $X_{ij} = 1|0, \forall i = \overline{1,m}, \forall j = \overline{1,n}$.

Цільова функція повинна забезпечити таке призначення юнітів за шляхами, при якому би досягався максимальний потенційний урон замку супротивника, що грає за вежі (що еквівалентно мінімальному урону юнітам гравця), отже

$$F = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{D_{ij} X_{ij}}{C_i} \rightarrow \max,$$

$$F = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{X_{ij}(Q_i L_{ij} + \sum_{u=0}^{U} Q_u L_{uij})}{C_i} \rightarrow \max.$$

Таким чином, математична модель оптимізаційної задачі про призначення штучним інтелектом юнітів на шляхи приймає наступний вигляд:

$$F = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{X_{ij}(Q_i L_{ij} + \sum_{u=0}^{U} Q_u L_{uij})}{C_i} \rightarrow \max,$$

$$\sum_{j=1}^{n} x_{ij} \leq 1,\ \forall_i = \overline{1,m};$$

$$\sum_{i=1}^{m} x_{ij} = 1,\ \forall_j = \overline{1,n};$$

$$X_{ij} = 1|0, \forall i = \overline{1,m}, \forall j = \overline{1,n}.$$

При грі ШІ на боці веж його метою є захист свого замку, тобто знищення юнитів нападника. Для цього потрібно наносити їм урон та знищувати. При встановленні вежі на полі, вона з нього нікуди не зникає, та має змогу впливати на подальший ігровий процес. При цьому, при розміщенні вежі у тій, чи іншій клітині на ігровому полі мають значення такі показники, як скільки урону буде завдано юнітам супротивника, які в даний момент часу знаходяться на полі; скільки з них при цьому буде знищено; та потенційний урон, що буде завданий вежею з обраної позиції у майбутньому.

Потрібно також додати, що для спрощення обчислень ми маємо змогу обирати для розрахунку не всі клітини, що є на ігровому полі, а лише ті, сума індексів яких ділиться на певне число, що обирається випадково.

Таким чином, буде рівномірно обрано декілька клітин із різних частин ігрового поля, що дозволить найбільши ефективно виконати обчислення. Тобто обираємо такі клітини C[x,y], для яких виконується:

$$(x + y) \bmod r = 0,$$

де r – випадкове число від 1 до 10.

Таким чином, задача розміщеня веж на ігровому полі в певній позиції може бути промодельована як оптимізаційна задача лінійного програмування, а саме як задача про призначення.

Множина клітин *P** обирається таким чином: P*=U–W–B, де *U* – множина всіх клітин на полі, *W* – множина клітин, що належить до шляху, *B* – множина клітин що зайняті вежами.

Для моделювання задачі про призначення веж на клітини мапи введемо наступні додаткові позначення:

– $T = \{T_i\}_{i=1}^{m}$ – список усіх веж, що доступні для запуску на поле в грі;

– Ci – ціна *i*-ї вежі;

– $P^{*}=\{P^{*}_{(x,y)}\}_{x=1,y=1}^{X,Y}$ – множина клітин на ігровому полі $X \times Y$.

При цьому, обираються лише ті клітини, для яких виконується умова:

$$(x+y) \bmod k = 0$$

де k –деяке число від 0 до 5, що обирається випадково.

Табличне надання оптимізаційної задачі призначення вежі на клітину надано в таблиці 3.2.

Таблиця 3.2 – Оптимізаціна задача призначення вежі на клітину

Вежі	Клітини			
	$P_{(1,1)}$	…	$P_{(x,y)}$	Обмеження на вежі T_i
T_1 (C_1)	$D_{1(1,1)}, A_{1(1,1)}, K_{1(1,1)}, B_{1(1,1)}$	…	$D_{1(x,y)}, A_{1(x,y)}, K_{1(x,y)}, B_{1(x,y)}$	1
T_2 (C_2)	$D_{2(1,1)}, A_{2(1,1)}, K_{2(1,1)}, B_{2(1,1)}$	…	$D_{2(x,y)}, A_{2(x,y)}, K_{2(x,y)}, B_{2(x,y)}$	1
…	…	…	…	
T_m (C_m)	$D_{m(1,1)}, A_{m(1,1)}, K_{m(1,1)}, B_{m(1,1)}$	…	$D_{m(x,y)}, A_{m(x,y)}, K_{m(x,y)}, B_{m(x,y)}$	1
Обмеження на клітини $P^{*}_{(x,y)}$	1		1	

$A_i(x,y)$ – урон, що може бути завданий юнітам супротивника, які у даний момент знаходяться на полі, якщо вежа *i* буде розташована у позиції *(x,y)*. $A_i(x,y)$ розраховується таким чином:

$$A_{i(x,y)} = \sum_{u=0}^{U}\left(Z_{iu}F_{ui(x,y)} + \sum_{t=0}^{T} Z_{tu}F_{tui(x,y)}\right)$$

де U – множина юнітів, що зараз є на полі;

T - множина веж, що зараз є на полі;

Z_{tu}- кількість урону, що отримає юніт *u* від вежі *t*;

$F_{ui(x,y)}$ – кількість разів, коли юніт *u* атакований вежою *i*, якщо вежа *i* буде розташована у клітині *(x,y);*

$F_{uti(x,y)}$ - кількість разів, коли юніт *u* атакований вежою *t*, чи ні, якщо вежа *i* буде розташована у клітині *(x,y) (*може приймати значення 0 чи 1).

$K_i(x, y)$ – кількість юнітів, з тих, що у даний момент знаходяться на ігровому полі та що будуть знищені, якщо розмістити *i*-ю вежою у позиції *(x,y)*. $K_i(x, y)$ розраховується таким чином:

$$K_{i(x,y)} = \sum_{u=0}^{U} L_{ui(x,y)}$$

де L_{uij} - факт того, дійде юніт *u* до кінця ігрового полю, чи ні, якщо вежа *i* буде розташована у позиції *(x,y)*. Може приймати значення 0 чи 1:

$$L_{uij} = \begin{cases} 0, H_u > T_{uij} \\ 1, H_u \leq T_{uij} \end{cases}$$

де H_u – кількість одиниць життя юніту *u*; T_{uij} – кількість урону, що буде завдано юніту *u*, якщо вежа *i* буде розташована у позиції *(x,y)*.

$D_i(x, y)$ – потенційний урон, що може бути завданий по навколишнім клітинам *i*-ю вежою у позиції *(x,y)*. $D_i(x, y)$ розраховується таким чином:

$$D_{i(x,y)} = V_i N_{i(x,y)}$$

де Vi – сила атаки вежі *i*;

$N_{i(x,y)}$ – кількість клітин, що може бути атаковано вежою *i* з позиції (x,y).

Для даної оптимізаційної задачі необхідно промоделювати наступні умови:

– ціна *i*-ї вежі повинна бути меньша або дорівнювати кількості ресурсів у гравця за вежі: *Ci* < *MT*, де *MT* – кількість ресурсів, що є в наявності в гравця за вежі;

– кожна вежа може бути призначена не більше ніж на одну клітину, отже $\sum_{j=1}^{n} b_{ixy} \leq 1,\ \forall_i = \overline{1, m}$;

– на кожен шлях може бути призначений лише один юніт, отже $\sum_{i=1}^{m} x_{ij} = 1,\ \forall_j = \overline{1,n}$;

– *i*-та вежа може бути призначена або не призначена на клітину *P*(x,y)*, отже $X_{ij} = 1|0, \forall i = \overline{1,m}, \forall j = \overline{1,n}$.

Введемо змінну $B_i(x,y)$, що символізує чи була поставлена *i*-та вежа у позицію *(x,y)*, $B_i(x,y)$ може приймати значення 1 (якщо вежа була поставлена) або 0 (якщо не була).

Таким чином, цільова функція може бути виражена як:

$$F = \sum_{i=1}^{m}\sum_{x=1}^{X}\sum_{y=1}^{Y}\frac{(A_{i(x,y)}+1)\left(K_{i(x,y)}+1\right)D_{i(x,y)}X_{i(x,y)}}{C_i} \to \max.$$

Отже, математична модель оптимізаційної задачі про призначення ШІ веж на позиції на ігровій мапі приймає наступний вигляд:

$$F = \sum_{i=1}^{m}\sum_{x=1}^{X}\sum_{y=1}^{Y}\frac{(A_{i(x,y)}+1)\left(K_{i(x,y)}+1\right)D_{i(x,y)}X_{i(x,y)}}{C_i} \to \max,$$

$$\sum_{x=1}^{X}\sum_{y=1}^{Y} b_{ixy} \le 1,\ \forall_x = \overline{1,X}, \forall_y = \overline{1,Y},$$

$$\sum_{i=1}^{m} b_{ixy} = 1,\ \forall_j = \overline{1,n},$$

$$B_{ij} = 1|0, \forall i = \overline{1,m}, \forall_x = \overline{1,X}, \forall_y = \overline{1,Y}.$$

На базі розроблених моделей потрібно розробити схему бази даних для зберігання потрібної інформації. Для практичної реалізації математичних моделей потрібно розробити алгоритми вирішення задачі про призначення з використанням методів, що були досліджені.

Для визначення ефективності розроблених алгоритмів для вирішення поставлених задач повинно бути проведено експериментальне дослідження.

3.5 Розробка бази даних для зберігання інформації з розроблених моделей

Проектування бази даних програмної системи складається з етапів:

– концептуального моделювання;

– логічного моделювання;

– фізичного моделювання.

Основний компонент концептуальної моделі - це опис об’єктів предметної області та зв’язків між цими об’єктами (див. рис. 3.2).

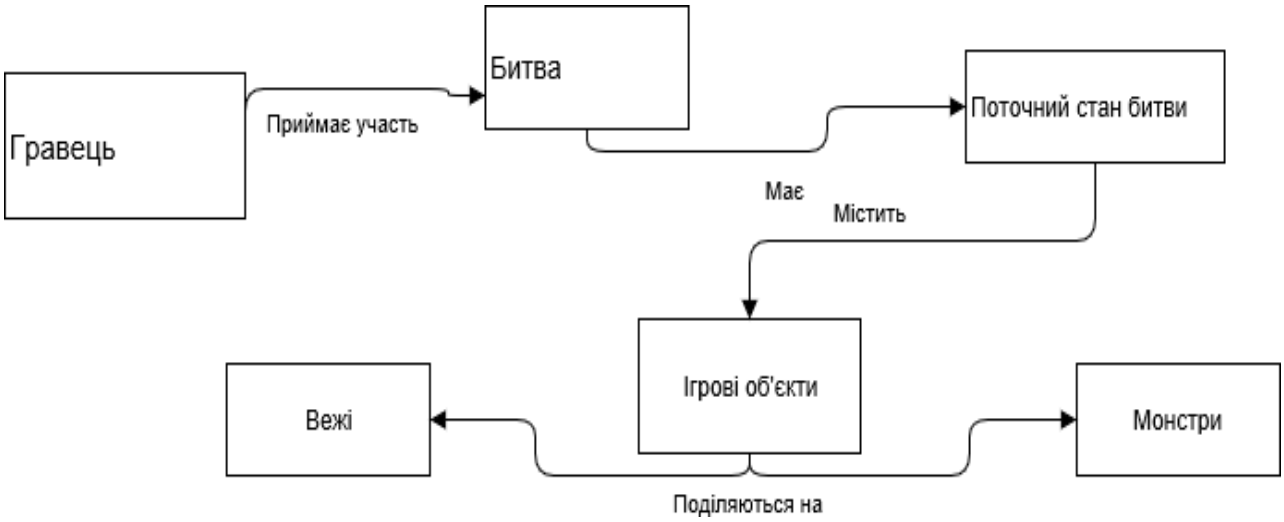

Рисунок 3.2 – Схема взаємозв’язку між об’єктами предметної області

Концептуальне моделювання використовують для загального огляду предметної області та створення концептуальної моделі, яка формує уявлення про структуру бази даних, який отриманий в результаті аналізу предметної області.

Тобто концептуальна модель - це опис структури бази даних, виконаний на природній мові, з використанням таблиць, діаграм та інших засобів. Модель повинна бути зрозумілою всім людям, що працюють з проектованою базою даних.

Для обраної ігрової системи основними поняттями, які повинні зберігатися в базі даних є користувач, бій, поточний стан бою, ігровий об’єкт, характеристики монстру та характеристики вежі. Щоб визначити допоміжні сутності необхідно провести аналіз цих понять та виявити їх характеристики.

Користувач системи матиме змогу реєструватися у системі, редагувати свої дані, розвиватись шляхом накопичення досвіду. Також користувач має змогу переглядати загальний рейтинг користувачів, та може приймати участь в бою, змагаясь при цьому з іншим користувачем.

Бій відбувається між двома користувачами, він має дату та час початку та дату та час кінцю.

Бій може біти завершеним та незавершеним. Завершений бій містить також інформацію про перемігшого гравця. Незавершений бій має певний стан.

Бій також може відбуватися між користувачем та комп’ютером. У такому випадку вважається що бій відбувається між гравцем та віртуальним гравцем, що має свій постійний унікальний ідентіфікатор гравця. Система знає та зберігає цей ідентифікатор.

Поточний стан бою являє собою серілізовану програмну сутність бою у JSON форматі. Також стан бою містить ідентифікатор версії стану, що змінюється кожен раз, коли змінюється серіалізований стан бою.

Сутність ігровий об’єкт - ідентифікатор об’єкту, що приймає участь в режимі бою. Ігрові об’єкти поділяються на монстрів, вежі та інші. Ігрові об’єкти, що відносяться до веж та монстрів мають ігрові характеристики.

Отже, основними сутностями системи є «Користувач», «Бій», «Поточний стан бою», «Ігровий об’єкт», «Характеристики монстру», «Характеристики вежі». У ході проектування між цими сутностями були виявлені наступні взаємозв’язки:

– бій відбувається рівно між двома користувачами;

– бій може мати один поточний стан, або жодного;

– ігровий об’єкт-вежа має одні характеристики вежі;

– ігровий об’єкт-монстр має одні характеристики монстру;

– ігровий об’єкт, що не є вежою, або монстром не має характеристик.

Отже, у ході аналізу та концептуального моделювання були виявлені основні сутності системи та їх атрибути, що мають стати базовими при моделюванні та розробці системи.

Сутність «Користувач» має наступні атрибути:

– унікальний ідентифікатор;

– прізвище;

– ім’я;

– email;

– логін;

– пароль;

– одиниці досвіду.

Сутність «Бій» повинна мати такі атрибути:

– унікальний ідентифікатор;

– ідентифікатор користувача, що грає на стороні монстрів;

– ідентифікатор користувача, що грає на стороні веж;

– ідентифікатор користувача, що переміг;

– дату та час початку;

– дату та час закінчення.

Сутність «Поточний стан бою» має такі атрибути:

– унікальний ідентифікатор бою;

– ідентифікатор версії стану;

– серіалізований стан бою (програмна сутність).

Сутність «Ігровий об’єкт» повинна мати такі атрибути:

– унікальний ідентифікатор;

– назва.

Сутність «Характеристики монстру» містить такі атрибути:

– унікальний ідентифікатор ігрового об’єкту - монстру;

– одиниці шкоди, що завдаються замку;

– швидкість руху;

– тип захисту;

– пріоритет вибору шляху;

– кількість одиниць здоров’я;

– ціна;

– ідентифікатор типу переміщення (наземний або повітряний);

–.особливі ефекти при переміщенні.

Сутність «Характеристики вежі» має такі атрибути:

– унікальний ідентифікатор ігрового об’єкту - вежі;

– сила атаки;

– швидкість атаки;

– пріоритет вибору цілі;

– тип атаки;

– дальність атаки;

– ціна;

– особливі ефекти при атаці.

Тут потрібно відзначити, що сутності «Характеристики монстру» та «Характеристики вежі» являють собою підстутності сутності «Ігровий об’єкт».

Сутності що стосуються даних, що використовуються математичними моделями це: «Поточний стан бою», «Характеристики монстру», «Характеристики вежі».

На рисунку 3.3 зображена логічна схема бази даних системи. Пунктирною лінією позначується логічне відношення включення сутності ігрових об’єктів у серіаьзованому стані.

Схема бази даних знаходиться в третій нормальній формі тоді і тільки тоді, коли вона знаходиться в другій нормальній формі і жоден з не ключовийх атрибутів не є транзитивно залежним від первинного ключа.

З представленої схеми видно, що всі її неключові атрибути функціонально залежні від цілих потенційних ключів, тобто вона відповідає другій нормальній формі.

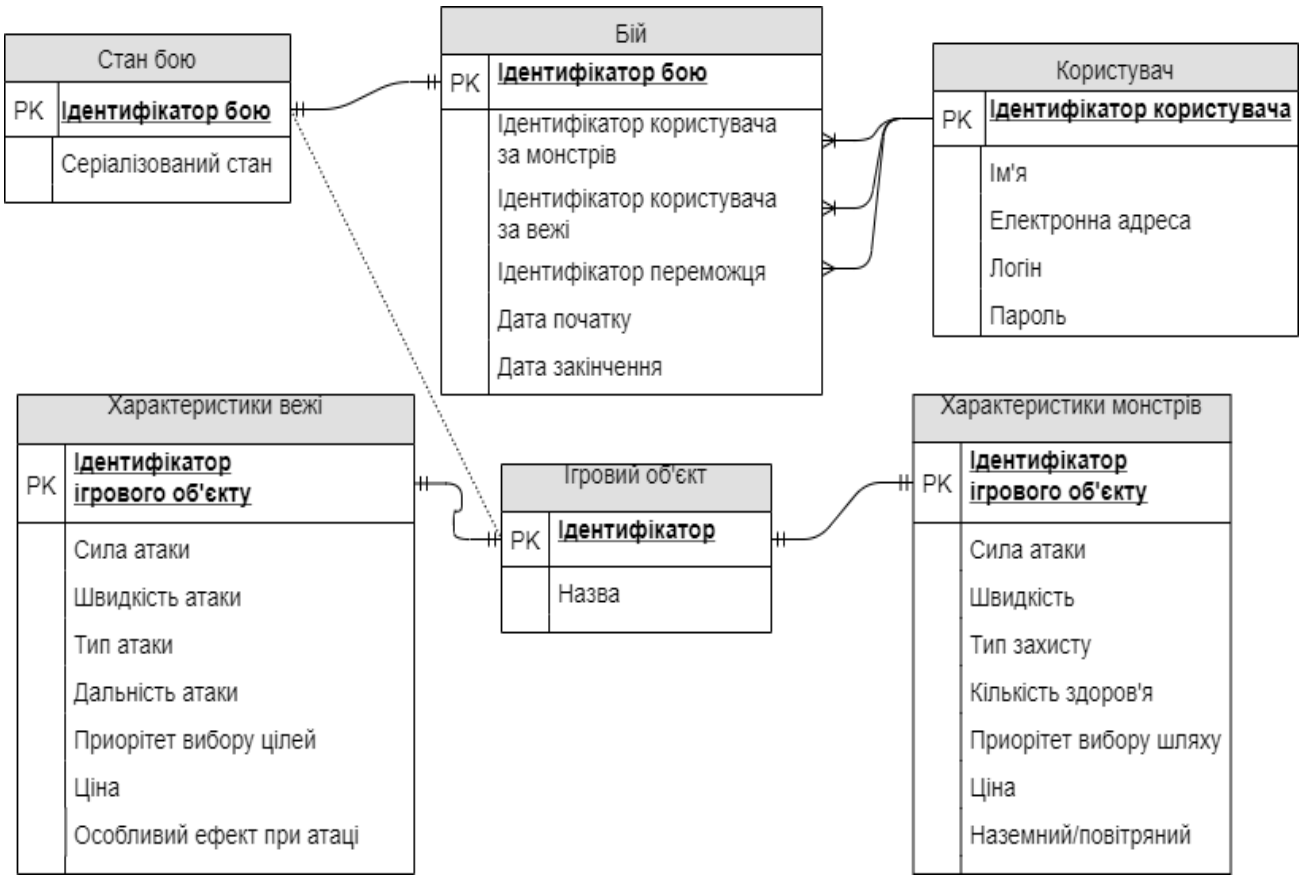

Рисунок 3.3 – Логічна схема бази даних

Крім того, кожен не ключовий атрибут не залежить один від одного, таких чином можна відзначити, що база даних відповідає третій нормальній формі.

Фізична реалізація бази даних буде складатись із двох фізічно незалежних серверів, що будуть зберігати різні об’єкти.

3.6 Розробка алгоритмів для вирішення промодельованих оптимізаційних задач

В практиці операційного менеджменту типовими для вирішення задач лінійного програмування є симплексний та транспортний методи, жадібний алгоритм, угорський алгоритм та інші. Симплек-метод – можна вважати базовим

для вирішення цього класу задач. Це алгебраїчна процедура, в результаті якої дослідник послідовно наближається до знаходження оптимального рішення. В теорії даний метод можна використовувати для вирішення задач, які включають будь-яку кількість обмежень і змінних, але якщо в них, більше чотирьох змінних чи обмежень, то обчислення стає досить складним.

Для вирішення задачі про призначення юнітів на шляхи та задачі про призначення веж на позиції було обрано розроблено та реалізовано два різних алгоритми на основі симплекс-методу та на основі жадібного алгоритму.

Симплексний метод - це алгебраїчна процедура, в результаті якої послідовно наближається до знаходження оптимального рішення.

В теорії даний метод можна використовувати для вирішення задач, які включають будь-яку кількість обмежень і змінних, але якщо в них, більше чотирьох змінних чи обмежень, то обчислення стає досить складним.

Транспортний метод представляє собою спрощений варіант симплексного методу. Він отримав таку назву, тому що широко застосовується для вирішення задач транспортування продукції з різних локацій в різні пункти призначення.

Жадібний алгоритм — евристичний алгоритм, досить простий і прямолінійний, який приймає найкраще рішення, виходячи з наявних на поточному етапі даних, не турбуючись про можливі наслідки, сподіваючись врешті-решт отримати оптимальне рішення.

Цей алгоритм був обраний для реалізації тому що, він є легким в реалізації і часто дуже ефективним за часом виконанн, та він не потребує значних ресурсів комп’ютера.

Для рішення задачі про призначення веж на позиції було обрано алгоритм на основі жадібного алгоритму.

На рисунку 3.4 наведено блок схему розробленого алгоритму для знаходження оптимального юніту для додавання на поле та оптимального шляху для нього.

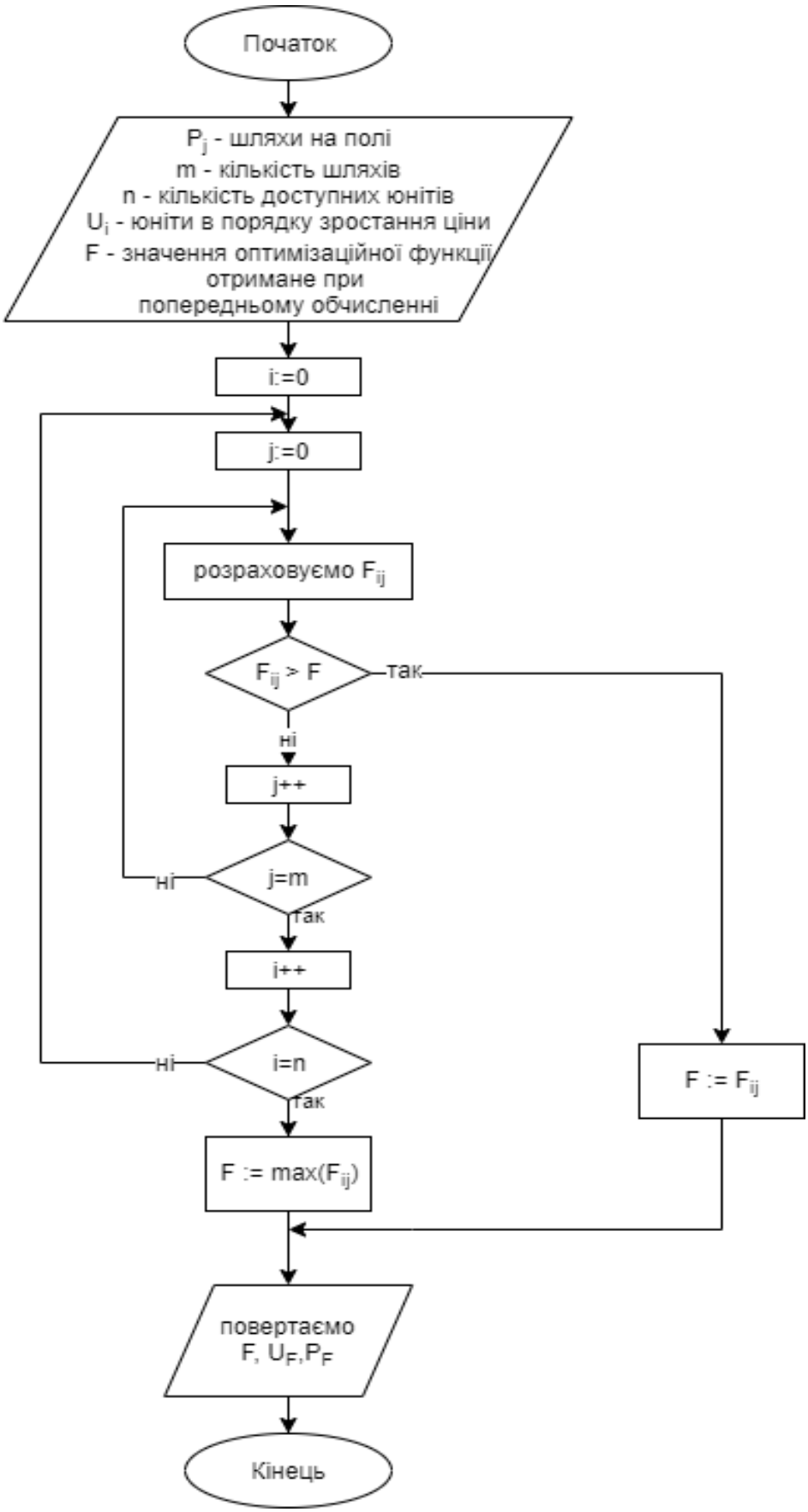

Рисунок 3.4 – Блок-схема алгоритму призначення юнітів на шляхи

На рисунку 3.5 наведено схему розробленого алгоритму для знаходження оптимальної вежі для додавання на поле та оптимальної позиції для неї.

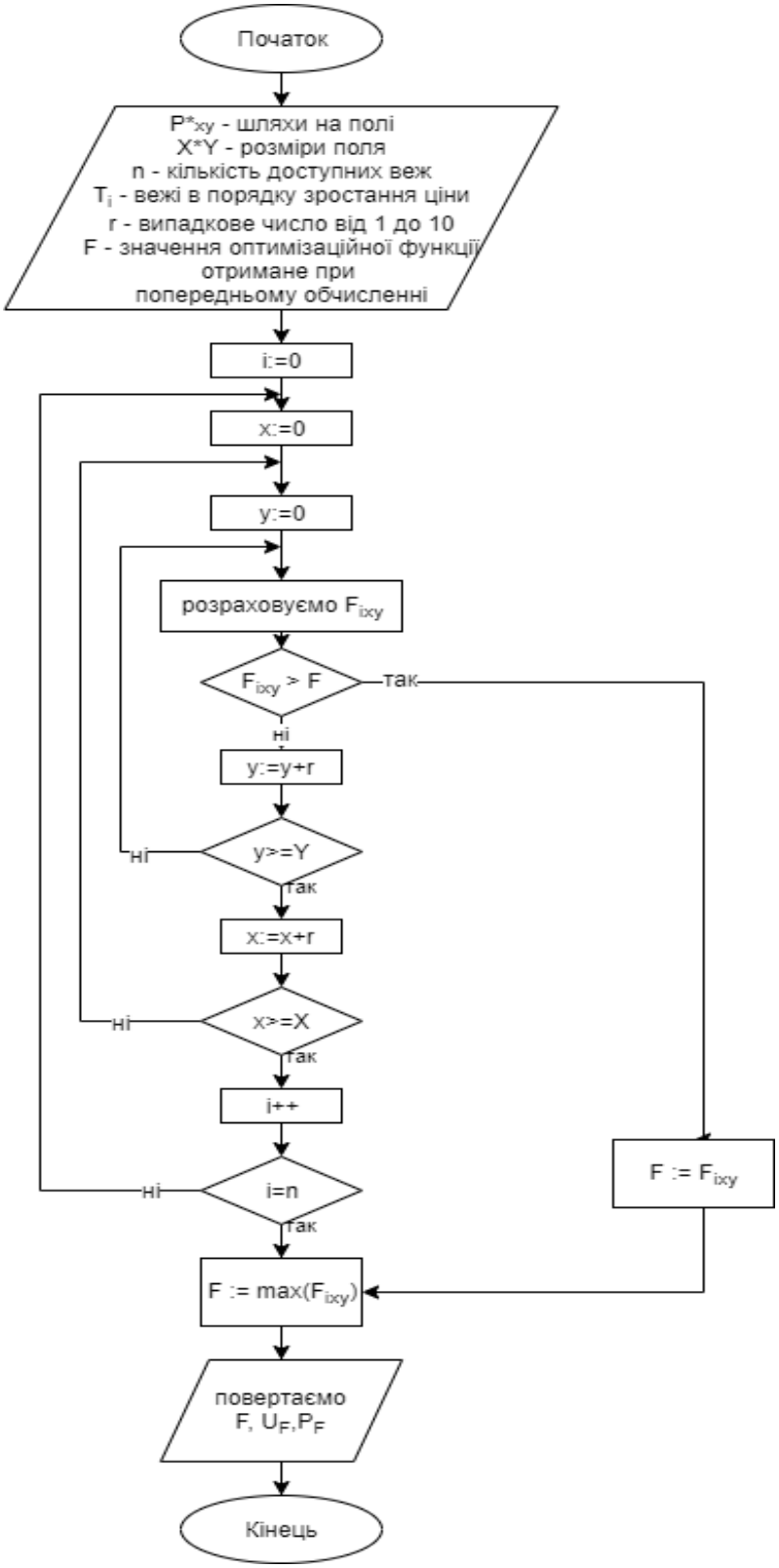

Рисунок 3.5 – Блок-схема алгоритму призначення веж на позиції

3.7 Проектування архітектури програмного забезпечення

Система «Програмний ігровий веб-додаток в жанрі «Tower Defence» складається з таких основних частин:

– серверу (ASP.NET Web Api);

– ігрового клієнту - настільного додатку(Unity Game engine);

– бази даних (MySQL).

Взаємодія компонентів системи представлена на діаграмі розгортання, що зображена на рисунку 3.6. На ній показані всі зв’язки між різними фізичними модулями системи.

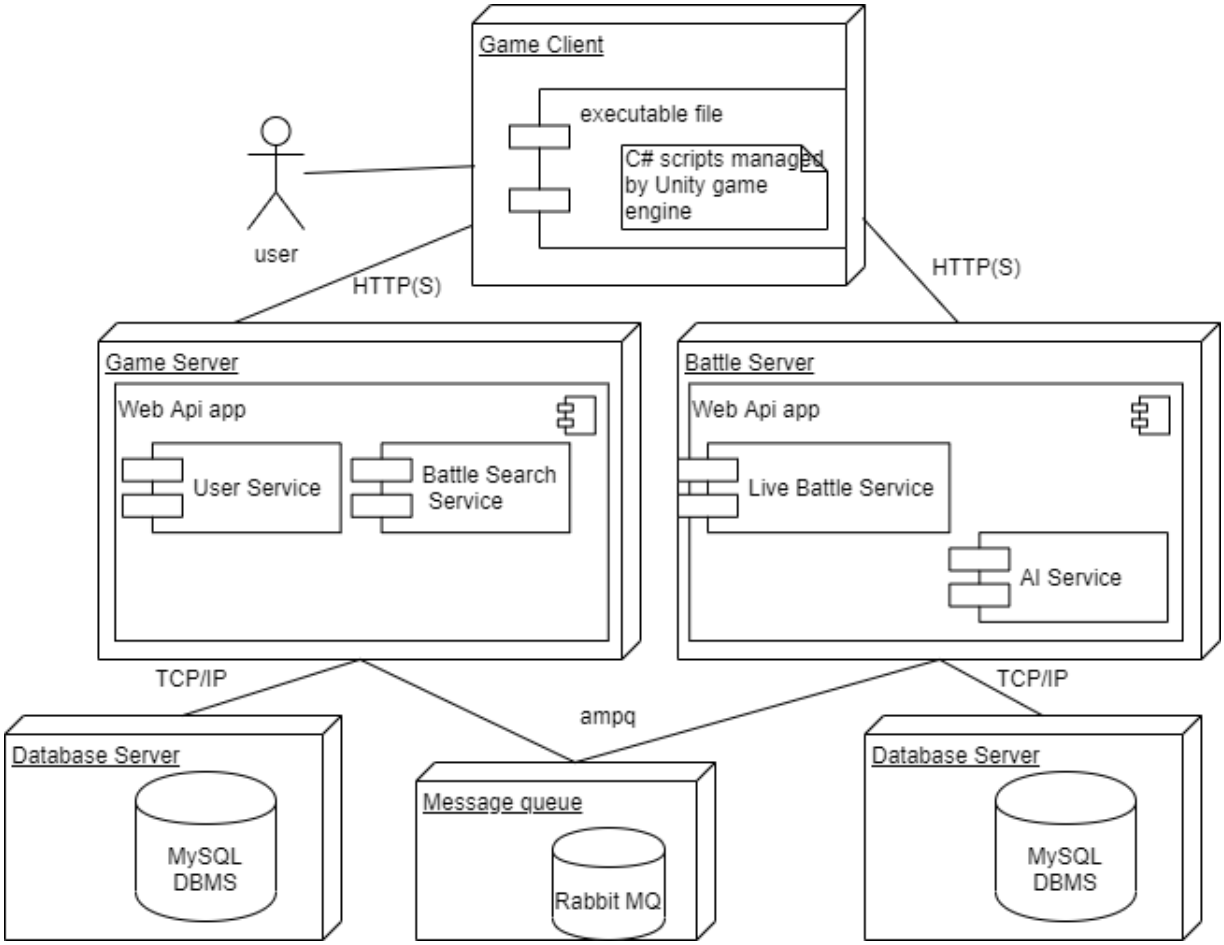

Рисунок 3.6 - Діаграма розгортання

Діаграма розгортання – діаграма в UML, на яка відображає основні обчислювальні вузли, які використовуються при роботі програми, компоненти, та об’єкти, що виконуються на цих вузлах. На діаграмі розгортання відображається фізичне представлення програмної системи разом та інформація про те, на якій платформі вона реалізована.

Ігровий клієнт, що являє собою настільний додаток, був побудований за допомогою платформи Unity [22], до якої входить інтегроване середовище розробки та ігровий рушій Unity 3D. Ігрові скрипти для рушію пишуться на мові C# (.NET). Обрана технологія дозволяє розробляти ігрові додатки дуже швидко та порівняно легко. Ігровий рушій повністю пов'язаний із середовищем розробки. Це дозволяє випробовувати гру прямо в редакторі.

Редактор Unity має простий Drag&Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Проект в Unity ділиться на сцени (рівні) — окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань.

Також Unity підтримує фізику твердих тіл. 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.

Недоліком даної технології є не дуже якісна робота з графічним відображенням та не оптимальне використання ресурсів, проте для розроблюємої системи дані недоліки ігрового рушія не є важливими.

Для реалізації веб-сервери системи була обрана технологія ASP.NET Web API. Web API представляє спосіб побудови програми на основі технології ASP.NET, дещо відмінний від ASP.NET MVC. Web API представляє собою веб-службу, яка може взаємодіяти з різними додатками. При цьому додаток може бути веб-додатком, мобільним або десктопних додатком.

Web API застосовує стиль REST (Representation State Transfer). REST-архітектура передбачає застосування запитів HTTP (GET, POST, PUT, DELETE) для взаємодії з сервером.

При розробці на даній технології код пишеться на мові програмування C#. Використання спільної мови для написання клієнтської та серверної частини дозволяє використовувати спільні моделі та спільну базу коду для цих частин.

ASP.NET надає можливість створення високопродуктивного та високонавантаженого веб-серверу, що є надійним та відмовостійким.

Для забезпечення обробки великої кількості паралельних запитів ASP.NET задає асинхронну модель запуску коду, засновану на обробці подій в неблокуючому режимі і визначенні обробників зворотніх викликів.

Для спрощення масштабування серверу систему він складається з двох фізично незалежних веб-сервісів. Для міжсервізної взаємодії використовується черга повідомлень на технології Rabbit MQ.

Rabbit MQ – технологія обміну повідомленнями між компонентами системи, що використовує комунікаційного стандарту AMPQ.

В якості сховища даних була обрана СКБД MySQL [23]. MySQL - реляційна система керування базами даних (СКБД).

Фундаментальна характеристика реляційної бази даних - це підтримка об’єктів і їх поведінки, включаючи типи даних, функції, операції, домени і індекси. Це робить MySQL неймовірно гнучким і надійним. Серед іншого, він вміє створювати, зберігати та видавати складні структури даних. Він забезпечує розширену ємність даних і надійність цілісності даних.

В якості менеджеру пакетів для ASP.NET використовувався nuget, який дозволяє керувати залежностями додатку та має велику базу пакетів готових для використання. Git був використаний для контрою версій під час розробки додатку. [24]

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Реалізація серверної частини

Для реалізації бази даних проекту було обрано MySQL [23] - реляційну систему керування базами даних.

Під час реалізації серверної частини було обрано технологію ASP.NET Core WebApi, що використовується для створення REST сервісів. Веб–сервіс системи складається з трьох основних частин – шару доступу до даних, шару бізнес–логіки та кінцевою точкою доступу для клієнтів (API).

Фізично сервер ігрової системи складається з двох окремо розгорнутих серверів, що являють собою веб-сервіси та використовують різні бази даних для зберігання своїх даних. Один сервіс займається адмініструванням користувачів та битв, авторизації, у той час, як інший – керування битвами у режіми реального часу.

Для взаємодії між веб-сервісами та обміну даних використовується черга повідомлень, яка реалізована з використанням технології Rabbit MQ.

Завдяки використанню веб-сервісу та багаторівневої архітектури досягається висока масштабованість системи в цілому та її компонентів окремо.

Усі алгоритми, що використовуються додатком для розрахунку ігрової логіки фізично працюють на ігровому сервері системи. Таким чином, ігровий сервер формує список дій для кожного моменту часу, які ігровий клієнт тільки показує графічно.

Фрагмент коду жадібного алгоритму для вирішення задачі про призначення ігрових об’єктів на ігрове поле наведено далі:

```
public (GameObjectType type, TDest destination)? GetOptimalVariant(Func<GameObjectType,
TDest, double> func)
{
  if (!_availableGameObjects.Any() || !_availableDestinations.Any())
  {
    return default;
  }
  var maxWeight = 0.0;
  var variants = new List<(GameObjectType type, TDest dest)>();
  foreach (var gameObject in _availableGameObjects)
```

```
{
  _availableDestinations.AsParallel().ForAll(dest =>
  {
    var weight = func(gameObject, dest);
    if (Math.Abs(weight - maxWeight) < double.Epsilon)
    {
      variants.Add((gameObject, dest));
    }

    if (weight > maxWeight)
    {
      variants.Clear();
      variants.Add((gameObject, dest));
      maxWeight = weight;
    }
  });
}
```

Далі наведено фрагмент коду ігрової логіки, в якому обирається оптимальний шлях для переміщення нападника (монстру). Вибір виконується на основі розрахунку оптимального шляху на основі багатьох критеріїв.

```
public int GetOptimalPath(Field field, Unit unit){
 var paths = field.StaticData.Path;
 var tableToAnalize = new double[paths.Length, _additiveConvolutionCalculator.NumberOfCriterias];
 for (int i = 0; i < paths.Length; i++){
   var towersOnPath = FindTowersThatCanAttackPath(field, i);
   var towers = towersOnPath.Select(t => field[t]).ToArray();
   var towersTypes = towers.Select(t => t.Type).ToArray();
   tableToAnalize[i, 1] = GetTotalAttackDamage(towersTypes, unit);
   tableToAnalize[i, 2] = GetAvgTowersRemoteness(paths[i], towers);
   tableToAnalize[i, 3] = GetTowersWithSpecialEffectRate(towersTypes);
 }
 return _additiveConvolutionCalculator.FindOptimalVariantIndex(tableToAnalize);
}

private double GetTotalAttackDamage(IEnumerable<GameObjectType> towerTypes, Unit unit){
 return 1 - (double)towerTypes.Sum(t =>_gameCalculator.CalculateDamage(unit.Type, t))/unit.Health;
}
private double GetAvgTowersRemoteness(Path path, ICollection<GameObject> towers){
 double remotenessSum = 0;
 foreach (var tower in towers)
 {
   var pos = path.First(point => _gameCalculator.IsTowerCanAttack(tower, point));
   remotenessSum += (double)path.PointOnThePathPosition(pos) / path.Length;
 }

 return remotenessSum / towers.Count;
}

private HashSet<int> FindTowersThatCanAttackPath(Field field, int pathId)
{
 var path = field.StaticData.Path[pathId];
 var towers = new HashSet<int>();
 foreach (var point in path)
   towers.UnionWith(field.FindTowersThatCanAttack(point, _statsLib));
  return towers;
```

4.2 Реалізація клієнтської частини

Для розробки ігрового клієнтського додатку було використано технологію Unity із графічним рушієм Unity 3D. Усі скрипти для ігрового дотатку були написані на мові програмування C#. Такий підхід дозволив використати спільні з серверною частиною моделі та фрагменти логіки на основі спільної бази коду. Для запитів на сервер був використаний стандартний клас ігрового рушія WWW. Цей клас дозволяє швидко та легко надсилати HTTP запити. Для передачі даних між клієнтом та сервером був використаний формат JSON. Ігровий клієнт в режимі битви працює за таким принципом: на початку битви клієнти обох гравців загружають дані поля битви. Далі, коли гравці змінюють стан битви, тобто додають на поле вежу, чи запускають монстра на сервер відправляється команда, яка свідчить про зміну стану, та сервер перерозраховує результат битви для кожного стану. Ігрові клієнти при цьому кожну секунду перевіряють, чи не змінився стан на сервері, та якщо змінився - клієнти загружають з серверу новий стан та порядок дій, які треба відобразити на полі. Таким чином реалізована синхронізація.

Нижче представлений фрагмент коду, що демонструє метод, який працює асинхронно та перевіряє, чи не змінився стан битви на ігровому сервері. Якщо стан змінився, метод відображає на клієнті новий стан та новий порядок дій.

```
private IEnumerator NetworkWorker()
{
  while (true)
  {
    var www = _gameProcessNetworkWorker.GetCheckBattleStateChange(_battleId, _revision);
    yield return www;
    if (!string.IsNullOrEmpty(www.text))  {
      var ticks = JsonConvert.DeserializeObject<ActionsResponseModel>(www.text);
      Field.SetState(ticks.State);
      if (_resolver != null){
        StopCoroutine(_resolver);
}
      RenderFieldState();
```

```
            _resolver = StartCoroutine(ResolveActions(ticks.ActionsByTicks));
        }
        yield return new WaitForSeconds(0.4f);
    }
```

На рисунку 4.1 зображено екранну форму геймплею режиму битви, на якій зображені ігрові об’єкти та основні елементи керування.

На представленому рисунку можна бачити головні ігрові об’єкти - вежі, що розташовані на узбіччі шляхів, та монстри, що рухаються по шляхам та мають змугу з індикатором здоров’я над ними.

Рисунок 4.1 – Екранна форма режиму битви

Вгорі екрану розташовані інформаційні панелі, що показують поточний рівень здоров’я замку, час, що залишився до закінчення битви та поточну кількість ресурсів гравця. Також вгорі розміщена панель вибору нового ігрового об’єкта (монстру чи вежі) з іконками. При виборі іконки об’єкта в лівій частині екрану з’являється панель характеристик обраного об’єкта.

4.3 Реалізація фізичної моделі бази даних

Для реалізації бази даних проекту було обрано MySQL [15] - реляційну систему керування базами даних.

Загальна схема фізичної моделі бази даних наведена на рисунку 4.2.

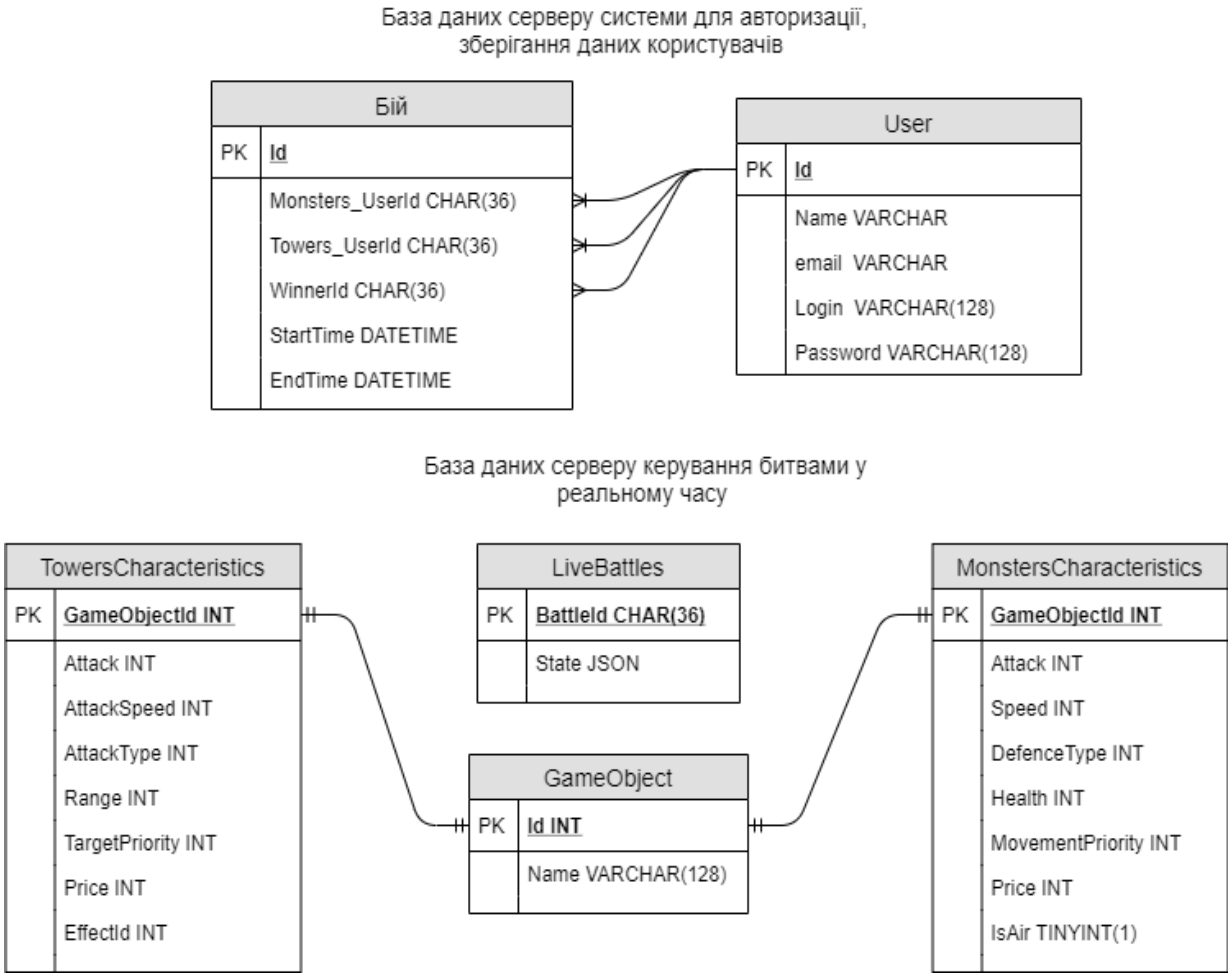

Рисунок 4.2 – Фізична модель бази даних

Потрібно зазначити, шо фізично база даних системи складається з двох фізічно незалежних серверів баз даних – ці сервери зберігають різні сутності ігрової системи, та обслуговують різні сервери.

5 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ МОДЕЛЕЙ ТА МЕТОДІВ

5.1 Планування експериментів

З метою стровення ефективного розвинутого ШІ було вирішено провести експериментальне дослідження поведінки ШІ під час призначення юнітів на шляхи або призначення веж на конкретні позиції на ігровому полі на базі розроблених математичних моделей. Під час планування серії експериментів використовувалися наступні передумови:

– базовий ШІ в серії експериментів не розглядався; його експериментальне дослідження було проведено окремо; дослідження показало, що хоча він і є найменьш вимогливими до обчислюваних ресурсів комп’ютера, проте не є дуже інтерактивними та не дає необхідного рівня перемог над супротивником-людиною;

– в якості високореактивного (або середнього) ШІ розглядалася поведінка ШІ на базі розробленої багатокритеріальної моделі вибору (1);

– в якості розвинутого ШІ розглядалася поведінка ШІ на базі розроблених моделей оптимізаційних задач про призначення.

Було заплановано 2 серії експериментів.

Під час цих першої серії експериментів необходимо дослідити наступні методи:

– призначення юнітів на шляхи шляхом багатокритеріального вибору на базі лінійної адитивної згортки (АЗ1);

– призначення юнітів на шляхи на основі жадібного алгоритму (ЖА1);

– призначення юнітів на шляхи на основі симплекс-методу підрахунку значень оптимізаційної моделі (БА1).

Під час цих другої серії експериментів необхідно дослідити наступні методи:

– призначення веж на клітини шляхом багатокритеріального вибору на базі лінійної адитивної згортки (АЗ2);

– призначення веж на клітини на основі жадібного алгоритму (ЖА2);

– призначення веж на шляхи на основі симплекс-методу (БА2).

Експерименти необхідно проводити на іграх різною розмірності:

– Ігра1: кількість юнітів - 5, розмір поля 10Х10, кількість веж 3, кількість шляхів – 2;

– Ігра2: кількість юнітів - 8, розмір поля 20Х20, кількість веж 5, кількість шляхів – 5;

– Ігра3: кількість юнітів - 10, розмір поля 50Х50, кількість веж 7, кількість шляхів – 10.

Для кожної серії експериментів було заплановано по 100 ігрових сесій, у яких ефективність обраних методів перевірялася за допомогою наступних показників:

– процентне співвідношення перемог ШІ по відношенню до іншої сторони;

– використання ресурсів обчислюючого пристрою, а саме час у мілісекундах, затрачуваний для прийняття одного рішення;

– кількість використаної памяті.

Усі дослідження необхідно проводити на одному фізичному сервері, який має наступні характеристики:

– центальний процесов Intel Xeon E-2224;

– частота процесору 3,4 Ghz;

– кеш 8 Mb;

– 4 ядра (8 логічних ядер);

– ОЗУ 8 Gb 2666 MHz.

При цьому дослідження на інших фізічних серверах хоч і є доцільним, проте не є обов’язковим, адже в експериментах головним чинов досліджувались та виявлялись закономірності.

5.2 Експериментальне дослідження

Для того, щоб забезпечити достаньо високу кількість серій дослідження, усі бої відбувались автоматично та без участі людини, тобто ШІ змагався сам з собою на базі розроблених моделей.

На серії зі 100 ігор в якості супротивника для ШІ в рівних пропорціях використовувалися всі 3 розроблені моделі. Це дало змогу змоделювати досить досвітченого гравця.

Результати дослідження першої серії експериментів для «Ігра1», «Ігра2» та «Ігра3» наведено в таблицях 5.1, 5.2 та 5.3 відповідно.

Таблиця 5.1 - Результати першої серії експериментів для «Ігра1»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середне використання оперативної пам’яті, Кб	Процент перемог монстрів, %
АЗ1	9	312	44
БА1	82	1016	48
ЖА1	51	956	48

Таблиця 5.2 - Результати першої серії експериментів для «Ігра2»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середне використання оперативної пам’яті, Кб	Процент перемог монстрів, %
АЗ1	32	425	42
БА1	196	1598	47
ЖА1	117	1056	46

Таблиця 5.3 - Результати першої серії експериментів для «Ігра3»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середне використання оперативної пам’яті, Кб	Процент перемог монстрів, %
АЗ1	183	531	39
БА1	567	1900	46
ЖА1	479	1576	45

Таким чином, можна зробити висновок, що результати достатньо закономірні та для ігор різної розмірності модифікований жадібний алгоритм є найбільш оптимальним, адже він дає значний вигріш у продуктивності, та майже не програє у ефективності у порівнянні із базовим.

При цьому метод лінійних адитивних згорток хоч і використовує значно меньше ресурсів, він не дає достатнього рівня ефективності, тому його викостистання є доцільним лише в разі значних обмежень по ресурсах.

Друга серія експериментів була присвячена дослідженні алгоритмів для призначення вежі до позиції.

Результати дослідження другої серії експериментів для «Ігра1», «Ігра2» та «Ігра3» наведено в таблицях 5.4, 5.5 та 5.6 відповідно.

Таблиця 5.4 - Результати другої серії експериментів для «Ігра1»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середне використання оперативної пам’яті, Кб	Процент перемог веж, %
БА2	801	7011	52
ЖА2	591	5678	53
АЗ2	54	988	47

Таблиця 5.5 - Результати другої серії експериментів для «Ігра2»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середне використання оперативної пам'яті, Кб	Процент перемог веж, %
БА2	2570	8329	55
ЖА2	1961	6571	55
АЗ2	870	1492	43

Таблиця 5.6 - Результати другої серії експериментів для «Ігра3»

Алгоритм/Показники	Середній час однієї роботи алгоритму, мс	Середне використання оперативної пам'яті, Кб	Процент перемог веж, %
БА2	4132	9221	56
ЖА2	3006	7221	55
АЗ2	1081	1912	41

Для оцінки похибки вимірювань були обчислені середьоквадратичні відхилення. Для нормального закону розподілу оцінка генерального середньоквадратичного відхилення (S) результатів спостережень визначається:

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{X})}$$

де n – кількість експериментів для кожного окремого випадку, тобто 100 в нашому випадку,

x_i – результат і-го експерименту,

$\bar{X}$ – середнє арифметичне.

Значення середньоквадратичного відхилення для показнику середнього часу однієї роботи алгоритму для всіх серій експериментів склало не більше 70

мс, при цьому чим більший показник, тим більше його середньоквадратичне відхилення, тобто цей показник є достатньо нестабільним та залежить від багатьох факторів у різні моменти часу.

Значення середньоквадратичного відхилення для показнику середнього використання оперативної пам’яті програмою для всіх серій експериментів склало не більше 60 Кб, тобто цей показник є остатньо стабільним, але тут потрібно відзначити, що оперативна пам’ять використовується програмою не тільки для досліджуваних обчислень.

Значення середньоквадратичного відхилення для показнику проценту перемог не перебільшує 0,5 %, тобто цей показник є достатньо ефективним.

За результатами проведених експериментів можна зробити висновок, що результати достатньо закономірні. Для ігор різної розмірності модифікований жадібний алгоритм є найбільш ефективним, адже він дає значний вигріш у продуктивності порівняно зі згортковим та майже не програє у ефективності у порівнянні із базовим.

При цьому метод лінійних адитивних згорток хоч і використовує значно меньше ресурсів, але не дає достатнього рівня ефективності. Модель, побудована для поведінки ШІ на стороні захисників (веж), показала більш значні обчислювальні витрати у зв’язку з її складністю, що зумовлено значним ігровим навантаженням таких юнітів, як вежі. Між тим, рішення з робудови захисних веж, за своєю природою, є більш стратегічними, отже отримані часові результати є цілком сприйнятними в іграх цього жанру.

Звичайно, відсутність реальної людини призвела до певного зниження процентного співвідношення перемог ШІ по відношенню до такого більш сильного супротивника (також ШІ), але в сукупності дозволило отримати цілком адекватні результати.

На основі проведених серій експериментів сформулюємо рекомендації стосовно використання розвиненого та середнього ШІ в іграх жанру Tower Defense:

– модель поведінки ШІ, що побудована на базі багатокритеріальної згортки, на іграх будь-якої розмірності дае значну економію з боку обчислувальних ресурсів, але уступає іншим моделям з точки зору перемог і, взагалі, не дозволяє перетнути поріг в 50%. Такі показники дозволяють рекомендувати модель для реалізації середнього інтелекту, але не рекомендується для реалізації ШІ для вірня «Експерт»;

– не лінійна залежність зростання обчислювальних ресурсів від розмірності ігри, між тим, дозволяє рекомендувати модель поведінки ШІ на базі багатокритеріальної згортки (алгоритми АЗ1 та АЗ2) для ігрових полів надвеликих розмірів (експерименти для ціх типів полів не проводились); така модель надає задовільний рівень ефективності рішень та має значно кращу обчислювальну швидкість;

– модель поведінки ШІ з використанням базового алгоритму для свого рішення (БА1 та БА2), взагалі, не рекомендовано для реалізації, оскільки майже на такому ж рівні перемог (+1%), як і при використання жадібного алгоритму (ЖА1 та ЖА2), воотримує значно гірші обчислівальні показники;

– найкращим вариінтом для реалізації розвинутого інтелекту (для ігрових полів середніх та великих розмірів, як «Ігра2» та «Ігра3») є модель поведінки ШІ з використанням жадібного алгоритму (ЖА1 та ЖА2), адже вони надають гарний рівень перемог, та мають пристойну обчислювальну швидкість.

ВИСНОВКИ

В результаті атестаційної роботи магістра було запропоновано математичні моделі поведінки штучного інтелекту для гри-стратегії у жанрі Tower Defense, що побудовано на базі оптимізаційних моделей теорії прийняття рішень:

– багатокритеріальна згорткова модель для поведінки ШІ під час призначення нападників (монстрів) на ігрові шляхи та захисних веж на клітини ігрового поля;

– оптимізаційна модель задачі про призначення для поведінки ШІ з боку нападника: під час запуску монстра на ігрове поле по певному шляху;

– оптимізаційна модель задачі про призначення для поведінки ШІ з боку захисника: під час розташування захисних веж на шляхах карти.

Розроблено алгоритми вирішення задач на базі розроблених моделей, що демонструють різний рівень складності обчислень: на базі аддитивної лінійної згортки, жадібного алгоритму та симплекс-методу.

Реалізація проекту відбувалася за допомогою мови програмування C#, JSON як формат передачі даних протоколом HTTP, Під час роботи використовувалися JetBrains Rider, Visual Studio 2017 та Unity. У якості СКБД виступає MySQL.

В процесі виконання роботи були закріплені теоретичні знання та навички розробки програмного продукту, починаючи з проектування і закінчуючи тестуванням програмного продукту, були підвищені знання в розробці програм, пов’язаних з платформою ASP.NET та технологією Unity.

Був проведений аналіз проблемної галузі, проведено аналіз ринку подібних систем та аналіз існуючих досліджень.

В ході атестаційної роботи магістра було:

– розроблено механіку та принципи гри в жанрі "Tower Defence";

– проведено аналіз опитізаційних моделей та методів з метою створення штучного інтелекту;

– розроблено математичні моделі поведінки ШІ у грі в жанрі Tower Defense на базі оптимізаційних задач теорії прийняття рішень;

– розроблено алгоритми вирішення оптимізаційних задач на базі розроблених моделей;

– виконано програмну реалізацію ігрової системи;

– були спланувані експериментальні дослідження розроблених методів та моделей;

– за результатами проведених експериментів були розроблені рекомендації стосовно використання обраного підходу для створення розвинутого ШІ в іграх у жанрі Tower Defense.

За результатами атестаційної роботи було розроблено презентацію (див. додаток А).

Були опубліковані тези доповіді на молодіжному форумі «Радіоелекторніка та молодь у XXI ст». Тези наведені у додатку Б.

Була подана стаття «A Study of Optimization Models for Creation of Advanced Artificial Intelligence for the Computer Game in the Tower Defense Genre». Стаття була подпнп до науково-технічного семінару «The Third International Workshop on Computer Modeling and Intelligent Systems (CMIS-2020)» що індексується наукометричною базою Web of Science (Core Collection ESCI). Статтю наведено у додатку Г.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Вільна енциклопедія Википедия [Електронний ресурс] / Мережева енциклопедія Wikipedia. 2000. – Forefront TMG 2010: Режим доступу: www/ URL : http://ru.wikipedia.org/

2. Jensen, B., Togelius, J., Nielsen, J., AI for General Strategy Game Playing. — ResearchGate, 2014. URL: https://www.researchgate.net/publication/ 275963230

3. Nareyek A. AI in computer games//Queue. 2004. No 10 (кн. 1). С. 58–65. URL: http://doi.acm.org/10.1145/971564.971593

4. Yannakakis G., Togelius J. Artificial Intelligence and Games. — Springer, 2018. URL: http://gameaibook.org.

5. Yildirim S., Stene S. A survey on the need and use of AI in game agents//Proceedings of the 2008 Spring Simulation Multiconference, ser. SpringSim'08/ Сан-Диего, CA, США: Society for Computer Simulation International, 2008. С. 124–131. URL: http://dl.acm.org/citation.cfm?id=1400549.1400565

6. Spronck P., Ponsen M., Sprinkhuizen-Kuyper I., Postma E. Adaptive game AI with dynamic scripting//Machine Learning. 2006. No 3 (кн. 63). С. 217–248. URL: http://dx.doi.org/10.1007/s10994-006-6205-6

7. Csikszentmihalyi M. Flow: The Psychology of Optimal Experience. —Нью-Йорк, NY, США: Harper and Row, 1990. — 456 с.

8. Bakkes S., Spronck P., van den Herik J. Opponent modelling for case-based adaptive game AI//Entertainment Computing. 2009. No 1 (кн. 1). С. 27–37.

9. Champandard A. 10 reasons the age of finite state machines is over. 2008. URL: http://aigamedev.com/open/article/fsm-age-is-over/

10. Vlasenko L., Chikrii A. On a differential game in a system with distributed parameters // Proceedings of the Steklov Institute of Mathematics, 2016, vol. 292 (1), P. 276-285 URL: https://link.springer.com/article/10.1134/ S0081543816020243

11. Millington I., Funge J. Artificial Intelligence for Games, 2nd ed. — Сан-Франциско, CA, США: Morgan Kaufmann Publishers Inc., 2009. — 896 с.

12. Bevilacqua F. Finite-state machines: Theory and implementation. 2013. URL: https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation-gamedev-11867

13. Лотов А.В., Поспелова І.І. Багатокритеріальні задачі прийняття рішень: Навчальний посібник, 197 с. – 2008.

14. Романьков, В.А. Вступ в теорію ігор: Навчальний посібник / Романьков В А. Москва, 2016. – 834 с.

15. Матузко Ю.О. Теорія прийняття рішень. Навчально-методичний посібник для студентів всіх спеціальностей / Матузко Ю.О. Запоріжжя, 2009. – 70 с.

16. Bertsimas, D. and Tsitsiklis, J. N. Introduction to Linear Optimization, 608 с. — 1997.

17. Волченська Т.В., Князьков В.С. Комп’ютерная математика: Частина 2. Теорія графів: Навчальний посібник. Пенза, 2002. – 101 с.

18. Playground [Електронний ресурс] Режим доступу: www/ URL: https://www.playground.ru/misc/sovetuem_poigrat_luchshie_igry_v_zhanre_tower_defense-134511 – Загол. з екрану.

19. Фаулер М., Дейвид Р. Архітектура корпоративних програмних додатків / Мартин Фаулер, Дейвид Райс; Москва, 2010. – 544 с.

20. Лотів А.В., Поспєлова І.І. Багатокритеріальні задачі прийняття рішень: Навчальний посібник. Москва, 2008. – 197 с.

21. Мазурова О.А. Метод автоматизированной поддержки формирования множества критериев для задач принятия решений // Вісник Національного Технічного університету "ХПІ". Тематичний збірник наукових праць "Нові рішення у сучасних технологіях". Харків: НТУ "ХПІ". 2001. № 15. с. 36-42

22. Хокінг Дж. Unity в дії. Мультиплатформенна розробка на C# СПб, 2016. – 336 с.

23. Белл Ч., Кіндай М., Талманн Л., Забезпечення високої доступності систем на основі MySQL СПб, 2012. – 624 с.

24. Scott Chacon. Pro Git / Scott Chacon Москва, 2015. – 1012 с.