Міністерство освіти та науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Системотехніки
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

ГЮИК 504310.008 ПЗ
(позначення документа)

Розробка компонентів мультиагентної системи з оптимізацією пошуку шляху в динамічному оточенні
(тема)

Виконав:
Студент 2 курсу, групи *СПРм-19-1*
Спеціальність *122 – Комп'ютерні науки*
(код і повна назва напряму)
Тип програми *освітньо-наукова*
(освітньо-професійна або освітньо-наукова)
Освітня програма *Системне проектування*
(повна назва освітньої програми)

Кудь О. Ю.
(прізвище, ініціали)

Керівник *проф. Мінухін С. В.*
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри системотехніки _____ *Гребеннік І. В.*
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність *122 – Комп'ютерні науки*
(код і повна назва)

Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Системне проектування*
(повна назва)

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

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Кудю Олександру Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *«Розробка компонентів мультиагентної системи з оптимізацією пошуку шляху в динамічному оточенні»*
затверджена наказом по університету від «_02_» __11_____ 2020 р. №___1516Ст_____
2. Термін подання студентом роботи (проекту) 24.12.2020 р.
3. Вихідні дані до роботи (проекту) теоретичні відомості про аналіз даних у предметній області мультиагентного пошуку
4. Зміст пояснювальної записки (перелік питань, що потрібно розробити)

4.1 *Аналіз предметної області системи мультиагентного пошуку шляхів.* ***4.2*** *Аналіз класичного мультиагентного пошуку шляхів.* ***4.3*** *Аналіз вирішувачів мультиагентного пошуку шляхів.* ***4.4*** *Аналіз MAPF у динамічному оточенні.* ***4.5*** *Постановка задачі.* ***4.6*** *Постановка задачі.* ***4.7*** *Розробка системних вимог до компонентів інформаційної системи.* ***4.8*** *Визначення функціональних вимог до компонентів інформаційної системи.* ***4.9*** *Розробка моделі потоків даних компонентів інформаційної системи* ***4.10*** *Діаграма варіантів використання компонентів інформаційної системи.* ***4.11*** *Діаграма класів.* ***4.12*** *Діаграми послідовностей.* ***4.13*** *Діаграми кооперацій системи.* ***4.14*** *Діаграма станів системи.* ***4.15*** *Діаграма діяльності системи.* ***4.16*** *Розробка структури додатку.* ***4.17*** *Обґрунтування вибору мови програмування та середовища розробки.* ***4.18*** *Розробка функціонального ядра.* ***4.19*** *Розробка верхнього рівня пошуку шляху.* ***4.20*** *Обґрунтування вибору алгоритму для низькорівневого пошуку шляху.* ***4.21*** *Оптимізація розрахунків під час симуляції.* ***4.22*** *Висновки* ***4.23*** *Перелік посилань* ***4.24*** *Додатки.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів) 5.1 Контекстна діаграма IDEF0 (1 аркуш формату А4). 5.2 Декомпозиція першого рівня в нотації IDEF0 (1 аркуш формату А4). 5.3 Декомпозиція функції «Виконання симуляції переміщення» в нотації IDEF0 (1 аркуш формату А4). 5.4 Контекстна діаграма в нотації DFD (1 аркуш формату А4). 5.5 Діаграма декомпозиції першого рівня в нотації DFD (1 аркуш формату А4). 5.6 Діграма варіантів використання (Use Case) (1 аркуш формату А4). 5.7 Діграма класів компонентів системи (1 аркуш формату А4). 5.8 Діграма послідовностей варіанту «Вірішення конфліктів» (1 аркуш формату А4). 5.9 Діграма кооперацій варіанту «Вірішення конфліктів» (1 аркуш формату А4). 5.10 Діграма станів системи (1 аркуш формату А4). 5.11 Діграма активностей системи (1 аркуш формату А4).

6. Консультанти розділів роботи (проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на атестаційну роботу	02.11.20	
2	Аналіз завдання, підбір літератури	02.11.20-05.11.20	
3	Аналіз літератури з досліджуваної проблеми	05.11.20-08.11.20	
4	Аналіз технічних засобів для реалізації	08.11.20-10.11.20	
5	Проектування системи	10.11.20-15.11.20	
6	Програмна реалізація	15.11.20-30.11.20	
7	Оформлення пояснювальної записки	30.11.20-08.11.20	
8	Перевірка на плагіат	20.12.20	
9	Представлення на рецензування	22.12.20	
10	Представлення атестаційної роботи в ДЕК	24.12.20	

Дата видачі завдання 02 11 2020 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Мінухін С. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Атестаційна робота: 66 аркушів, 34 рисунка., 2 таблиці., 1 формула, 28 джерел, 2 додатки.

В цій роботі об’єктом дослідження є процес автоматизації мультиагентного пошуку шляхів у динамічному просторі. Предметом дослідження визначено методи розробки компонентів інформаційної системи мультиагентного пошуку шляхів. Для розробки проекту було використано такі методи:

– аналіз сфери мультиагентного пошуку шляхів;
– огляд існуючих алгоритмів;
– функціональне моделювання системи;
– розробка програмної системи.

Метою роботи є розробка компонентів мультиагентної інформаційної системи пошуку шляхів у динамічному оточенні.

Система є комплексною і складається з декількох підсистем. Архітектура системи описана за допомогою відповідних діаграм.

Одною з основних вимог до системи є можливість власноруч задавати параметри рухомих об’єктів та карти, на основі якої будується граф. В ході роботи було розроблено компоненти інформаційної системи під платформу Windows.

Система була розроблена на основі завдання на атестаційну роботу і згодом може бути використано в реальних умовах. Компоненти системи можуть використовуватись як підсистеми у більших системах.

ІНФОРМАЦІЙНА СИСТЕМА, ПОШУК ШЛЯХУ, МУЛЬТИАГЕНТНА СИСТЕМА, МУЛЬТИАГЕНТНИЙ ПОШУК ШЛЯХУ, СИСТЕМНЕ ПРОЕКТУВАННЯ, ПОШУК ШЛЯХУ В ДИНАМІЧНОМУ ОТОЧЕННІ, C++, MAPF, D-MAPF, CBS , EPEA

ABSTRACT

Certification work: 66 sheets, 34 figures., 2 tables., 1 formula, 28 sources, 2 appendices.

In this work, the object of research is the process of automating multi-agent pathfind in dynamic space. The subject of the research is the methods of development of components of the information system with multi-agent pathfind. The following methods were used to develop the project:

- analysis of the sphere of multiagent pathfind;
- review of existing algorithms;
- functional modeling of the system;
- software system development.

The aim of the work is to develop components of a multi-agent information system for finding paths in a dynamic environment.

The system is complex and consists of several subsystems. The architecture of the system is described by the corresponding diagrams.

One of the main requirements for the system is the ability to manually set the parameters of moving objects and the map on which the graph is built. During the work, the components of the information system for the Windows platform were developed.

The system can later be used in real conditions. System components can be used as subsystems in larger systems.

INFORMATION SYSTEM, PATHFIND, MULTIAGENT SYSTEM, MULTIAGENT PATHFIND, SYSTEM DESIGN, C++, MAPF, D-MAPF, CBS , EPEA

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

MAPF – мультиагентний пошук шляхів.
CBS – пошук, заснований на конфліктах.
ID – виявлення незалежностей.
OD – операторний розклад.
ICTS – дерево зростання витрат.
CT – дерево конфліктів.
EPEA* - розширене часткове розширення A*.
D-MAPF – динамічний мультиагентний пошук шляхів.
PDB – бази даних шаблонів.
GPS - система глобального позиціонування.
UML - уніфікована мова моделювання.
FSM - скінченний автомат.
IRIS GL - графічна бібліотека інтегрованої системи растрової графіки.
OpenGL - відкрита графічна бібліотека.
GLSL - графічна бібліотека шейдерної мови.
GLM - математика OpenGL.
JSON - об'єктна нотація JavaScript.
SOIL2 - проста бібліотека зображень OpenGL 2.
BMP - растрове зображення.
XML - розширювана мова розмітки.

ВСТУП

Протягом довгих років людство приділяє особливу увагу розвитку інформаційних технологій. Спеціальні технології створюються не тільки для зручності людства, але і для поліпшення життя людей. За допомогою ІТ люди пізнають себе і світ в цілому. На початку XXI століття людська раса розвиває спеціальні технології значно швидше, ніж в попередньому XX столітті, який ознаменований відкриттям обчислювальної машини, комп'ютерів, електронно - променевої трубки і багато іншого.

Особливо швидко розвиваються комп’ютерні ігри, робототехніка, системи автоматизації складів, навігаційні систем тощо. Наприклад, пошук шляху в контексті комп'ютерних ігор стосується шляху, на якому об'єкт, що рухається шукає шлях навколо перешкод. Найбільш часто завдання пошуку шляху виникає в стратегіях реального часу, в яких гравець дає завдання ігровим юнітам рухатися через ігровий рівень, який містить перешкоди. Крім стратегій, завдання пошуку шляху, так чи інакше, в тій чи іншій мірі зустрічається в більшості сучасних ігрових жанрів.

Багатоагентний пошук шляхів (MAPF) - це важливий тип проблеми планування, в якій завдання полягає в плануванні шляхів для кількох агентів, де ключовим обмеженням є те, що агенти можуть йти цими шляхами одночасно, не стикаючись між собою. MAPF має низку відповідних сучасних застосувань, включаючи автоматизовані склади, автономні транспортні засоби та робототехніку. За останні роки цій проблемі приділяють увагу різні дослідницькі групи та академічні спільноти по всьому світу.

Кожен агент може розглядатися як динамічна перешкода для інших агентів. Перешкоди та агенти призводять до певних обмежень при виконанні плану. Під час виконання обчислюваного плану MAPF в динамічному середовищі (наприклад, на складі, який не є повністю автономним) можуть відбутися зміни: існуючі перешкоди можуть бути усунені із середовища або переміщені в інше місце в середовищі, існуючі агенти можуть залишити вийти із середовища, або нові агенти можуть бути включені в команду з новими завданнями. Тоді мета полягає у пошуку нового рішення для нової команди агентів у модифікованому середовищі. Це задача багатоагентного пошуку шляхів у динамічному середовищі.

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз класичного мультиагентного пошуку шляхів

Багатоагентний пошук шляху (MAPF) - складне завдання для багатьох практичних застосувань в робототехніці, відеоіграх, маршрутизації транспортних засобів тощо. Приклади завдання складаються з графа G = (V, E) і агентів. У кожного агента є стартова позиція і кінцева позиція. Завдання полягає в тому, щоб без зіткнень перемістити всіх агентів до їх цілей, мінімізуючи функцію сукупних витрат. У своїй загальній формі MAPF є NP-повною задачею, тому що це узагальнення головоломки ковзної плитки, яка є NP-повною. Через складність проблеми велика частина досліджень була зосереджена на децентралізованих підходах, які можуть повертати неоптимальні рішення і, в деяких випадках, не є повними.

Вхідні дані MAPF: граф G (V, E) і k агентів, позначених як a_1, a_2 … $a_к$. Кожен агент a_i пов'язаний з вершинами початку і цілі: s_i і g_i. У початковий момент часу t = 0 кожен агент аі знаходиться в місці розташування s_i. Між послідовними точками часу кожен агент може виконати дію з переміщення (move) в сусіднє місце розташування або може залишатися в режимі очікування (wait) в своєму розташуванні. Обмеження полягає в тому, що кожна вершина може бути зайнята не більше ніж одним агентом в даний момент. Крім того, якщо a і b є сусідніми вершинами, два різних агента не можуть одночасно перетнути з'єднуючу кромку в протилежних напрямках (від a до b і від b до a). Однак агентам дозволено слідувати один за одним, тобто агент a_i може переміщатися від x до y одночасно з переміщенням агента a_j від y до z. Завдання полягає в тому, щоб знайти послідовність дій {move, wait} для кожного агента таким чином, щоб кожен агент перебував у своїй цільової позиції, прагнучи мінімізувати глобальну функцію витрат. Функція вартості - це сума (за всіма агентам) кількості часових кроків, необхідних для досягнення цільового місцеположення. Отже, обидві дії - переміщення і очікування - коштують 1.0, за винятком випадку, коли дія очікування застосовується до цільовому місцезнаходження агента і нічого не коштує. Якщо агент чекає m разів у своєму цільовому місцезнаходження, а потім переміщається, вартість цього переміщення становить m + 1 [1, 2].

1.1.1 Аналіз типів конфліктів у класичному MAPF

Основною метою вирішувачів MAPF є пошук рішення, тобто одноагентного плану для кожного агента, який може бути виконаний без зіткнень. Щоб досягти цього, вирішувачі MAPF використовують поняття конфліктів під час планування, де рішення MAPF називається дійсним, якщо між будь-якими двома одноагентними планами не існує конфлікту. Визначення того, що являє собою конфлікт, залежить від середовища, і, відповідно, література про класичний MAPF включає кілька різних визначень того, що становить конфлікт між планами. Нехай π_i та π_j – пара одноагентних планів.

– Конфлікт вершин. Конфлікт вершин між π_i та π_j виникає тоді, коли згідно з цими планами агенти планують зайняти одну і ту ж вершину на одному і тому ж кроці. Формально існує конфлікт вершин між π_i та π_j, якщо існує часовий крок x такий, що $\pi_i[x] = \pi_j[x]$.

– Конфлікт грані. Конфлікт грані між π_i та π_j виникає, якщо згідно з цими планами агенти намагаються пройти одну і ту ж межу за один і той самий крок в одному напрямку. Формально існує конфлікт грані між π_i та π_j, якщо існує часовий крок x такий, що $\pi_i [x] = \pi_j [x]$ та $\pi_i [x + 1] = \pi_j [x + 1]$.

– Конфлікт переслідування. Такий конфлікт між π_i та π_j виникає, якщо один агент планується зайняти вершину, яка була зайнята іншим агентом на попередньому кроці часу. Формально існує конфлікт переслідування між π_i та π_j, якщо існує часовий крок x такий, що $\pi_i [x + 1] = \pi_j [x]$.

– Циклічний конфлікт. Циклічний конфлікт між набором одноагентних планів π_i, $\pi_i + 1$,. . . π_j відбувається, якщо на тому ж кроці часу кожен агент рухається до вершини, яка раніше була зайнята іншим агентом, утворюючи шаблон “обертального циклу”. Формально конфлікт циклу між набором планів π_i, $\pi_i + 1$,. . . π_j відбувається, якщо існує часовий крок x, в якому $\pi_i (x + 1) = \pi_i + 1 (x)$ та $\pi_i + 1 (x + 1) = \pi_i + 2 (x)$. . . і $\pi_j - 1 (x + 1) = \pi_j (x)$ та $\pi_j (x + 1) = \pi_i (x)$.

– Конфлікт обміну місцями. Конфлікт обміну між π_i та π_j виникає, якщо агенти планують обмінятися місцями за один часовий крок. Формально, існує конфлікт обміну місць між π_i та π_j, якщо існує часовий крок x такий, що

$\pi_i [x + 1] = \pi_j [x]$ та $\pi_j [x + 1] = \pi_i [x]$. Цей конфлікт у сучасній літературі MAPF іноді також називають конфліктом грані.

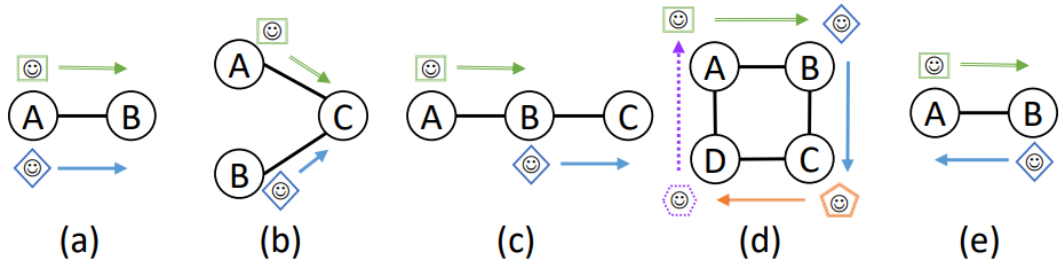

Рисунок 1.1 – Типи конфліктів у класичному MAPF: конфлікт грані (a), вершини (b), переслідування (c), циклічний конфлікт (d), конфлікт обміну (e)

Типи конфліктів у класичному MAPF наведені на рисунку 1.1.

Наведений набір визначень конфліктів, безумовно, не є повним набором усіх можливих конфліктів. Беручи до уваги офіційні визначення цих конфліктів, очевидно, що між ними існують відносини:

– заборона вершинних конфліктів означає, що також заборонені конфлікти країв;

– заборона конфліктів слідування передбачає заборону конфліктів циклів і конфліктів обміну також;

– заборона конфліктів циклів означає, що конфлікти заміни також заборонені.

Навпаки, дозвіл на існування конфліктів країв означає, що також допускаються конфлікти вершин тощо.

Щоб правильно визначити класичну проблему MAPF, потрібно вказати, які типи конфліктів допускаються при вирішенні. Найменшим обмеженням є заборона лише крайових конфліктів. Однак, усі попередні роботи над класичним MAPF також забороняють вершинні конфлікти. Деякі роботи над MAPF з передачею корисного навантаження дозволяють конфлікти обміну місцями. Більшість робіт над алгоритмами MAPF на основі пошуку забороняють обмін конфліктами, але дозволяють конфлікти слідування.

1.1.2 Поведінка агентів у класичному MAPF

У вирішенні класичної проблеми MAPF агенти можуть досягти своїх цілей на різних етапах часу. Тому, потрібно визначити, як поводиться агент після досягнення цілі та до того, як останній агент досяг цілі. Існує два загальних припущення щодо того, як агенти поводяться у кінцевих позиціях.

– залишається у цілі. Згідно з цим припущенням, агент чекає у кінцевій позиції, поки всі агенти не досягнуть своїх цілей. Цей агент очікування спричинить конфлікт вершин з будь-яким планом, який проходить через ціль після того, як він її досяг;

– зникає у цілі. Згідно з цим припущенням, коли агент досягає цілі, він негайно зникає. Це означає, що план цього агента не матиме жодних конфліктів після того періоду часу, за який відповідний агент досяг своєї мети.

Більшість робіт над класичним MAPF передбачають поведінку збереження у цільовому положенні.

1.1.3 Цільові функції у класичному MAPF

Можна з упевненістю сказати, що в більшості реальних додатків MAPF деякі рішення MAPF кращі за інші. Щоб зрозуміти це, робота в класичному MAPF розглядає цільову функцію, яка використовується для оцінки рішень MAPF. Дві найпоширеніші функції, що використовуються для оцінки рішення в класичному MAPF, - це час розрахунку і сума витрат.

Час розрахунку – це кількість кроків часу, необхідних всім агентам для досягнення цілі. Для MAPF-рішення $\pi = \{\pi_1, \ldots \pi_k\}$, час π визначається як $\max_{1 \le i \le k} | \pi_i |$.

Сума витрат – це сума часових кроків, необхідних кожному агенту для досягнення своєї мети. Сума витрат π визначається як $\sum_{1 \le i \le k} | \pi_i |$.

Якщо поведінка агента у цілі – залишатися на місці, а цільова функція - це сума витрат, тоді потрібно вказати, як перебування на цілі впливає на суму витрат. Наприклад, можна визначити, що якщо агент чекає у кінцевій позиції, це не збільшує суми витрат. Типовим припущенням у більшості робіт є те, що агент, що перебуває у

своїй цілі, враховується за дію очікування, якщо він не планує знову рухатися зі свого місця. Наприклад, припустимо, що агент і досягає цілі на кроці часу t, залишає ціль на кроці часу t_1, повертається до цілі на кроці часу t_2, а потім залишається у цілі, доки всі агенти не досягнуть своєї мети. Тоді цей одноагентний план внесе t_2 до суми витрат на відповідне рішення.

Проте, це не єдино можливі цільові функції для класичного MAPF. Можна визначити інші цільові функції, такі як загальна кількість дій очікування, необхідних для досягнення цілі (деякі називають це сумою палива), і загальний час, витрачений агентом до моменту досягнення кінцевої позиції.

1.1.4 MAPF на зважених графах

Усі вищезазначені класичні варіанти MAPF використовують наступні припущення:

– час розбивається на часові кроки;

– кожна дія займає рівно один часовий крок;

– на кожному часовому кроці кожен агент займає рівно одну вершину.

Припущення, що кожна дія - переміщення або очікування - займає рівно один часовий крок, та неявно передбачає дещо спрощену модель руху для агентів. У більш складних моделях руху різні дії можуть мати різну тривалість. Це означає, що базовий граф, що представляє можливі місця розташування агентів, які можуть зайняти агенти, тепер є зваженим графом, де вага кожного ребра відображає тривалість, необхідну агенту для проходження цього ребра [3].

Типи зважених графів, які використовуються у дослідженнях MAPS:

– MAPF у 2^k-сусідніх сітках. Такі карти є обмеженою формою зважених графів, в яких кожна вершина представляє клітинку у двовимірній сітці. Дії переміщення агента у клітці - це всі його 2^k сусідні клітки, де k - параметр. Витрати базуються на евклідовій відстані, тому, коли $k > 2$, це вводить дії з різними витратами. Наприклад, у 8-сусідній сітці діагональний хід коштує $\sqrt{2}$, тоді як рух в одному з основних напрямків коштує 1 (див. рисунок 1.2).

– MAPF в евклідовому просторі. Це узагальнення MAPF, в якому кожен вузол у G представляє евклідову точку (x, y), а ребра - дозволені дії переміщення.

Такі параметри виникають, наприклад, коли основний граф є дорожньою картою, сформованою для безперервного евклідового середовища [4].

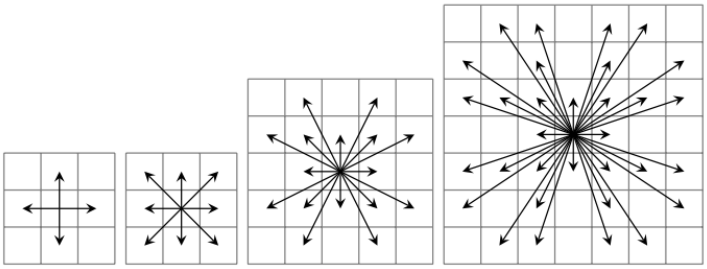

Рисунок 1.2 – Приклад руху агента у 2k-сусідніх сітках, де k = 2, 3, 4, 5 відповідно

1.1.5 Правила доцільності в MAPF

Визначення дійсного рішення, що використовується в класичному MAPF - відсутність конфліктів - це лише один із типів вимог до рішення. Для позначення вимоги щодо рішення MAPF використовується термін правила доцільності. Інші існуючі правила:

– правила надійності. Це правила, призначені для того, щоб рішення MAPF враховувало випадкові затримки у виконанні. K-надійний план MAPF створює достатній буфер для того, щоб агенти затримувались до k часових кроків, не приводячи до конфлікту. Коли вірогідність майбутніх затримок відома, правила надійності можуть вимагати, щоб ймовірність конфлікту агента під час виконання була нижчою за задану межу [5] або поєднувалася з політиками виконання, щоб гарантувати безконфліктне виконання;

– правила формування. Це обмеження щодо дозволених дій переміщення агента, які залежать від розташування інших агентів, але не пов'язані зі зіткненнями. Наприклад, обмеження, призначені для агентів для підтримання зазначеного формування, або для підтримки комунікаційного зв'язку з набором сусідніх агентів.

1.1.6 Алгоритми планування руху

У класичному MAPF агенти, як вважається, займають рівно одну вершину, в певному сенсі не мають обсягу, не мають форми і рухаються з постійною

швидкістю. Навпаки, алгоритми планування руху безпосередньо враховують ці властивості. У них агент знаходиться на кожному кроці часу не лише у вершині, а й у конфігурації, де конфігурація визначає місце розташування агента, орієнтацію, швидкість тощо, а ребро між конфігураціями представляє кінематичний рух.

Деякі дослідження MAPF розглядали агентів з певною геометричною формою та об'ємом [6]. Той факт, що агенти мають обсяг, породжує питання про те, як вони розташовані в базовому графу G і як вони рухаються в ньому. Зокрема, якщо агент знаходиться в одній вершині, це може заборонити іншим агентам займати найближчі вершини. Подібним чином, якщо агент рухається вздовж ребра, це може заборонити іншим агентам рухатися уздовж пересічних ребер або залишатися на вершинах, які знаходяться занадто близько до ребра. Це також може запровадити нові типи конфліктів, таких як конфлікти між вершинами, між ребрами та ребрами та і вершинами.

Деякі дослідження MAPF розглядали кінематичні обмеження щодо дії агента. Тобто дії переміщення, які може виконувати агент, залежать не тільки від його поточного місцезнаходження, але і від таких параметрів стану, як швидкість та орієнтація. Побічним продуктом таких обмежень є те, що базовий граф стає спрямованим, оскільки можуть бути ребра, які можуть бути прохідними лише в одному напрямку через кінематичні обмеження агента. Як приклад, MAPF-POST - це алгоритм MAPF, який враховує ці кінематичні обмеження шляхом подальшої обробки рішення, створеного алгоритмом MAPF. Існує також підхід, заснований на скороченні, який передбачає дії обертання як половину шляху у кінематичних обмеженнях.

1.1.7 Завдання і агенти

У класичному MAPF кожен агент має одне завдання – досягнути своєї мети. У роботах по MAPF було розроблено кілька розширень, в яких агентам може бути призначено більше однієї цілі [7].

– анонімний MAPF. У цьому варіанті MAPF метою є переміщення агентів до набору цільових вершин, але не має значення, який агент досягає якої цілі;

– кольоровий MAPF. Цей варіант MAPF є узагальненням анонімного MAPF, в якому агенти згруповані в команди, і кожна команда має набір цілей. Мета полягає в тому, щоб перемістити агентів у кожній команді до їх цілей. Інший формулювання цього варіанту - це проблема, при якій кожен агент може бути призначений цілям лише з набору цілей, призначених для його команди. Можна узагальнити кольорові MAPF, призначаючи ціль та агента декільком командам;

– онлайн MAPF. В цьому варіанті послідовність задач MAPF вирішується на одному графі. Проблеми онлайнового MAPF можна класифікувати наступним чином:

1. Складська модель. Це модель, коли фіксований набір агентів вирішує проблему MAPF, але після того, як агент знаходить ціль, може бути поставлено завдання перейти до іншої цілі. Вона була створена для вирішення проблем автономних складів;

2. Модель перетину. Це модель, де можуть з’являтися нові агенти, і кожен агент має одне завдання - досягти своєї мети.

Звичайно, можливі також гібридні моделі, в яких агент може отримати нове завдання, коли досягне своєї мети, а з часом можуть з’явитися нові агенти.

1.1.8 Підходи до тестування в класичному MAPF

Проблема MAPF визначається графом та набором вихідних та цільових вершин. Типи карт, які зазвичай використовуються для перевірки продуктивності, включають:

– карти Dragon Age Origins (DAO). Це сітки взяті з гри Dragon Age Origin і є загальнодоступними. Ці сітки відносно великі та відкриті, деякі мають розмір 1000 × 1000 і більше;

– відкриті сітки N × N. Це сітки, де загальними значеннями N є 8, 16 і 32. Вони дозволяють проводити експерименти, в яких велике відношення агентів до простору або їх щільність, при цьому менше вершин без агента в них [8];

– сітки N × N із випадковими перешкодами. Це N × N сітки, де набір кліток сітки обраний випадковим чином і вважається непрохідним (перешкоди);

– складські сітки. Це сітки, що мають форму подібних до автоматизованих складів з довгими коридорами.

Вибравши тип карти, потрібно встановити вихідну та цільову вершини агентів. Існує кілька методів встановлення джерел та цілей агентів, серед яких:

– випадкові. Встановлення вихідної та цільової вершин шляхом випадкового вибору вершин та переконавшись, що між ними на графі є шлях;

– кластерні. Встановлення джерела та цілі першого агента шляхом випадкового вибору вершин на графі. Встановлення джерел і цілей усіх інших агентів на відстані не більше r від джерела та цілі першого агента, відповідно, де r – параметр;

– визначені. Встановлення джерела для кожного агента шляхом випадкового вибору із зазначеного набору можливих вершин джерела та встановлення цілі кожного агента аналогічним чином шляхом вибору з набору визначених цільових вершин.

Випадкове визначення є найпоширенішим [9]. Кластерний метод присвоєння використовується, щоб зробити проблеми MAPF більш складними. Метод присвоєння визначених вершин використовувався для імітувати автоматизованих складів та автономних транспортних засобів на перехрестях.

Загальнодоступний і найбільш популярний тест складається з 24 карт, взятих з карт реальних міст, відеоігор Dragon Age Origins та Dragon Age 2, відкритих сіток із випадковими перешкодами та без них, лабіринтних сіток , та сітки, схожі на кімнату. На рисунку 1.3 наведено приклад карти кожного з цих типів.

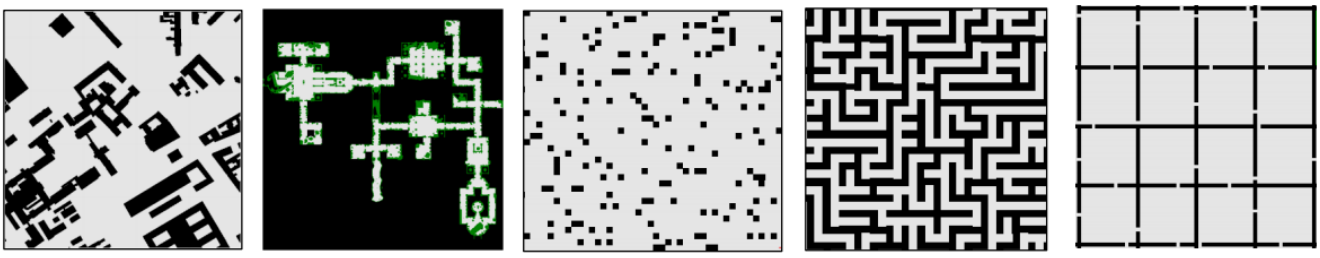

Рисунок 1.3 – Приклад карт для тестування мультиагентного пошуку шляхів

На рисунку 1.4 зображено приклад результатів тесту на таких картах. Кожна карта має 25 сценаріїв. У кожному сценарії є список вихідних і цільових вершин, які були встановлені за допомогою варіанту випадкового методу. Усі точки в найбільшій доступній області кожної карти були випадковим чином з'єднані в пари, а потім перші 1000 проблем були введені в сценарій .

Для вибраного алгоритму MAPF, типу карти та сценарію вирішується якомога більше агентів у кожному сценарії, додаючи їх у послідовному порядку. Тобто, тест починається із створення проблеми MAPF двох агентів, використовуючи перші дві пари джерело-ціль, пов’язані з обраним сценарієм, і запускається обраний алгоритм MAPF для вирішення цієї проблеми. Якщо алгоритм вибору успішно вирішує цю проблему MAPF за невеликий час, створюється нова проблема MAPF з трьома агентами, використовуючи перші три пари джерело-ціль цього сценарію, і вирішується за допомогою обраного алгоритму. Це продовжується ітеративно, доки алгоритм не зможе вирішити проблему MAPF за визначений час. Потім оцінений алгоритм може повідомити для кожного сценарію максимальну кількість агентів, яку він зміг розв'язати за визначений час.

Встановлено 30 секунд як обмеження часу роботи. Різні рядки відповідають різним картам. Стовпець “Size” показує кількість рядків і стовпців на кожній карті. У стовпці "Problems" вказано кількість проблем, доступних для кожної карти. Це число агрегується за 25 сценаріями, де кількість проблем, доступних у сценарії, це кількість пар джерело-ціль, визначених для нього. У стовпці "Solved" повідомляється про кількість проблем, вирішених за вказаний час. У таблиці є два додаткові стовпці - “Min” та “Max”. У цих стовпцях повідомляється про максимальну кількість агентів, розв’язаних у сценарії, а саме найменше число (“Min”) і воно було найбільше (“Max”). Наприклад, значення "Min" для карти brc202d дорівнює 2, а "Max" дорівнює 22. Це означає, що для цієї карти існує сценарій, в якому алгоритм зміг вирішити щонайбільше 2 агенти до досягнення тайм-ауту, і є інший сценарій для цієї карти, в якому програма змогла вирішити для 22 агентів. Тобто складність вирішення різних сценаріїв на одній і тій самій карті може відрізнятися.

Type	Map	Size	Problems	Solved	Min	Max
City	Berlin_1_256	256 X 256	25000	892	4	52
	Boston_0_256	256 X 256	25000	718	13	47
	Paris_1_256	256 X 256	25000	805	3	47
DAO	brc202d	481 X 530	25000	252	2	22
	den312d	81 X 65	25000	577	7	36
	den520d	257 X 256	25000	661	5	47
	lak303d	194 X 194	25000	377	8	27
	orz900d	656 X 1491	25000	162	2	12
	ost003d	194 X 194	25000	535	7	37
Dragon Age 2	ht_chantry	141 X 162	25000	513	11	35
	ht_mansion_n	270 X 133	25000	795	17	42
	w_woundedcoast	578 X 642	25000	336	6	22
	lt_gallowstemplar_n	180 X 251	25000	493	10	32
Open	empty-8-8	8 X 8	800	528	18	25
	empty-16-16	16 X 16	3200	840	15	52
	empty-32-32	32 X 32	12800	1190	12	81
	empty-48-48	48 X 48	25000	1349	18	118
Open+ obstacles	random-32-32-10	32 X 32	11525	1027	15	68
	random-32-32-20	32 X 32	10225	862	15	46
	random-64-64-10	64 X 64	25000	1450	24	87
	random-64-64-20	64 X 64	25000	1078	10	64
Maze	maze-32-32-2	32 X 32	8325	327	8	17
	maze-32-32-4	32 X 32	9875	317	4	23
	maze-128-128-10	128 X 128	25000	272	5	20
	maze-128-128-2	128 X 128	25000	178	4	15
Room	room-32-32-4	32 X 32	12800	469	10	26
	room-64-64-16	64 X 64	25000	629	12	45
	room-64-64-8	64 X 64	25000	360	8	24

Рисунок 1.4 – Приклад результатів тесту

1.2 Аналіз вирішувачів мультиагентного пошуку шляхів

Послідовність дій очікування та переміщення одного агента, що ведуть агента від si до gi, позначається як шлях, і використовується термін «рішення» для позначення набору k шляхів, по одному для кожного агента. Конфліктом між двома шляхами є кортеж ⟨ai, aj, v, t⟩ , де агент ai та агент aj планують зайняти вершину v в момент часу t. Вартість шляху визначається як кількість дій у ньому, а вартість рішення як сума витрат на складові шляхи. Рішення є дійсним, якщо воно безконфліктне.

Алгоритми MAPF для пошуку дійсних рішень можна класифікувати на три класи: оптимальні вирішувачі, неоптимальні вирішувачі та обмежені неоптимальні вирішувачі.

Класифікація вирішувачів MAPF зображена на рисунку 1.5.

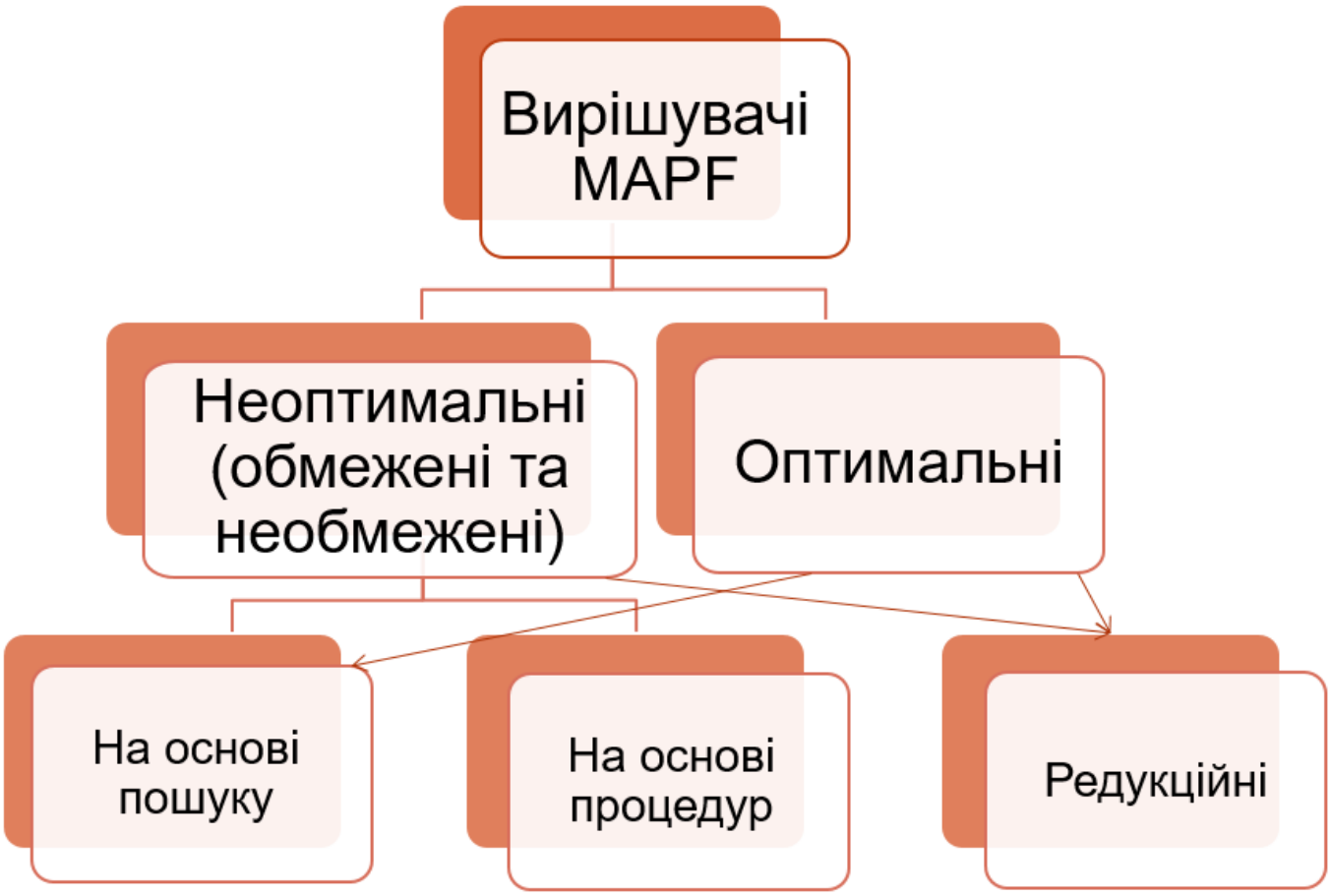

Рисунок 1.5 – Класифікація вирішувачів MAPF

1.2.1 Оптимальні вирішувачі

Існує ряд оптимальних вирішувачів MAPF, які базуються на алгоритмі A*. Простір станів включає всі перестановки розміщення k агентів у V місцях. Стендлі представив систему виявлення незалежності (ID). ID розбиває проблему на менші незалежні підзадачі, якщо це можливо, і вирішує кожну проблему окремо. Стенлі також представив операторне розкладання (OD), яке враховує переміщення окремого

агента за раз. OD зменшує коефіцієнт розгалуження за рахунок збільшення глибини розчину в дереві пошуку. Інший відповідний варіант A * - Розширене часткове розширення A * (EPEA *). EPEA * використовує апріорні знання домену для сортування всіх наступників даного стану відповідно до їх значень [10].

M * - це алгоритм, заснований на A*, який динамічно змінює фактор розгалуження на основі конфліктів. Загалом, вузли розширюються лише за допомогою одного дочірнього елемента, де кожен агент робить свій оптимальний рух до мети. Коли між q агентами виникають конфлікти, фактор розгалуження збільшується, щоб охопити всі внутрішні можливості. Покращена версія, яка називається Рекурсивний M* (RM*), ділить q конфліктуючих агентів на підгрупи, кожна з яких має незалежні конфлікти. Тоді RM* викликається рекурсивно для кожної з цих груп. Варіант ODRM * поєднує OD Стендлі та RM*.

Нещодавно було запропоновано два оптимальні алгоритми пошуку MAPF, які не базуються на A*. Перший – це алгоритм пошуку дерева зростання витрат (ICTS) оптимально вирішує MAPF, перетворюючи його в набір швидко вирішуваних проблем прийняття рішень. Другий - пошук на основі конфлікту (CBS) та його мета-агент CBS (MA-CBS) [11].

1.2.2 Неоптимальні вирішувачі

Багато існуючих вирішувачів MAPF спрямовані на швидкий пошук рішення, дозволяючи повертати неоптимальне рішення. Більшість неоптимальних вирішувачів не надають жодних гарантій щодо якості повернутого шляху.

Ранні роботи представили теоретичний алгоритм поліноміального часу для пошуку дійсних рішень MAPF. Цей алгоритм є повним для будь-якої проблеми MAPF. Однак рішення, яке він повертає, може бути далеко не оптимальним, і повернутий план переміщує агентів з часом, тобто по одному на кожному шляху часу.

Важлива група неоптимальних алгоритмів MAPF базується на пошуку. Яскравим прикладом є кооперативний алгоритм A* (CA*) та його варіанти HCA* та WHCA*. CA* та його варіанти послідовно планують шлях для кожного окремого агента. Кожен запланований шлях записується в глобальну таблицю резервування, і

наступним агентам не дозволяється планувати шлях, що суперечить шляхам у таблиці резервування. Важливо, що CA* та його варіанти не є повними. Стендлі та Корф запропонували повний неоптимальний вирішувач MAPF, який називається MSG1, що є спокійною версією згаданого вище алгоритму Стенді на базі A* [12].

Інша група неоптимальних алгоритмів базується на правилах. Вони включають специфічні правила руху для різних сценаріїв. Вирішувачі, що базуються на правилах, зазвичай мають деякі обмеження, і вони направлені на швидке дійсне рішення, але якість поверненого рішення може бути далеко не оптимальною. Алгоритм Push and swap (PS) використовує два оператори, які “штовхають” агента в порожнє місце та “міняють місцями” два агенти. PS підходить для графів принаймні з двома порожніми місцями. Паралельний Push and swap (PPS) - це вдосконалений варіант PS, який планує паралельні маршрути для різних агентів замість переміщення лише одного агента одночасно. Також існують гібриди алгоритмів на основі пошуку та правил.

1.2.3 Обмежені неоптимальні вирішувані [13]

Обмежений неоптимальний алгоритм пошуку приймає параметр w і повертає рішення, яке гарантовано буде меншим або рівним w * C, де C - вартість оптимального рішення. Обмежені неоптимальні алгоритми пошуку забезпечують посередництво між оптимальними та необмеженими неоптимальними алгоритмами. Встановлення різних значень w дозволяє користувачеві контролювати компроміс між часом роботи та якістю рішення.

Існує декілька обмежених неоптимальних алгоритмів пошуку, таких як WA*. Ці алгоритми зазвичай розширюють оптимальні алгоритми пошуку, наприклад, A * або IDA*, розглядаючи розширену версію допустимої евристики. Таким чином, ці алгоритми можна застосувати до вирішувача MAPF на базі A*, такого як EPEA*, A* з OD та M*.

Однак незрозуміло, як модифікувати вирішувачі MAPF, які не базуються на A*, до неоптимальних версій. Зокрема, метод розширення евристики не застосовується у пошуку на основі конфліктів, оскільки він не використовує евристику.

1.2.4 Оптимальний вирішувач на основі зменшення [13]

Цей клас вирішувачів, що використовується в останніх роботах, зводить MAPF до інших проблем, які добре вивчені в інформатиці. Яскраві приклади включають зменшення до логічної задоволеності (SAT), цілочисельне лінійне програмування (ILP) та програмування набору відповідей (ASP). Ці методи повертають оптимальне рішення і, як правило, призначені для функції витрат часу. Вони менш ефективні або навіть не застосовуються для функції суми витрат. Крім того, ці алгоритми зазвичай є високоефективними лише в невеликих екземплярах проблем. У великих екземплярах проблем процес зменшення з екземпляра MAPF на необхідну проблему має дуже великі, але при цьому поліноміальні накладні витрати, що робить ці підходи неефективними.

1.3 MAPF у динамічному оточенні

Під час виконання обчислення MAPF в динамічному середовищі (наприклад, на складі, який не є повністю автономним) можуть відбутися зміни: існуючі перешкоди можуть бути усунені із середовища або переміщені в інше місце в середовищі, існуючі агенти можуть залишити оточення, або нові агенти можуть бути включені в команду з новими завданнями. Тоді мета полягає у пошуку нового рішення для нової команди агентів у модифікованому середовищі. Цю проблему називають задачею багатоагентного пошуку шляхів у динамічному оточенні (D-MAPF). D-MAPF успадковує нерозв'язність MAPF, з урахуванням обмежень довжини плану [14].

Одним із можливих рішень для D-MAPF є перепланування - створення нового екземпляру MAPF, визначеного поточними розташуваннями та цільовими розташуваннями як існуючих, так і нових агентів, та оновленого середовища, та обчислення цього екземпляра. Хоча перепланування знаходить рішення, якщо воно існує, воно не використовує повторно попередньо обчисленні плани існуючих агентів і може не бути обчислювально ефективним. На рисунку 1.6 зображена схема вирішення конфліктної ситуації у D-MAPF.

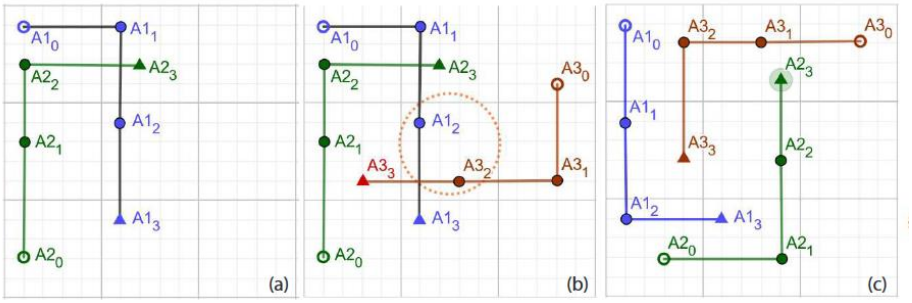

Рисунок 1.6 – Приклад додавання нового агента, визначення конфліктної ситуації та перебудова шляхів

1.4 Постановка задачі

Виходячи з аналізу предметної області можна виділити, що область вивчення MAPF володіє великою кількістю різноманітних алгоритмів та методів розв’язання задач пошуку шляху. Проте частина цих алгоритмів не гарантує оптимального вирішення. Базові алгоритми мультиагентного пошуку напрямлені на вирішення статичної задачі пошуку шляхів, тобто розв’язання задачі передбачає лише одноразовий розрахунок маршрутів агентів і не передбачає ситуацій, при яких можуть виникати конфлікти під час руху агентів.

Базовим рішенням для динамічного MAPF є перепланування - створення нового екземпляру MAPF, визначеного поточними розташуваннями та цільовими розташуваннями як існуючих, так і нових агентів, та оновленого середовища, та обчислення цього екземпляра. Хоча перепланування знаходить рішення, якщо воно існує, воно не використовує попередньо обчисленні плани існуючих агентів повторно і може бути обчислювально неефективним.

Тому потрібно провести функціональне моделювання компонентів мультиагентної системи пошуку шляхів, побудувати діаграми класів, послідовностей та кооперацій, станів системи та активностей, розробити метод, що оптимізує базовий варіант динамічного мультиагентного пошуку шляху. Розроблений метод повинен розв’язувати задачі часткового перепланування швидше, ніж аналог, та повертати оптимальні шляхи, як результат.

Також потрібно провести порівняльне тестування розробленого методу оптимізації з аналогами на картах для тестування MAPF різного розміру з різною

кількістю агентів (наприклад, 5, 8, 12) та рухомих об’єктів, що будуть викликати конфлікти. Розроблений компонент системи повинен давати змогу динамічно добавляти нових агентів та перешкоди.

2 РОЗРОБКА ВИМОГ ДО КОМПОНЕНТІВ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ПОШУКУ ШЛЯХІВ

2.1 Розробка системних вимог до компонентів інформаційної системи.

Компоненти, що розробляються, призначені для пошуку оптимальних шляхів на тривимірній ігровій карті.

Виходячи з цілей створення компонентів системи, розглянемо системні вимоги до неї:

– компоненти повинні бути реалізовані у вигляді віконного додатку;

– керування має здійснюватися шляхом натискання на клавіші клавіатури та миші;

– система повинна видавати повідомлення у разі помилок.

2.2 Визначення функціональних вимог до компонентів інформаційної системи

Найбільш зручною методологією функціонального моделювання, призначеної для формалізації і опису бізнес-процесів, є IDEF0. В IDEF0 система представляється як сукупність взаємодіючих робіт або функцій [15].

Перша діаграма - головна - контекстна діаграма. Вона вміщує тільки один блок, який відображає головну функцію системи. Головна функція системи – пошук оптимальних шляхів у динамічному оточенні.

Контекстна діаграма зображена на рисунку 2.1.

Після опису системи в цілому, проводиться розбиття її на великі фрагменти. Цей процес називається функціональної декомпозицією, а діаграми, які описують кожен фрагмент і взаємодія фрагментів, називають діаграмами декомпозиції [16].

Після декомпозиції контекстної діаграми проводиться декомпозиція кожного великого фрагмента системи на більш дрібні і так далі, до досягнення потрібного рівня детальності опису. Так досягається відповідність моделі реальним процесам на будь-якому і кожному рівні моделі. Синтаксис опису системи в цілому і кожного її фрагмента однаковий у всій моделі.

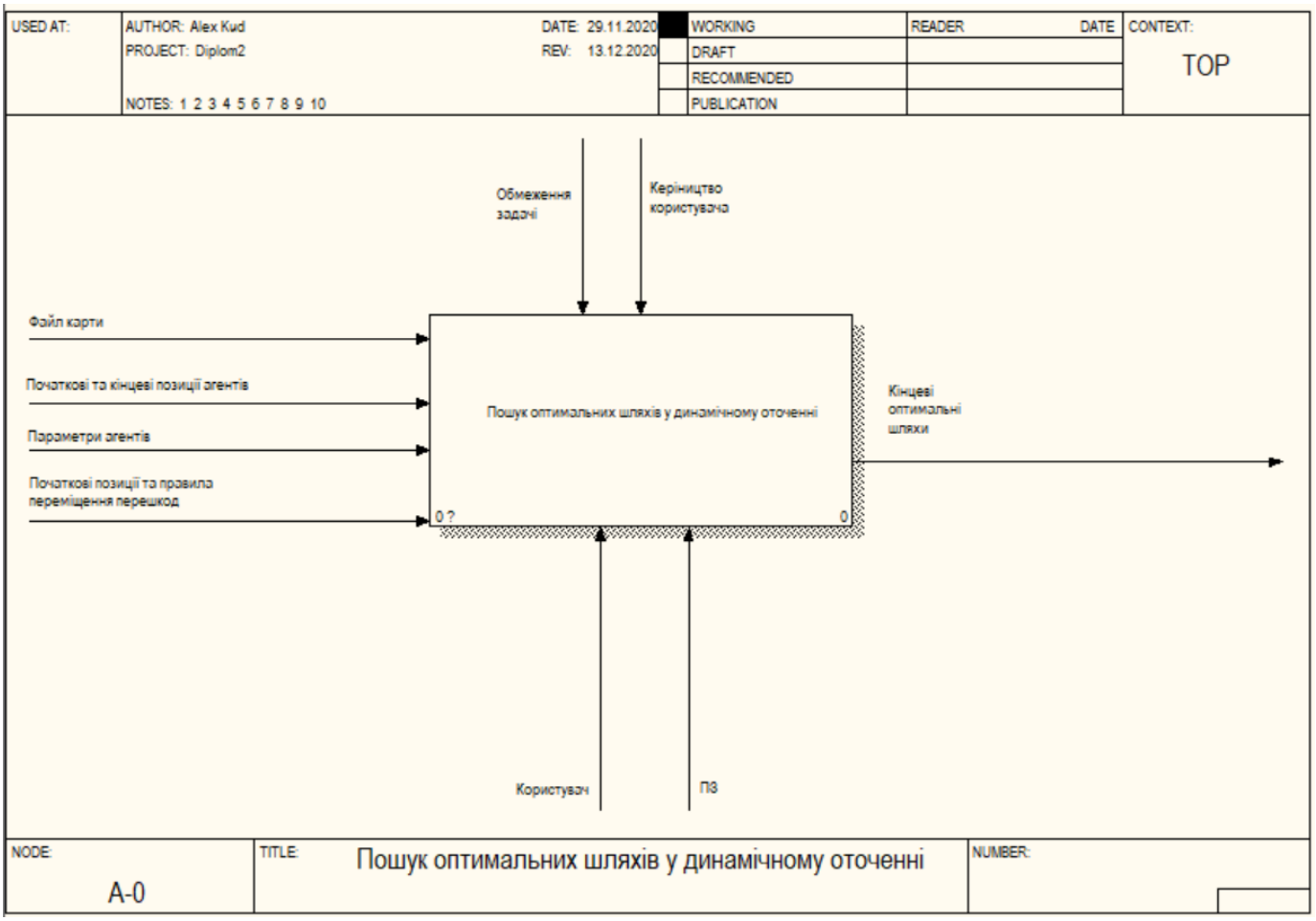

Рисунок 2.1 – Контекстна діаграма

Після декомпозиції контекстної діаграми «Пошук оптимальних шляхів у динамічному оточенні» виникло п’ять функцій:

– побудова карти – зчитування додатком текстури прохідності, побудова на її основі ландшафту та графу прохідності, по якому будуть переміщуватися агенти та рухомі перешкоди;

– встановлення перешкод – встановлення перешкод користувачем на згенерованому ландшафті;

– визначення агентів – встановлення агентів на згенерованому ландшафті;

– побудова вихідних шляхів – пошук шляху для усіх встановлених на карті агентів перед запуском симуляції;

– виконання симуляції переміщення – виконується переміщення усіх рухомих перешкод та агентів за їх шляхами.

Декомпозиція контекстної діграми зображена на рисунку 2.2.

Функція «Побудова карти» складається з трьох підфункцій – завантаження текстури прохідності, генерації ландшафту на основі текстури і генерації графу прохідності на основі отриманої ландшафтної сітки.

Декомпозиція цієї функції зображена на рисунку 2.3.

Функція «Встановлення та редагування перешкод» складається з чотирьох підфункцій – додавання нової перешкоди на карту, вибір перешкоди з вже встановлених, задання перешкоді її шлях і правила руху.

Декомпозиція даної функції зображена на рисунку 2.4.

Функція «Встановлення та редагування агента» складається з чотирьох підфункцій – додавання нової перешкоди на карту, вибір перешкоди з вже встановлених, задання перешкоді її шлях і правила руху.

Декомпозиція даної функції зображена на рисунку 2.5.

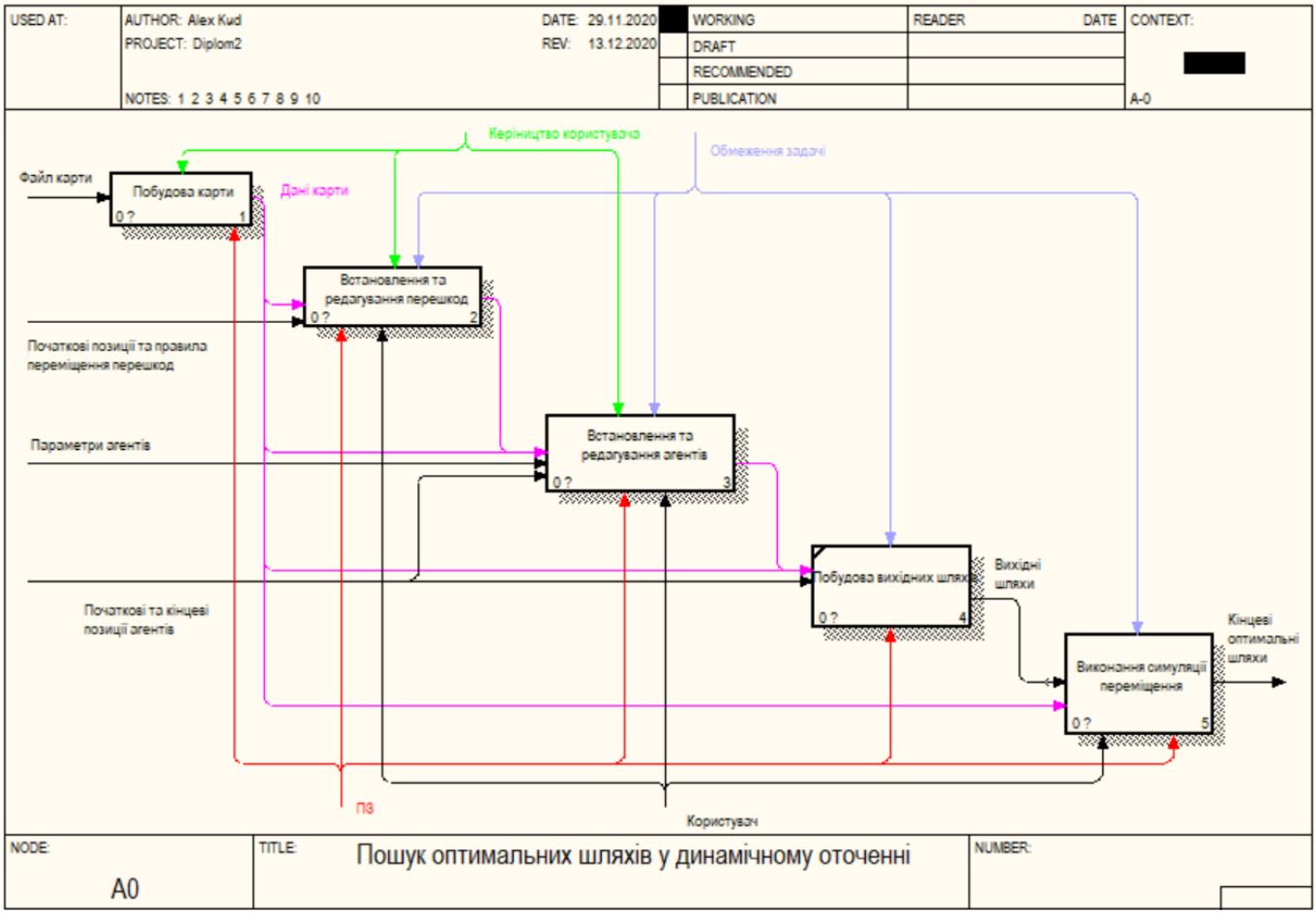

Рисунок 2.2 – Діаграма декомпозиції (A0)

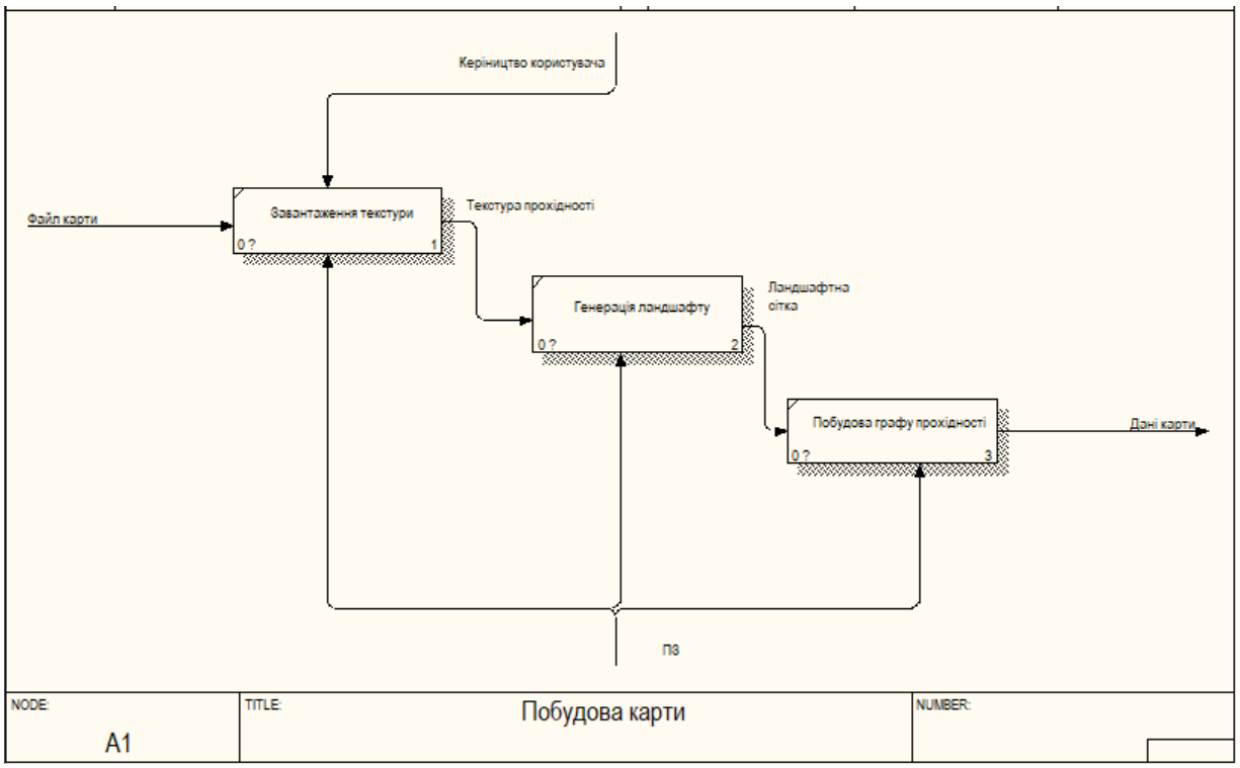

Рисунок 2.3 – Діаграма декомпозиції функції «Побудова карти» (А1)

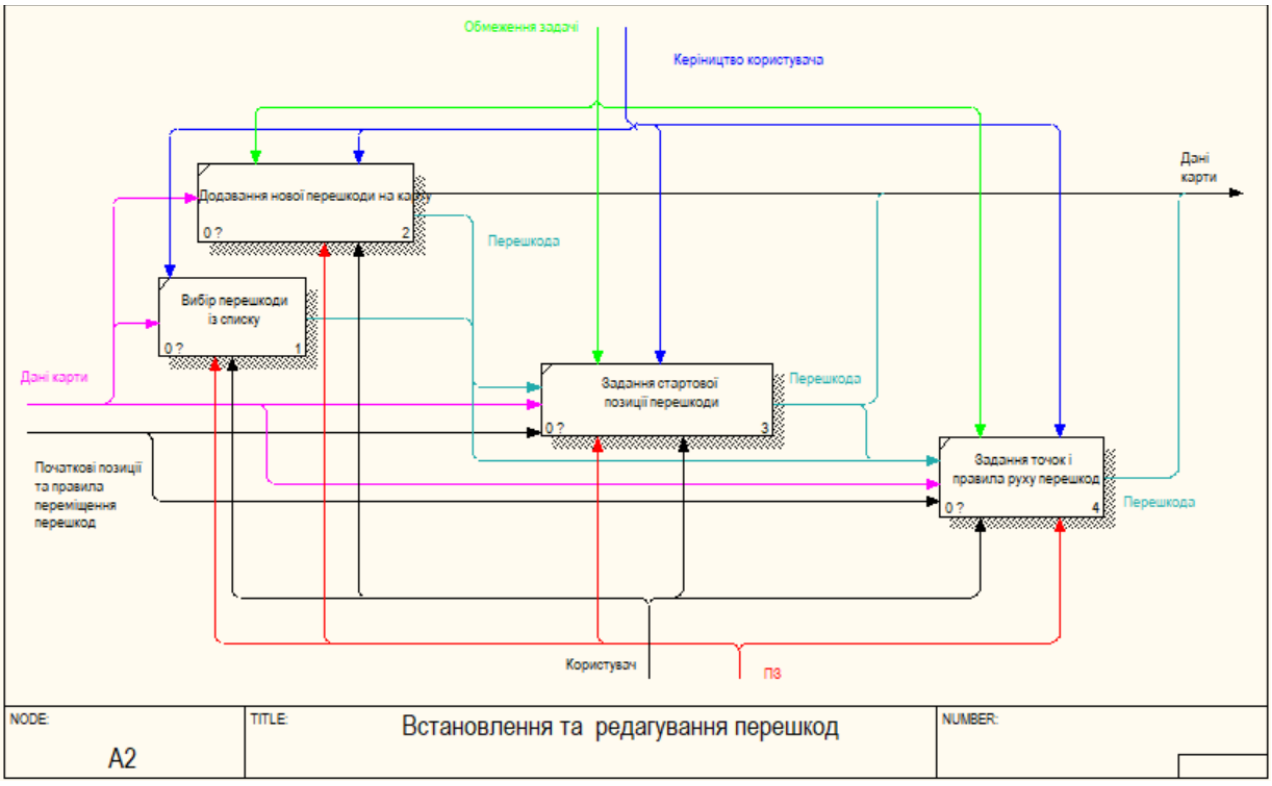

Рисунок 2.4 – Діаграма декомпозиції функції «Встановлення та редагування перешкод» (А2)

Функція «Встановлення та редагування агентів» складається з п’яти підфункцій:

– вибір агента із існуючих – користувач вибирає агента із існуючих на сцені;

– додавання нового агента – користувач додає новий об’єкт агента на сцену;

– задання стартової позиції агенту – користувач обирає стартовий вузол на графі, за якого буде починатися рух об’єкта.

Декомпозиція цієї функції зображена на рисунку 2.5.

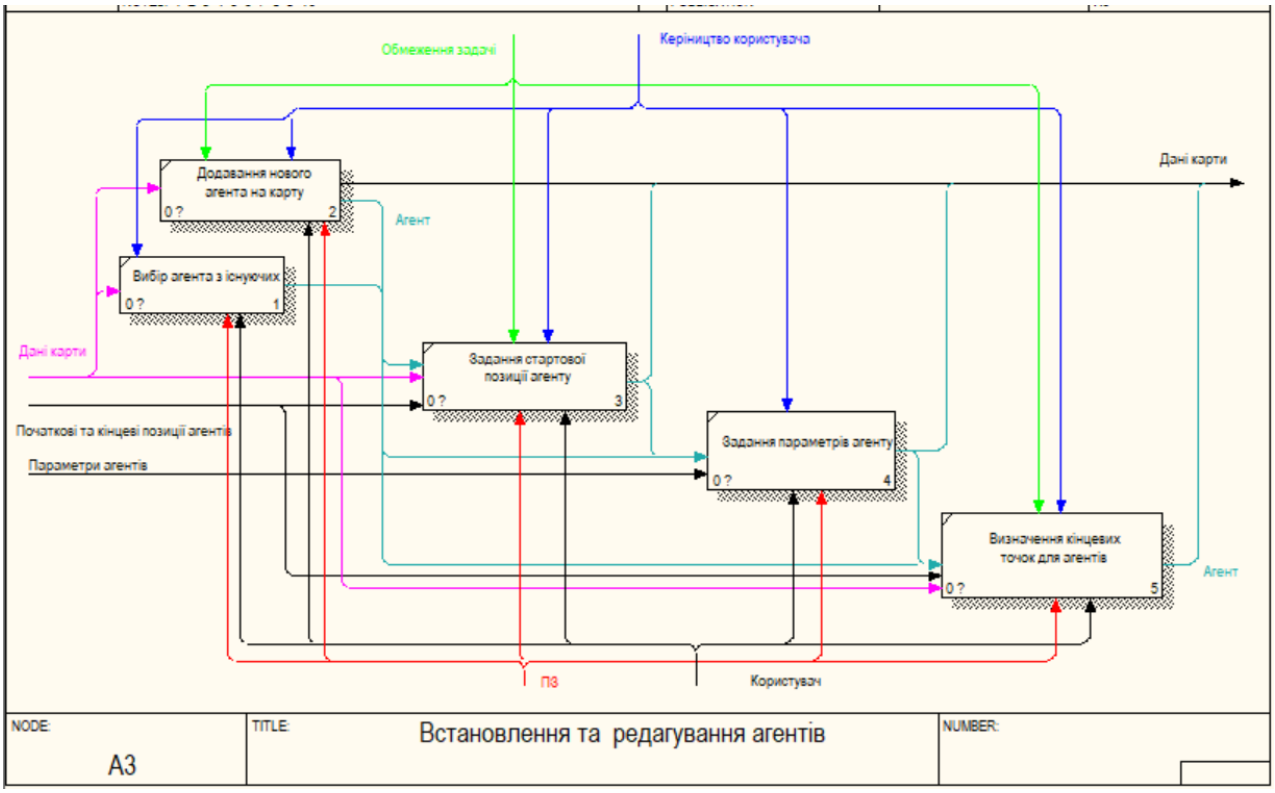

Рисунок 2.5 – Діаграма декомпозиції функції «Встановлення та редагування агентів» (A3)

Функція «Виконання симуляції переміщення» складається з двох підфункцій – переміщення агентів та вирішення конфліктних ситуацій.

Декомпозиція даної функції зображена на рисунку 2.6.

Функція «Переміщення агентів» у свою чергу також складається з двох підфункцій – переміщення агентів по їх шляхам та виявлення конфліктів, зв’язанних з переміщенням. Декомпозиція даної функції зображена на рисунку 2.7.

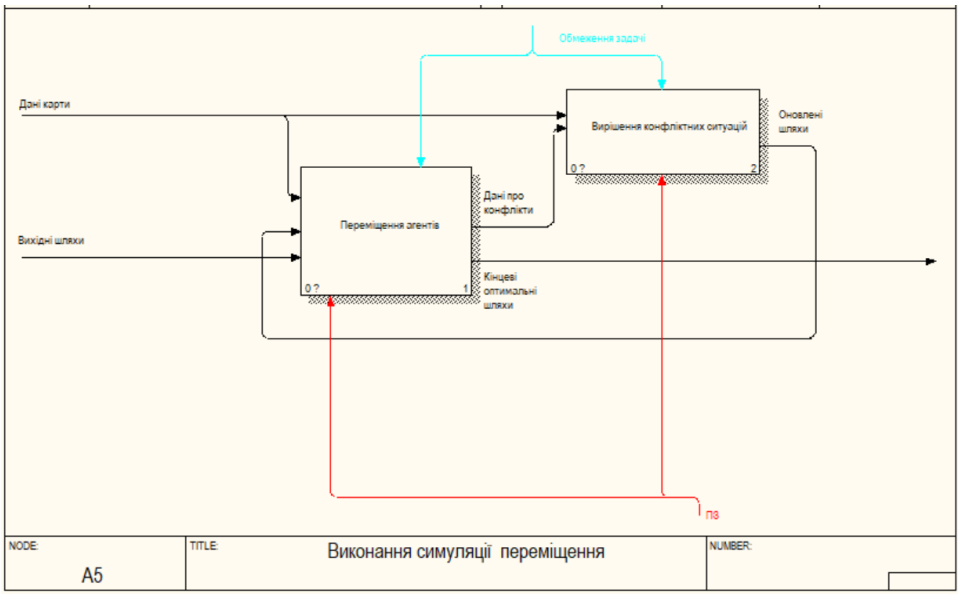

Рисунок 2.6 – Діаграма декомпозиції функції «Виконання симуляції переміщення» (A5)

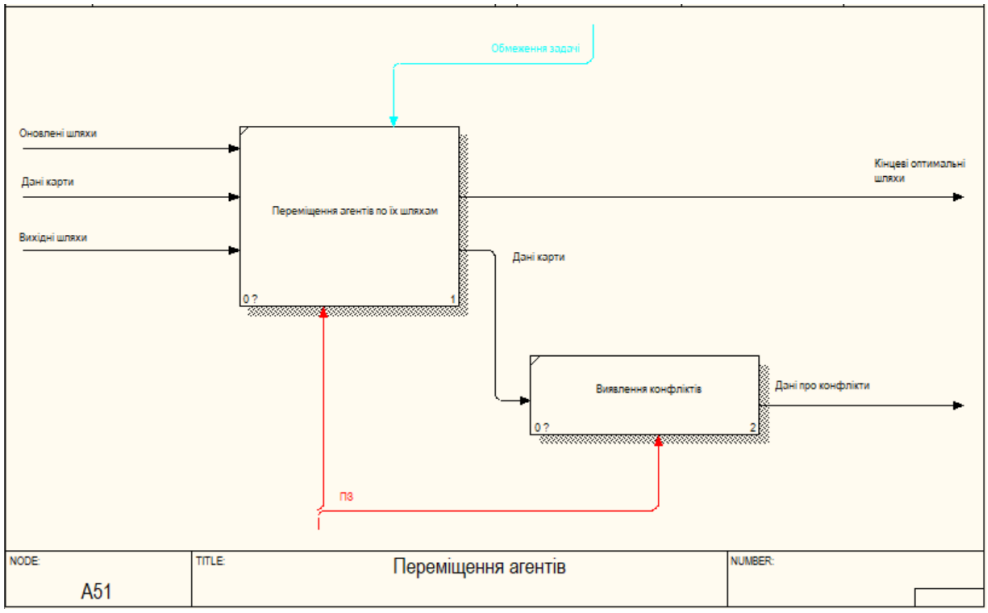

Рисунок 2.7 – Діаграма декомпозиції функції «Виконання симуляції переміщення» (A51)

Функція «Вирішення конфліктних ситуацій», як і попередня, складається з двох функцій – перебудова шляхів конфліктуючих агентів та розрахунок шляхів нових агентів, якщо ті були добавлені під час симуляції руху.

Декомпозиція даної функції зображена на рисунку 2.8.

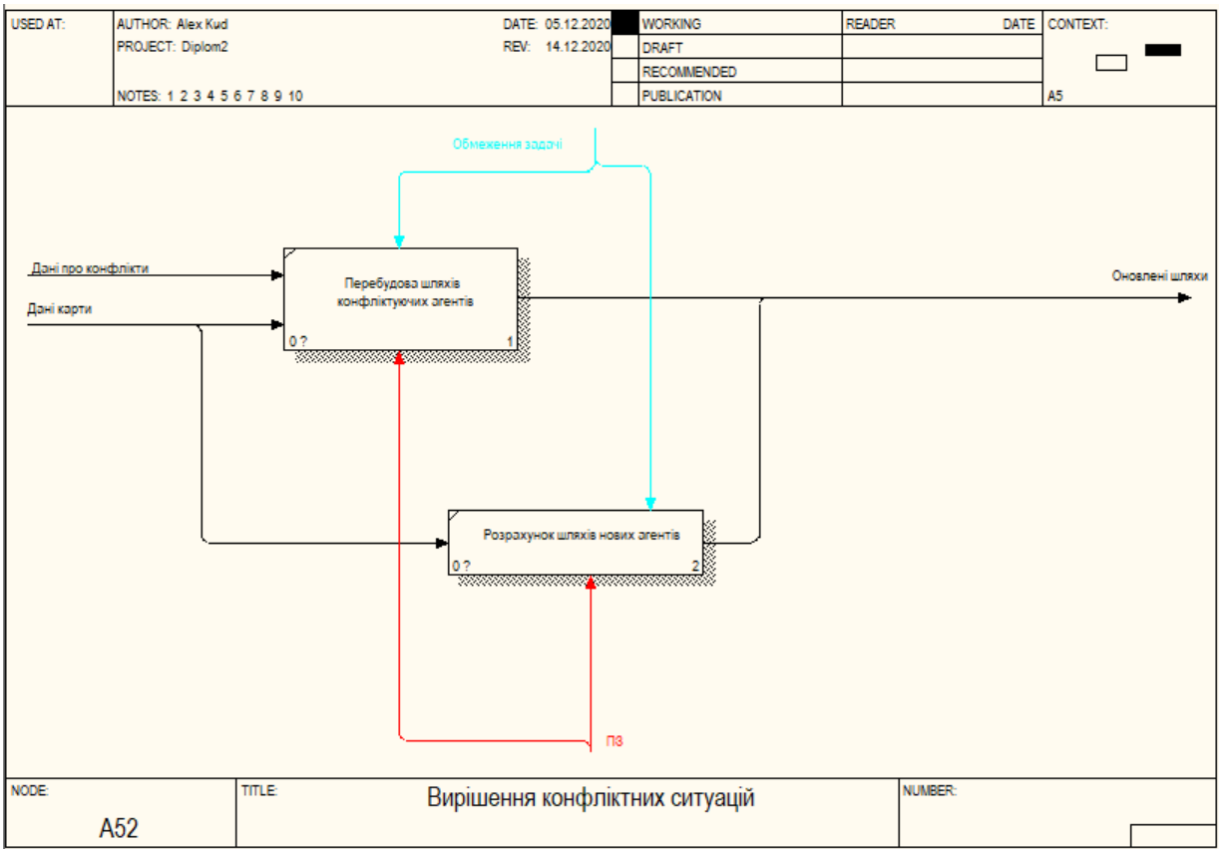

Рисунок 2.8 – Діаграма декомпозиції функції «Виконання симуляції переміщення» (A52)

Таким чином, функціональне моделювання дозволило визначити такі функціональні вимоги до компонентів системи, що розробляються:

– побудова карти:
– завантаження текстури;
– генерація ландшафту;
– побудова графу прохідності;
– встановлення та редагування перешкод:
– вибір перешкоди зі списку;

– додавання нової перешкоди на карту;
– задання стартової позиції перешкоди;
– задання точок і правила руху перешкод;
– встановлення та редагування агентів:
– вибір агента з існуючих;
– додавання нового агента на карту;
– задання стартової позиції агенту;
– задання параметрів агенту;
– визначення кінцевих точок для агентів;
– побудова вихідних шляхів;
– виконання симуляції переміщення;
– переміщення агентів по їх шляхам;
– виявлення конфліктів;
– перебудова шляхів конфліктуючих агентів;
– розрахунок шляхів нових агентів.

2.3 Розробка моделі потоків даних компонентів інформаційної системи

Наступним етапом, при визначенні вимог до інформаційної системи, являється створення діаграми потоків даних DFD. Діаграми потоків даних використовуються для опису документообігу та обробки інформації. DFD представляє систему, що моделюється, як мережу пов'язаних між собою робіт [17]. Їх можна використовувати як доповнення до моделі IDEF0 для більш наочного відображення поточних операцій документообігу в корпоративних системах обробки інформації. Головна мета DFD - показати, як кожна робота перетворює свої вхідні дані у вихідні, а також виявити відносини між цими роботами.

На рисунку 2.9 зображена контекстної діаграма, що показує потоки даних між користувачем та системою.

На рисунку 2.10 зображена декомпозиція контекстної діаграми. Сховище «Репозиторій» надає текстури, в яких знаходяться мапи для генерації ландшафту. У сховищі «Дані карти» знаходяться дані про ландшафт, список перешкод та граф.

Також присутнє сховище «Список агентів» що містить у собі список агентів, їх шляхи та параметри.

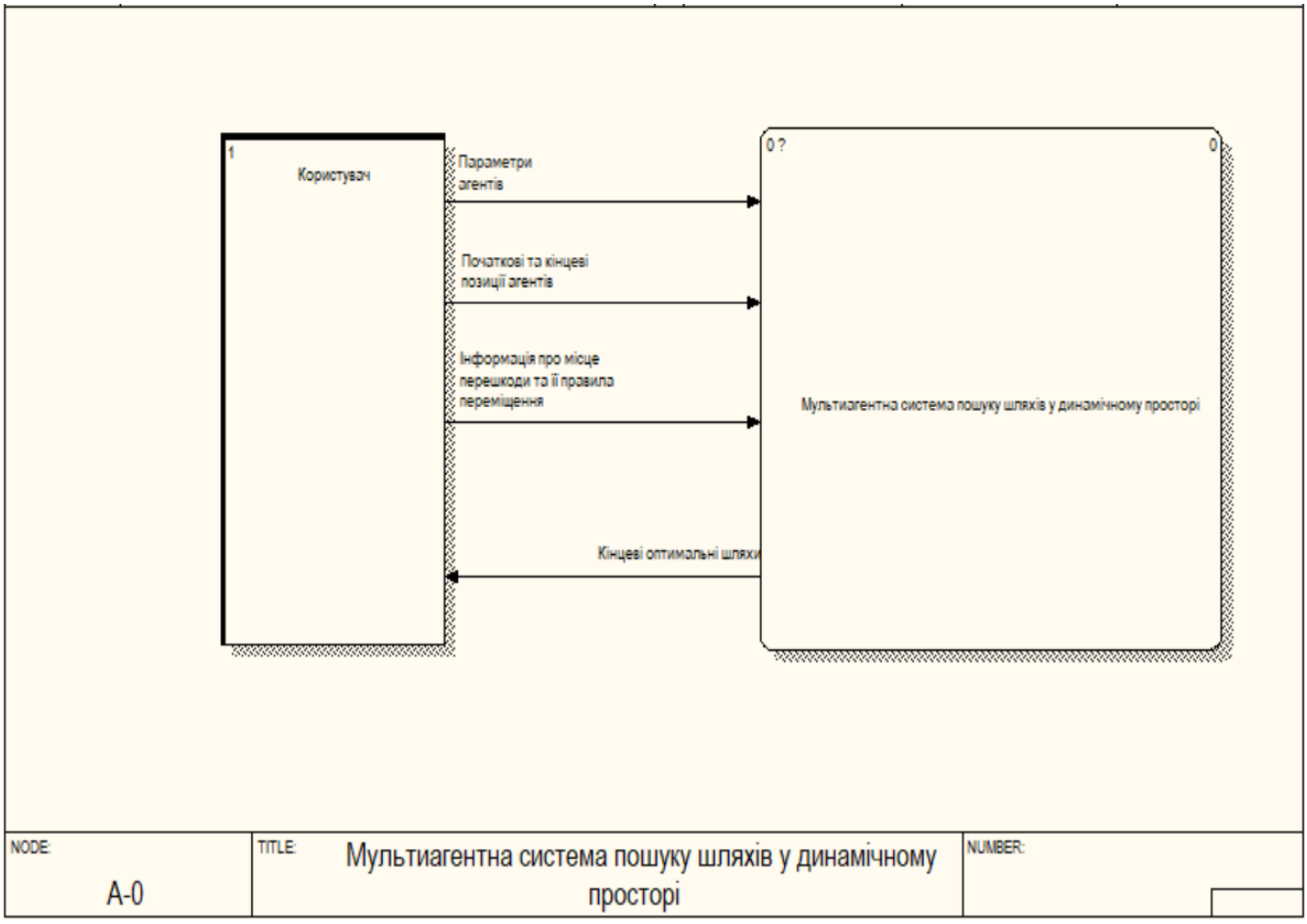

Рисунок 2.9 – Контекстна діаграма DFD

На рисунку 2.11 зображена декомпозиція функції «Встановити та редагувати перешкоди». В ній відбувається додавання нової перешкоди на карту, вибір перешкоди з вже встановлених, задання перешкоді її шлях і правила руху.

На рисунку 2.12 зображена декомпозиція функції «Встановити та редагувати агентів». В ній відбувається вибір агента із існуючих, додавання нового агента, задання стартової позиції обраному агенту, задання йому кінцевої точки та параметрів.

На рисунку 2.13 зображена декомпозиція функції «Виконати симуляцію переміщення агентів».

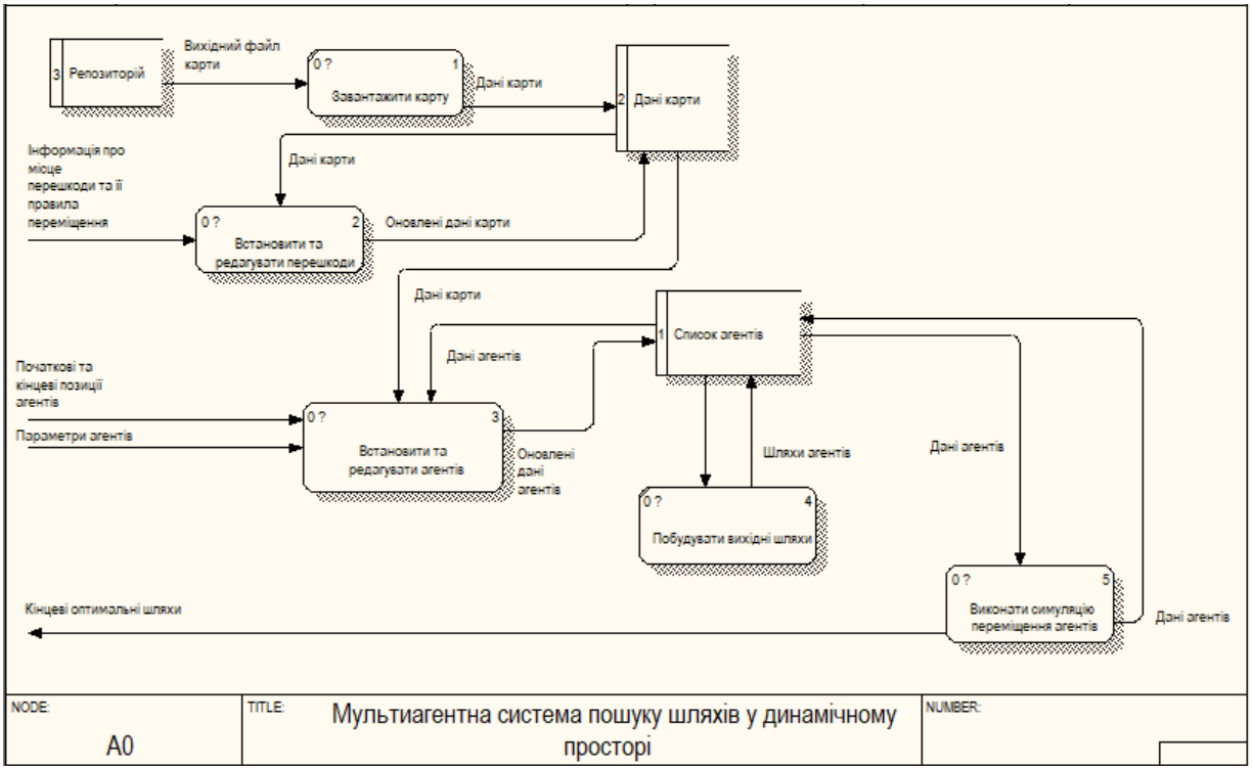

Рисунок 2.10 – Декомпозиція контекстної діаграми DFD

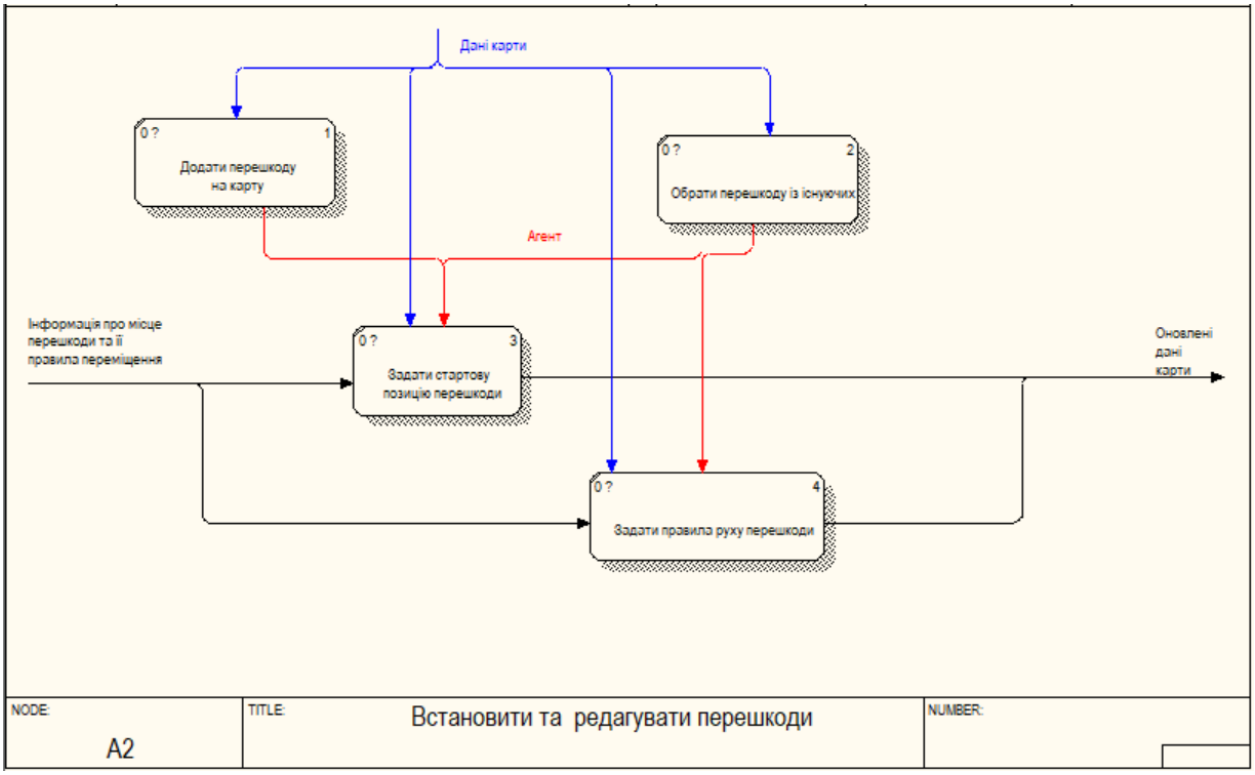

Рисунок 2.11 – Декомпозиція функції «Встановити та редагувати перешкоди»

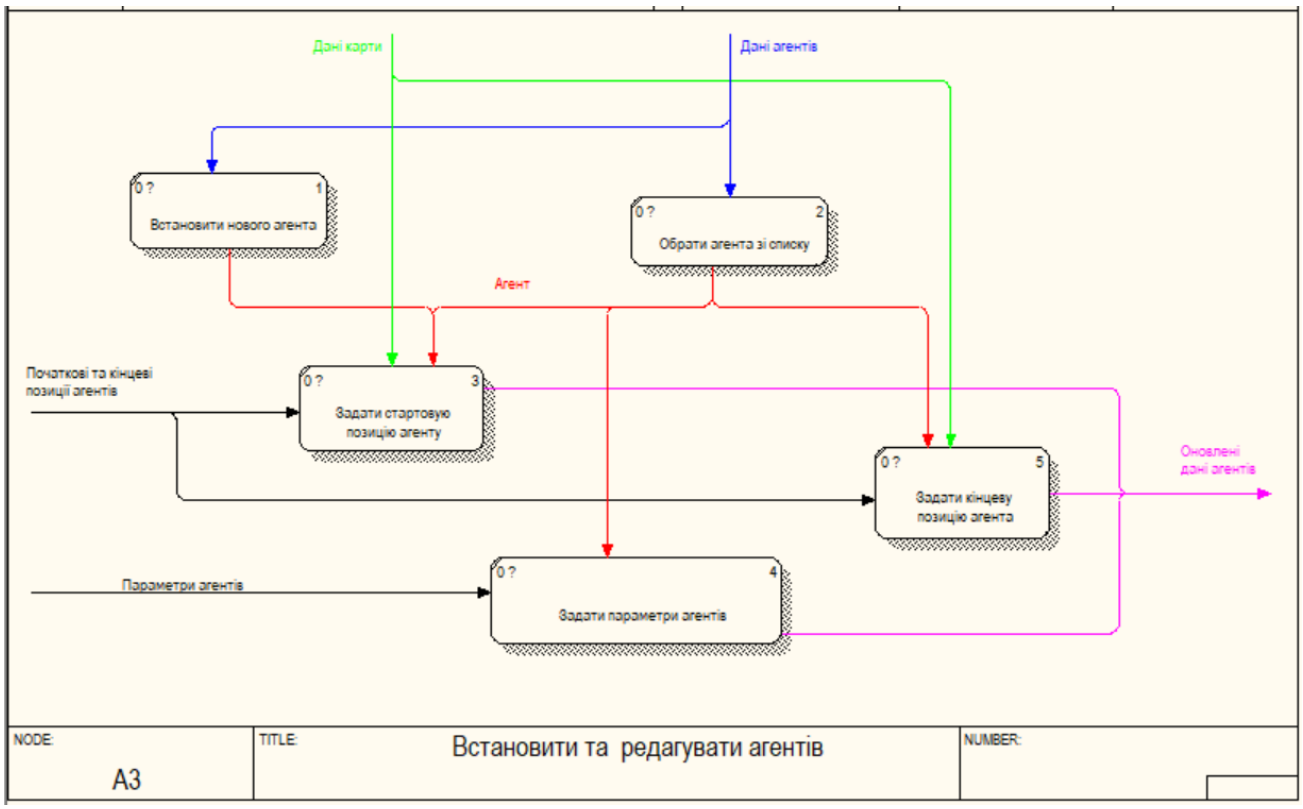

Рисунок 2.12 – Декомпозиція функції «Встановити та редагувати агентів»

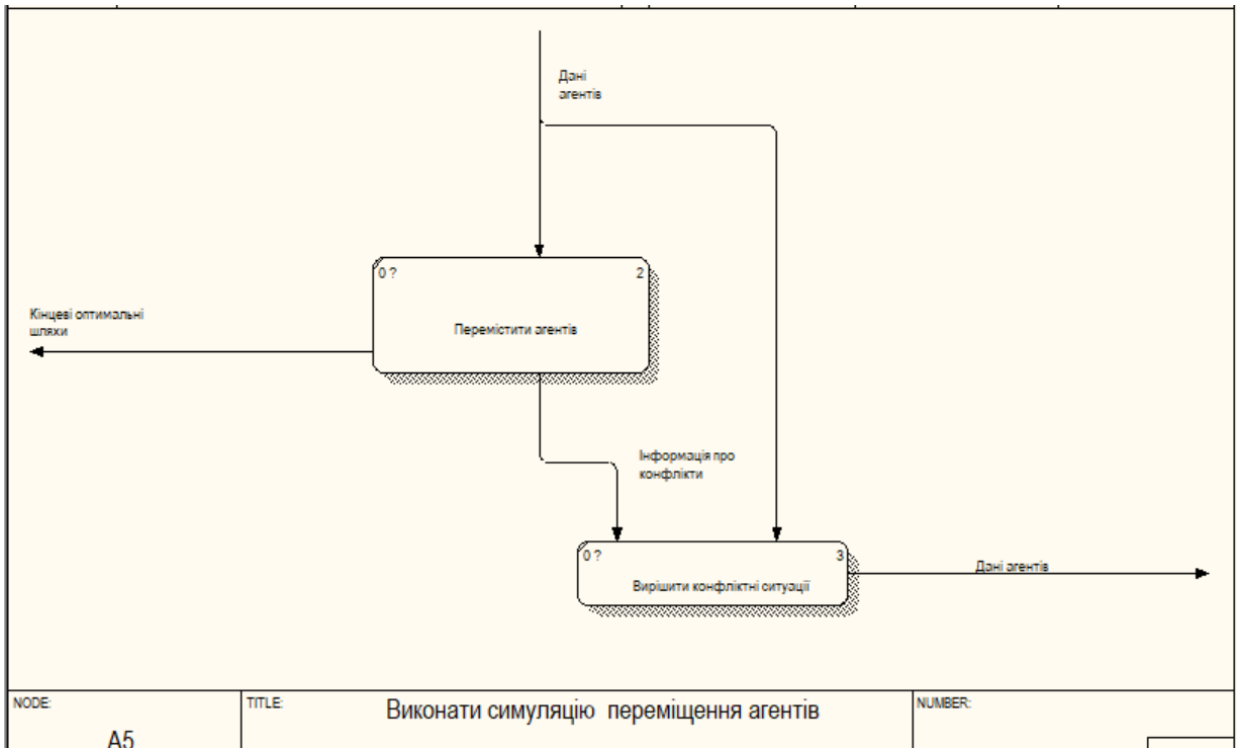

Рисунок 2.13 – Декомпозиція функції «Встановити та редагувати агентів»

2.4 Діаграма варіантів використання компонентів інформаційної системи

Діаграма варіантів використання (Use Case Diagram) є вихідним концептуальним поданням системи в процесі її проектування і розробки. Дана діаграма складається з акторів, варіантів використання і відносин між ними [18]. При побудові діаграми можуть використовуватися також загальні елементи нотації: примітки і механізми розширення.

Суть даної діаграми складається в наступному: проектована система представляється у вигляді безлічі акторів, що взаємодіють з системою за допомогою так званих варіантів використання. При цьому актором називається будь-який об'єкт, суб'єкт або система, що взаємодіє з моделюється системою ззовні. У свою чергу варіант використання - це специфікація сервісів (функцій), які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, що здійснюються системою при взаємодії з актором. При цьому в моделі ніяк не відбивається те, яким чином буде реалізовано цей набір дій.

На рисунку 2.14 зображено діаграму варіантів використання для компонентів системи пошуку оптимальних шляхів у динамічному оточенні.

Користувач – це людина, яка є користувачем системи пошуку шляху. Вона може як ініціювати створення нової сцени, з якою буде працювати, так і взаємодіяти з вже відкритою сценою.

Варіант «Відкрити сцену» - функція, що ініціює основні процеси програмного додатку. У цьому варіанті система, ініційована користувачем, завантажує ресурси, потрібні для роботи системи.

Варіант «Завантажити карту» - наступна функція системи, до якої та переходить після відкриття сцени. Цей варіант означає що програмний продукт завантажить текстуру карти та побудує на її основі ландшафт.

Варіант «Побудувати граф» - означає що система, згідно з побудованим ландшафтом, побудує граф прохідності, що буде використовуватися під час пошуку шляху.

Варіант «Встановити перешкоду» - прецедент, що передбачає встановлення користувачем рухомих перешкод на карті.

Варіант «Додати агента» - прецедент, що передбачає встановлення користувачем агента на карті.

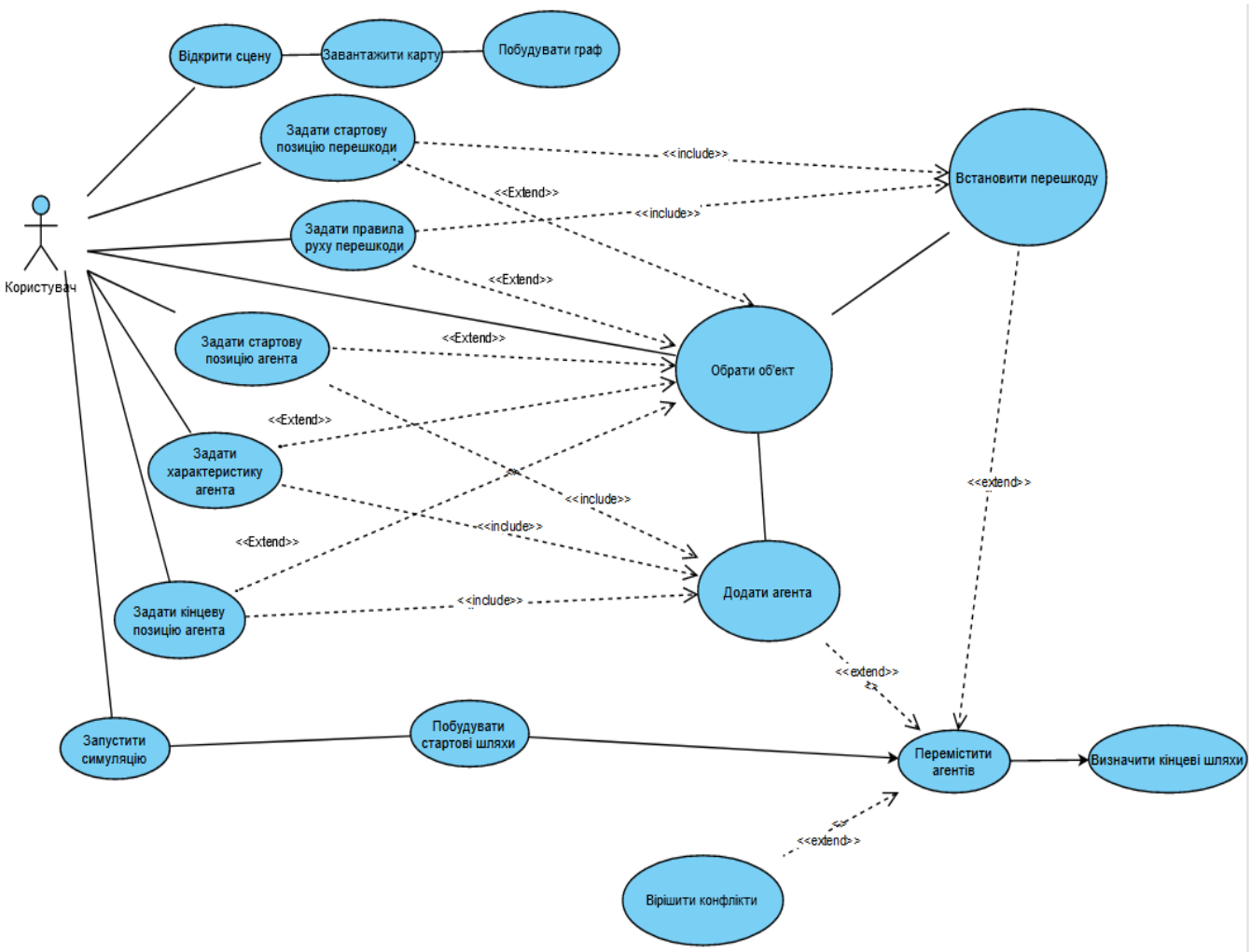

Рисунок 2.14 – Діаграма варіантів використання для компонентів системи пошуку оптимальних шляхів у динамічному оточенні

Варіант «Обрати об’єкт» - прецедент, що передбачає обрання користувачем рухомої перешкоди або агента з встановлених в сцені.

Варіант «Задати стартову позицію перешкоди» - прецедент, що передбачає обрання користувачем точки, з якої повинна вирушити рухома перешкода після запуску симуляції. Перед ним обов’язково повинен виконатися або прецедент «Обрати об’єкт», або прецедент «Встановити перешкоду».

Варіант «Задати правила руху перешкоди» - прецедент, що передбачає задання користувачем точок, за якими повинен рухатися об’єкт після запуску симуляції та правила переміщення – слідувати за точками у порядку встановлення чи рухатися у випадковому порядку. Перед ним обов’язково повинен виконатися або прецедент «Обрати об’єкт», або прецедент «Встановити перешкоду».

Варіант «Задати стартову позицію агента» - прецедент, що передбачає обрання користувачем точки графа, з якої повинен вирушити агент після запуску симуляції. Перед ним обов’язково повинен виконатися або прецедент «Обрати об’єкт», або прецедент «Додати агента».

Варіант «Задати характеристику агента» - прецедент, що передбачає введення користувачем розміру та швидкості пересування агента. Перед ним обов’язково повинен виконатися або прецедент «Обрати об’єкт», або прецедент «Додати агента».

Варіант «Задати кінцеву позицію агента» - прецедент, що передбачає обрання користувачем точки графа, у якій повинен опинитися агент після симуляції руху. Перед ним обов’язково повинен виконатися або прецедент «Обрати об’єкт», або прецедент «Встановити перешкоду».

Варіант «Запустити симуляцію» означає запуск симуляції руху об’єктів.

Варіант «Побудувати стартові шляхи» - прецедент, що відбувається одразу після запуска симуляції. На даному етапі система побудує шляхи для усіх агентів, встановлених перед запуском.

Наступним іде прецедент «Перемістити агентів». Він означає що під час активної симуляції існуючі на сцені агенти будуть рухатися за їх шляхами до відповідних кінцевих точок. Цей прецедент розширюють прецеденти «Додати агента», «Встановити перешкоду» так «Вирішити конфлікти». Під час переміщення агентів користувач може додати нових агентів або перешкоди на сцену, що може викликати утворення конфліктів шляху у існуючих агентів.

Варіант «Вирішити конфлікти» - прецедент, що може відбуватися під час симуляції руху об’єктів, якщо у агентів виникли конфлікти. Він передбачає перебудову шляхів конфліктуючих агентів.

Останнім варіантом являється прецедент «Визначити кінцеві шляхи». Він відбувається одразу після завершення переміщення усіх агентів на сцені та збирає для користувача усі фактичні пройдені агентами шляхи.

2.5 Діаграма класів

Діаграма класів визначає типи класів системи і різного роду статичні зв'язки, які існують між ними. На діаграмах класів зображуються також атрибути класів, операції класів та обмеження, які накладаються на зв'язку між класами. Вид і інтерпретація діаграми класів істотно залежить від точки зору (рівня абстракції): класи можуть представляти сутності предметної області (в процесі аналізу) або елементи програмної системи (в процесах проектування і реалізації). Основними елементами є класи і зв'язку між ними. Класи характеризуються за допомогою атрибутів і операцій [19].

На рисунку 2.15 зображена діаграма класів системи.

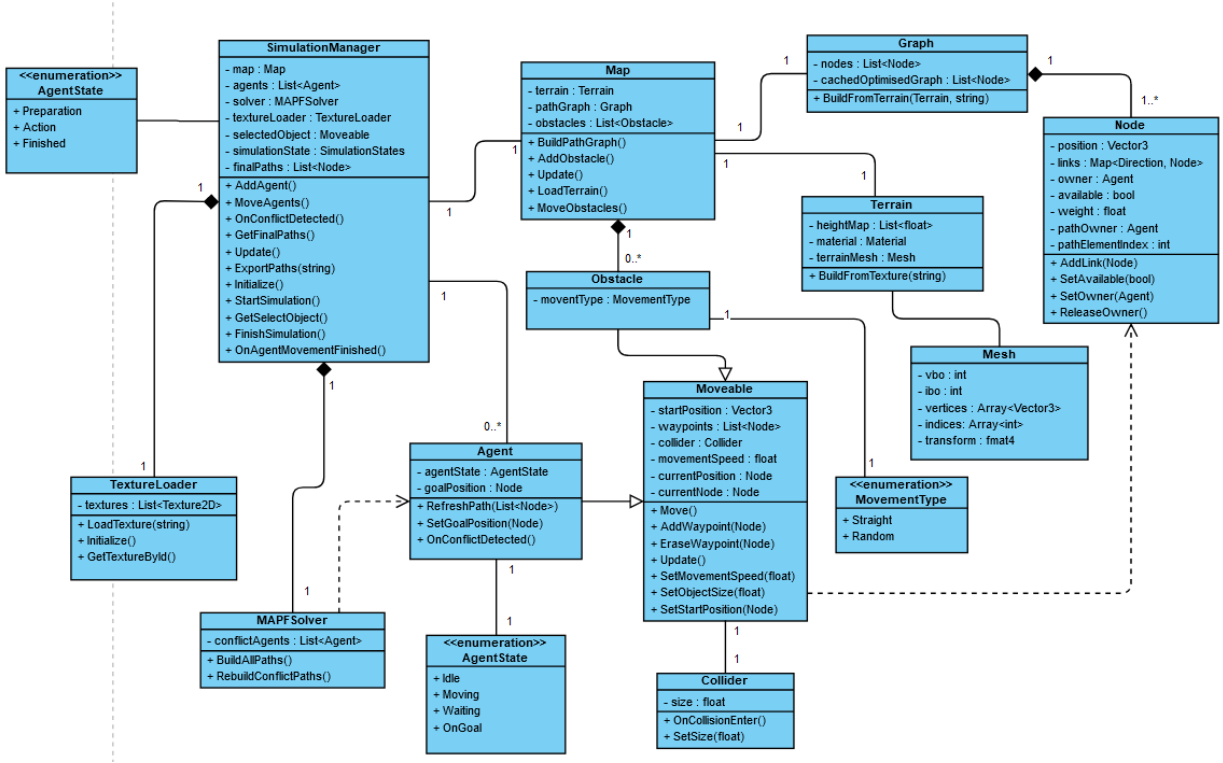

Рисунок 2.15 – Діаграма класів системи пошуку оптимальних шляхів у динамічному оточенні

Атрибути описують властивості об'єктів класу. Більшість об'єктів в класі отримують свою індивідуальність через відмінності в їх атрибутах і взаємозв'язку з іншими об'єктами. Однак, можливі об'єкти з ідентичними значеннями атрибутів і взаємозв'язків. Тобто індивідуальність об'єктів визначається самим фактом їх

існування, а не відмінностями в їх властивості. Ім'я атрибута повинно бути унікально в межах класу. За ім'ям атрибута може слідувати його тип і значення за замовчуванням.

2.6 Діаграми послідовностей

Діаграми послідовностей використовуються для уточнення діаграм прецедентів, більш детального опису логіки сценаріїв використання [20]. Вони зазвичай містять об'єкти, які взаємодіють в рамках сценарію, повідомлення, якими вони обмінюються, і які повертаються результати, пов'язані з повідомленнями. Втім, часто повертаються результати позначають лише в тому випадку, якщо це не очевидно з контексту.

На рисунку 2.16 зображена діаграма послідовностей для прецеденту «Відкрити сцену». На ній зображена послідовність дій, які проходять у системі після того як користувач запустив програмний додаток. Спочатку основний керуючий клас ініціює послідовність завантажень текстур у пам’ять програми з диску. Потім, на основі текстури створюється ландшафт, а нього генерується граф прохідності.

На рисунку 2.17 зображена діаграма послідовностей для прецедентів «Додати агента» та «Обрати кінцеву позицію». На ній зображено, як спочатку користувач встановлює нового агента на карту, задає йому стартову позицію та параметри, а після обирає кінцеву точку стартової позиції.

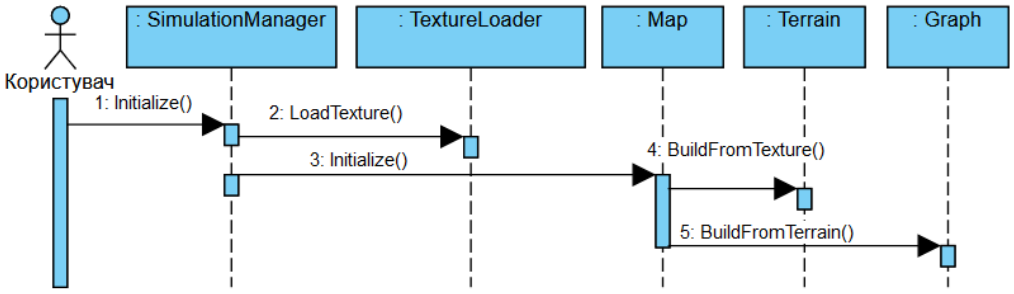

Рисунок 2.16 – Діаграма послідовностей для прецеденту «Відкриття сцени»

На рисунку 2.18 зображена діаграма послідовностей для прецедентів «Запустити симуляцію», «Перемістити агентів» та «Вирішити конфлікти». На ній

зображена послідовність дій, які запускають симуляцію, переміщують агентів та реагують на виникнення конфліктних ситуацій.

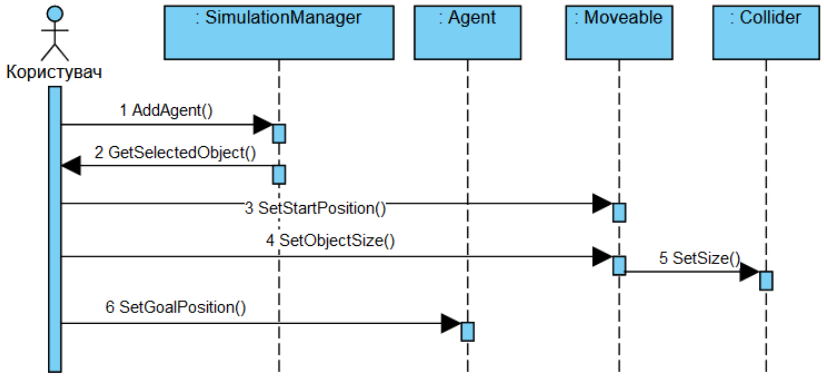

Рисунок 2.17 – Діаграма послідовностей для прецедентів «Додати агента» та «Обрати кінцеву позицію»

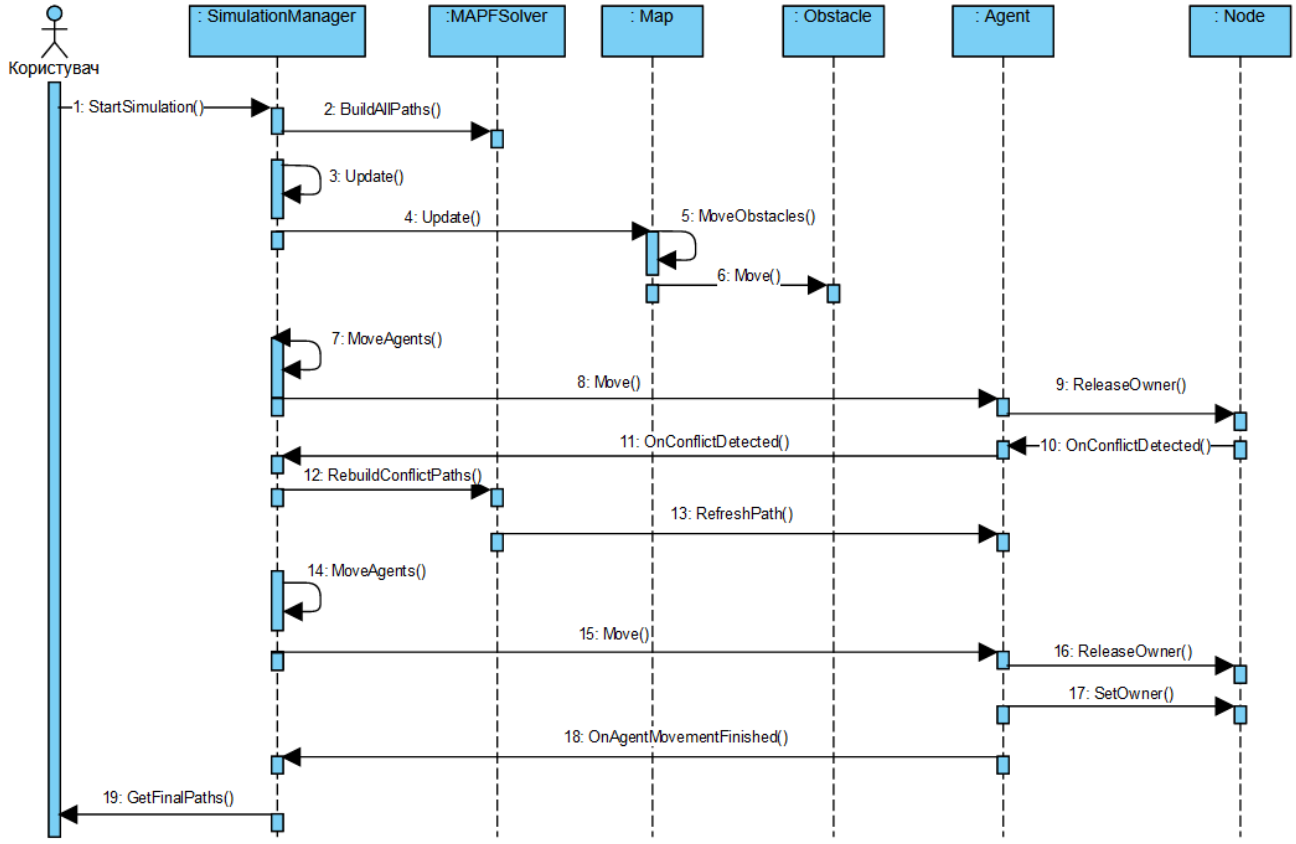

Рисунок 2.18 – Діаграма послідовностей для прецедентів «Запустити симуляцію», «Перемістити агентів» та «Вирішити конфлікти»

2.8 Діаграми кооперацій системи

Діаграма кооперації призначена для опису поведінки системи на рівні окремих об'єктів, які обмінюються між собою повідомленнями, щоб досягти потрібної мети або реалізувати певний варіант використання [21].

Кооперації (collaboration) - специфікація безлічі об'єктів окремих класів, спільно взаємодіючих з метою реалізації окремих варіантів використання в загальному контексті модельованої системи.

Поняття кооперації - одне з фундаментальних в мові UML. Мета самої кооперації полягає в тому, щоб специфікувати особливості реалізації окремих варіантів використання або найбільш значущих операцій в системі. Кооперація визначає структуру поведінки системи в термінах взаємодії учасників цієї кооперації.

На рисунку 2.19 зображена діаграма кооперацій для прецеденту «Відкриття сцени».

На рисунку 2.20 зображена діаграма кооперацій для прецедентів «Додати агента» та «Обрати кінцеву позицію».

На рисунку 2.21 зображена діаграма кооперацій для прецедентів «Запустити симуляцію», «Перемістити агентів» та «Вирішити конфлікти»

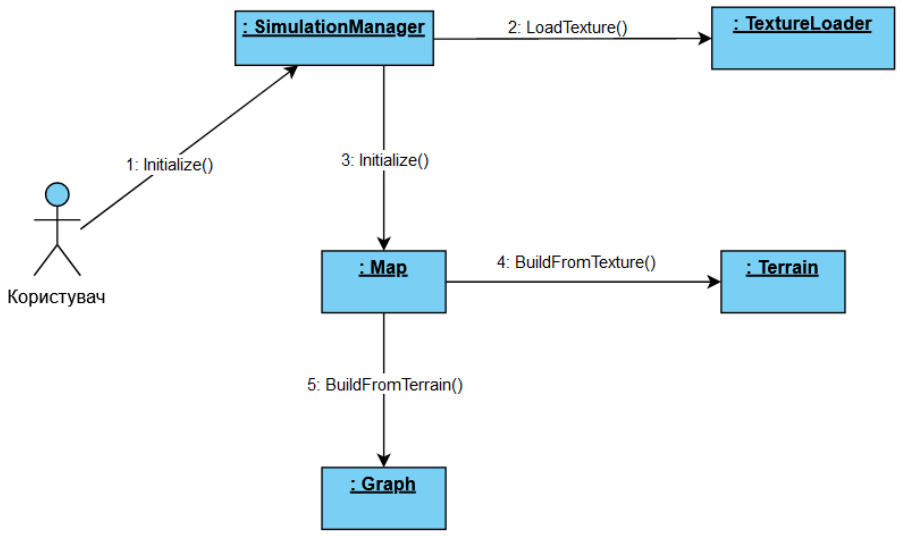

Рисунок 2.19 – Діаграма кооперацій для прецеденту «Обрати кінцеву позицію»

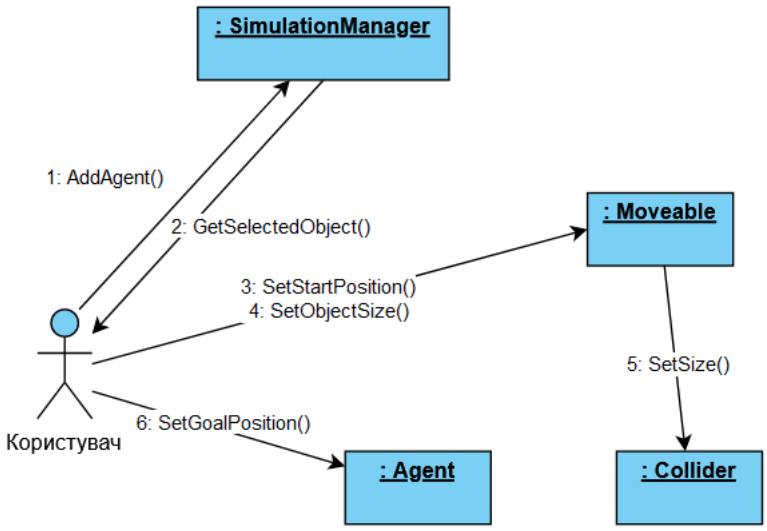

Рисунок 2.20 – Діаграма кооперацій для прецедентів «Додати агента» та «Обрати кінцеву позицію»

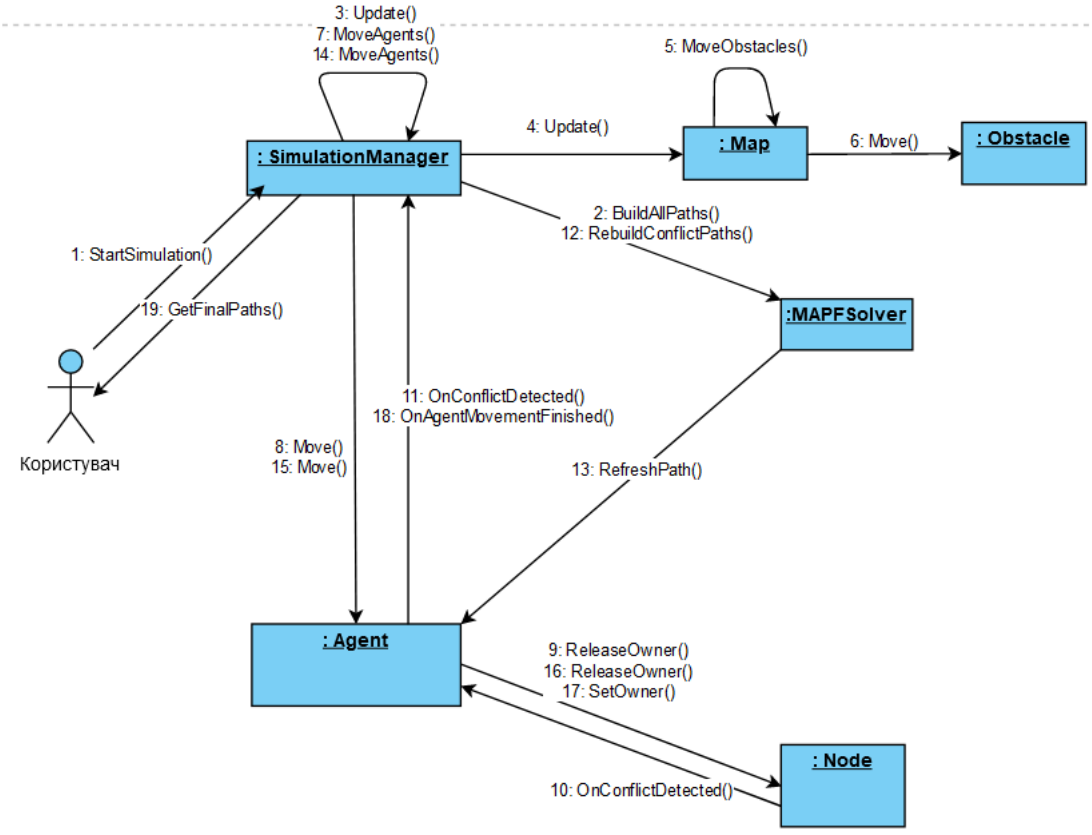

Рисунок 2.21 – Діаграма кооперацій для прецедентів «Запустити симуляцію», «Перемістити агентів» та «Вирішити конфлікти»

2.9 Діаграма станів системи

Кожна діаграма станів в UML описує всі можливі стани одного примірника певного класу і можливі послідовності його переходів з одного стану в інший, тобто моделює всі зміни станів об'єкта як його реакцію на зовнішні впливи. Діаграми станів найчастіше використовуються для опису поведінки окремих об'єктів, але також можуть бути застосовані для специфікації функціональності інших компонентів моделей, таких як варіанти використання, актори, підсистеми, операції і методи [22].

Діаграма станів зображена на рисунку 2.22.

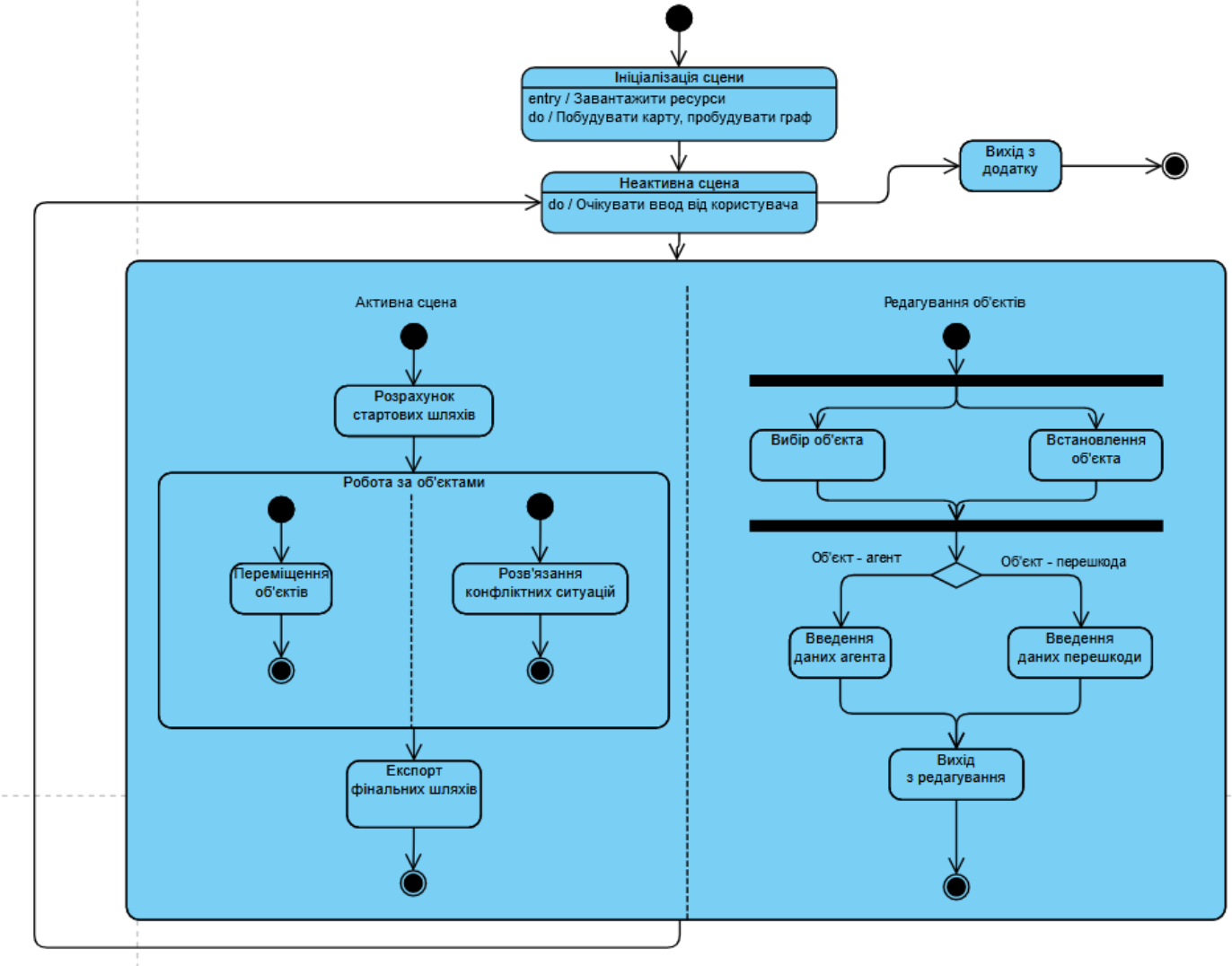

Рисунок 2.22 – Діаграма станів системи

2.10 Діаграма діяльності системи

При моделюванні поведінки системи виникає необхідність не тільки представити процес зміни її станів, але і деталізувати особливості алгоритмічної і логічної реалізації виконуваних системою операцій. Для цього раціонально буде побудувати діаграму діяльності.

Кожна діаграма діяльності акцентує увагу на послідовності виконання певних дій або елементарних операцій, які в сукупності призводять до отримання бажаного результату. Вони можуть бути побудовані для окремого випадку використання, кооперації, методу і т. д. Застосовувана в них графічна нотація багато в чому схожа на нотацію діаграм станів, оскільки діаграми діяльності є їх різновидом, але якщо на першій основна увага приділяється статичним станів, то на другий – дій [23].

Діаграма діяльності зображена на рисунку 2.23.

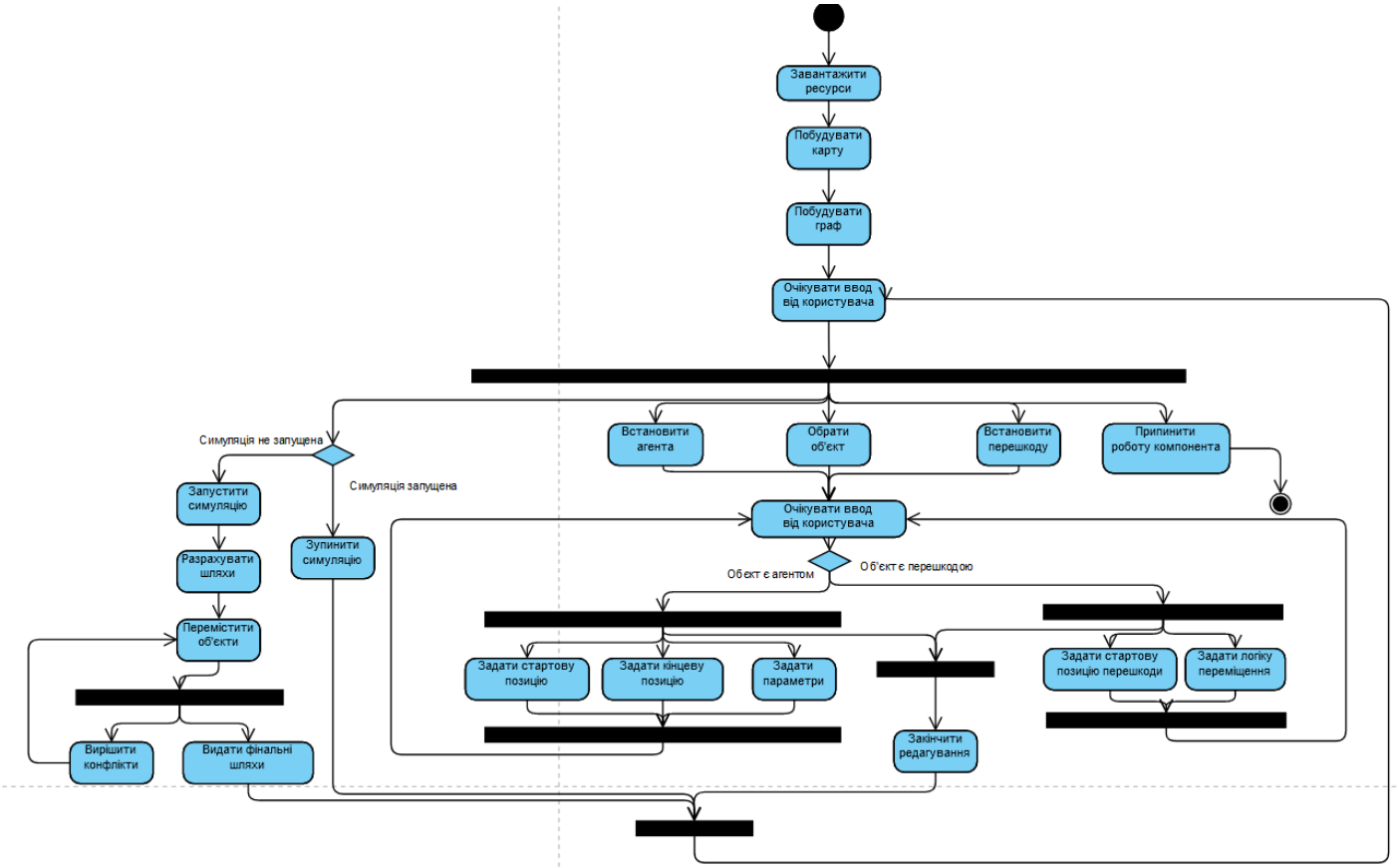

Рисунок 2.23 – Діаграма діяльності системи

3. ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

3.1 Розробка структури компонентів

Після визначення всіх функціональних вимог до компонентів та побудови необхідних діаграм, було вирішено для створення компонентів використати базову структуру ігрових двигунів.

Структура ігрових движків базується на безкінечному циклі. Для програмного додатку, що виконує пошуку шляху достатньо реалізувати основні компоненти: базове функціональне ядро мультиагентної системи, менеджер ресурсів та MAPF вирішувач.

3.2 Обґрунтування вибору мови програмування та середовища розробки

Для розробки програмної частини була обрана мова C++. C++ - це компільована, строго типізована мова програмування загального призначення. Вона підтримує такі парадигми як процедурне програмування, об'єктно-орієнтоване програмування, узагальнене програмування, забезпечує модульність, роздільну компіляцію, обробку винятків, абстракцію даних, оголошення типів об'єктів, віртуальні функції. Стандартна бібліотека включає, в тому числі, загальновживані контейнери і алгоритми. C++ поєднує властивості як високорівневих, так і низькорівневих мов. У порівнянні з його попередником - мовою C, - найбільшу увагу приділено підтримці об'єктно-орієнтованого і узагальненого програмування.

C++ широко використовується для розробки програмного забезпечення, будучи одним з найпопулярніших мов програмування. Область його застосування включає створення операційних систем, різноманітних прикладних програм, драйверів пристроїв, додатків для вбудованих систем, високопродуктивних серверів, а також розважальних програм. Існує безліч реалізацій мови C++, як безкоштовних, так і комерційних і для різних платформ.

Мова C++ була створена Бьерном Страуструпом з первинною метою позбавити себе і своїх друзів від програмування на асемблері, C або різних інших мовах високого рівня.

На думку автора мови, відмінність між ідеологією С і С++ полягає приблизно в наступному: програма на С відображає "спосіб мислення" процесора, а С++ - спосіб мислення програміста. Відповідаючи вимогам сучасного програмування, С++ робить акцент на розробці нових типів даних найбільш повно відповідних концепцій обраної галузі знань і завданням програми. Клас є ключовим поняттям С ++. Опис класу містить опис даних, потрібних для подання об'єктів цього типу і набір операцій для роботи з подібними об'єктами [24].

На відміну від традиційних структур С і Паскаля, членами класу є не тільки дані, але і функції. Функції-члени класу мають привілейований доступ до даних усередині об'єктів цього класу і забезпечують інтерфейс між цими об'єктами і решті програмою. При подальшій роботі зовсім не обов'язково пам'ятати про внутрішню структуру класу і механізм роботи вбудованих функцій. У цьому сенсі клас подібний до електричного приладу - мало хто знає про його пристрої, але всі знають, як ним користуватися.

Мова С++ є засобом об'єктного програмування, новітньої методики проектування та реалізації програм, яка в поточному десятилітті, швидше за все, замінить традиційне процедурне програмування. Головною метою творця мови доктора Бьерна Страуструпа було оснащення мови С++ конструкціями, що дозволяють збільшити продуктивність праці програмістів і полегшити процес оволодіння великими програмними продуктами.

Абстракція, реалізація, успадкування і поліморфізм є необхідними властивостями якими володіє мовою С++, завдяки чому він не тільки універсальний, як і мова С, але і є об'єктною мовою.

Серед переваг можна виділити наступні:

– обчислювальна продуктивність;

– підтримка різних стилів програмування: структурний, об'єктно-орієнтоване, узагальнене програмування, функціональне програмування, що породжує метапрограмування;

– автоматичний виклик деструкторів об'єктів (в порядку зворотному виклику конструкторів) спрощує і підвищує надійність управління пам'яттю і іншими ресурсами;

– перевантаження операторів;

– шаблони (дають можливість побудови узагальнених контейнерів і алгоритмів для різних типів даних);

– можливість розширення мови для підтримки парадигм, які не підтримуються компіляторами безпосередньо;

– доступність - для C ++ існує величезна кількість навчальної літератури, перекладеної на різні мови.

Проте, також мова має і деякі недоліки:

– погано продуманий синтаксис звужує спектр застосування мови;

– мова не містить багатьох важливих можливостей;

– мова містить небезпечні можливості;

– продуктивність праці програмістів на мові виявляється невиправдано низька;

– громіздкість синтаксису;

– майже постійна необхідність стежити за пам'яттю.

Наразі, завдяки гнучким можливостям роботи з пам’яттю та продуктивності, C++ активно використовується у сфері геймдеві. З тих же причин я вирішив також використовувати її.

В якості середи розробки обрано Visual Studio 2017. Microsoft Visual Studio — інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Вона дозволяє розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

3.3 Розробка функціонального ядра

Функціональне ядро складається з основного керуючого класу SimulationManager. Об’єкт керуючого класу також можна вважати об’єктом «сцени» бо він об’єднує у собі об'єкти карти та агентів. Він ініціює завантаження ресурсів, створення карти та дозволяє встановлювати агентів з перешкодами. Також цей клас

займається симуляцією руху усіх рухомих об'єктів у реальному часі та визиває для них побудову шляхів.

У першу чергу відразу після запуску додатку SimulationManager визиває побудову карти на основі текстури. У якості менеджера ресурсів, що займається завантаженням файлів виступає клас TextureLoader. Він завантажує із конфігураційного файлу сцени імена файлів текстур, а потім створює об’єкти текстур у пам’яті.

Конфігураційний файл містить у собі назви файлів ресурсів, основні параметри сцени та графу прохідності. Цей файл записаний у форматі JSON.

JSON (англ. JavaScript Object Notation, укр. запис об'єктів JavaScript, вимовляється джейсон) — це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією). Розробив і популяризував формат Дуглас Крокфорд.

За рахунок своєї лаконічності в порівнянні з XML, формат JSON може бути більш придатним для серіалізації складних структур.

JSON будується на двох структурах:

– набір пар назва/значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативним масивом;

– впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список, або послідовність.

Це універсальні структури даних. Теоретично всі сучасні мови програмування підтримують їх у тій чи іншій формі. Оскільки JSON використовується для обміну даними між різними мовами програмування, то є сенс будувати його на цих структурах.

У JSON використовуються такі їхні форми:

– об'єкт — це послідовність пар назва/значення. Об'єкт починається з символу «{» і закінчується символом «}». Кожне значення слідує за : і пари назва/значення відділяються комами;

– масив — це послідовність значень. Масив починається символом «[» і закінчується символом «]», а значення відділяються комами;

– значення може бути рядком в подвійних лапках, або числом, або логічними true чи false, або null, або об'єктом, або масивом. Ці структури можуть бути вкладені одна в одну;

– рядок — це послідовність з нуля або більше символів юнікода, обмежена подвійними лапками, з використанням escape-послідовностей, що починаються зі зворотної косої риски \. Символи представляються простим рядком.

Тип «Рядок» (String) дуже схожий на String в мовах C і Java. Число теж дуже схоже на C- або Java-число, за винятком того, що восьмеричні та шістнадцятирічні формати не використовуються. Пропуски можуть бути вставлені між будь-якими двома лексемами.

У сфері розробки ігрових додатків JSON швидко набув більшої популярності, аніж XML. Це видно з графіку, що зображений на рисунку 3.1. Така популярність обґрунтована тим, що його легше читати та редагувати, він, на відміну від XML.

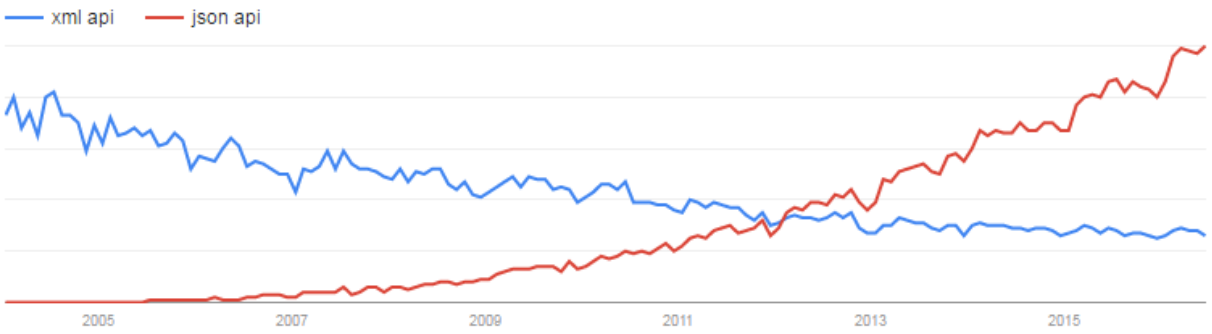

Рисунок 3.1 – Графік популярності JSON та XML при розробці ігрових додатків

Після завантаження значень із конфігураційного файлу система повинна яким-то чином завантажити специфічні файли об’єктів. Для роботи з файлами зображень була використана бібліотека SOIL2 [25].

SOIL2 (Simple OpenGL Image Library 2) - це маленька бібліотека C, яка використовується, в основному, для завантаження текстур у OpenGL. Вона може також використовуватися для збереження та завантаження зображень у різних форматах, що може бути корисно також для програм, що не працюють з OpenGL.

SOIL2 може читати наступні формати:

– BMP;
– PNG;
– JPG;
– TGA;
– DDS;
– PSD;
– HDR.

Також бібліотека підтримує можливість записувати зображення у деякі формати:

– TGA;
– BMP;
– DDS;
– PNG;
– JPG.

Окрім запису та зчитування SOIL2 має ряд позитивних особливостей, таких як:

– можливість завантажувати кубічні карти безпосередньо в текстуру OpenGL;
– дозволяє скомпонувати шість окремих файлів зображень безпосередньо в текстуру кубічної карти OpenGL;
– дає можливість взяти один файл зображення, де ширина в шість разів перевищує висоту (або навпаки), і розділити його на текстуру карти куба OpenGL;
– у бібліотеці відсутні зовнішні залежності;
– бібліотека займає дуже мало місця і має мінімум зайвого коду;
– вона багатоплатформна – підтримується системами Windows, Linux, Mac OS X, FreeBSD, Solaris, Haiku, iOS, Android, та будь-якою іншою платформою з підтримкою OpenGL.

Для завантаження зображення потрібно викликати функцію SOIL_load_image(), передати у неї назву файлу і позначити формат кольору зображення. На виході можна отримати буфер зі значеннями кольорів кожного пікселя.

Після завантажені текстури використовуються для створення ландшафту, що зберігається у класі Map і є об’єктом класу Terrain. Він містить у собі полігональну сітку, де кількість вершин дорівнює кількості пікселей у текстурі карти. Приклад полігональної сітки ландшафту зображений на рисунку 3.3. Об’єкт класу Terrain зчитує кольори із чорно-білої текстури карти висот у форматі RGB, тобто, кожній вершині умовно відводиться 3 значення у проміжку від 0 до 255. Далі для кожної з вершин виконується розрахунок усередненого значення за формулою 3.1:

$$c\frac{(r+g+b)}{3} = h\,, \tag{3.1}$$

де h – значення усередненої висоти вершини;

r, g, b – значення кольорових каналів відповідно;

c – коефіцієнт масштабу карти.

Приклад текстури висот зображений на рисунку 3.2.

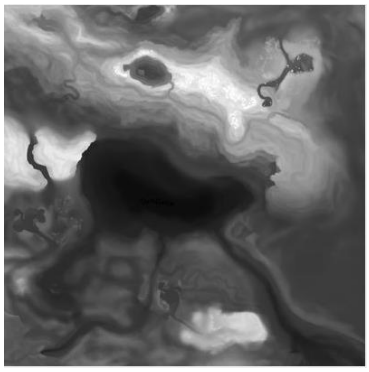

Рисунок 3.2 – Приклад карти висот розміром 400x400 пікселей.

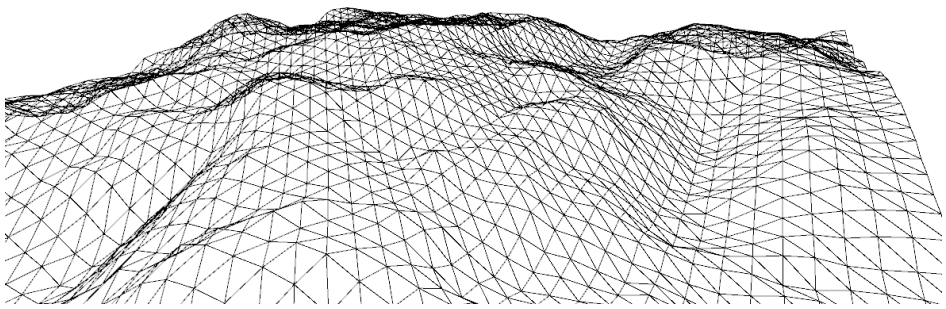

Рисунок 3.3 – Приклад полігональної сітки ландшафту.

Після цього на основі згенерованого ландшафту будується об’єкт класу Graph, в якому кожен об’єкт Node відповідає вершині полігональної сітки та має ту ж саму позицію.

Наступним кроком іде виключення зайвих вершин і зв’язків з графу прохідності. Для цього використовується чорно-біла карта прохідності. Вона повинна буди того ж розміру у пікселях, що і карта висот. Чорний колір на карті висот відповідає непрохідній місцевості, білий – прохідній. Вершини, помічені чорним кольором виключаються з графу прохідності разом з усіма своїми зв’язками. Приклад карти прохідності зображений на рисунку 3.4.

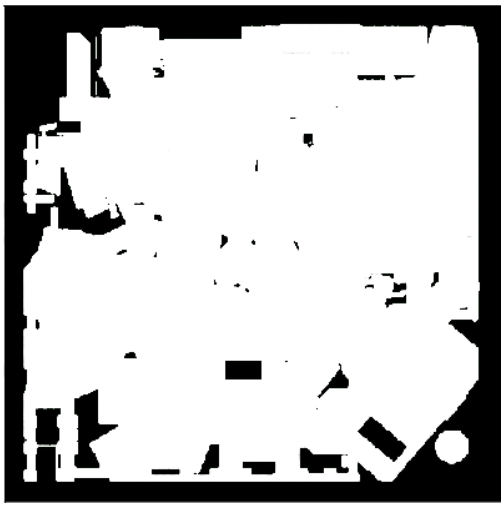

Рисунок 3.4 – Приклад карти прохідності розміром 400x400 пікселей.

Після завантаження сцени за допомогою методів AddAgent() та AddObstacle() користувач може встановити на граф агентів та перешкоди відповідно. Також з’являється можливість запустити симуляцію руху. При цьому стан сцени зміниться на Action.

Усього SimulationManager може перебувати у трьох станах:

– Prepairing – сцена завантажена, проте симуляція ще не запущена;
– Action – симуляція запущена та відбувається рух агентів;
– Finished – усі агенти дістались своєї кінцевої позиції.

Таким чином, клас SimulationManager також є містить у собі примітивну машину станів, як і об’єкти класу Agent.

Скінченна машина станів (FSM) — особливий різновид абстракції, що використовується для описання шляху зміни стану об'єкта в залежності від поточного стану та інформації отриманої ззовні [26]. Його особливістю є скінченність множини станів автомату. Поняття скінченного автомата було запропоновано як математична модель технічних приладів дискретної дії, оскільки будь-який такий пристрій (в силу скінченності своїх розмірів) може мати тільки скінченну кількість станів.

Скінченні автомати можуть розв'язувати велику кількість задач, серед яких автоматизація проектування електронних приладів, проектування комунікаційних протоколів, синтаксичний аналіз та інші інженерні застосування. В біології і дослідженнях штучного інтелекту, автомати або їх ієрархії іноді використовуються для описання неврологічних систем і в лінгвістиці для описання граматики природних мов.

Існує кілька категорій машин станів, які програмісти ігор використовують весь час. До них відносяться приймальні машини, перетворювачі та секвенсори.

Кінцевий автомат має тільки два ключових компонента. По-перше, він містить стани, які також можна назвати вузлом або вершиною. По-друге, він містить переходи, які називаються ребрами. Завжди існує один із станів, який позначено як початковий стан та один як стан виходу. При запуску FSM розпочинає роботу в початковому стані, після чого деякі події або умови призводять до переходу у інший стан. Існує багато способів зображення машини станів. Наприклад, на рисунку 3.5 зображено машину станів у вигляді графу, де “0” - це початковий стан, а “3” - кінцевий.

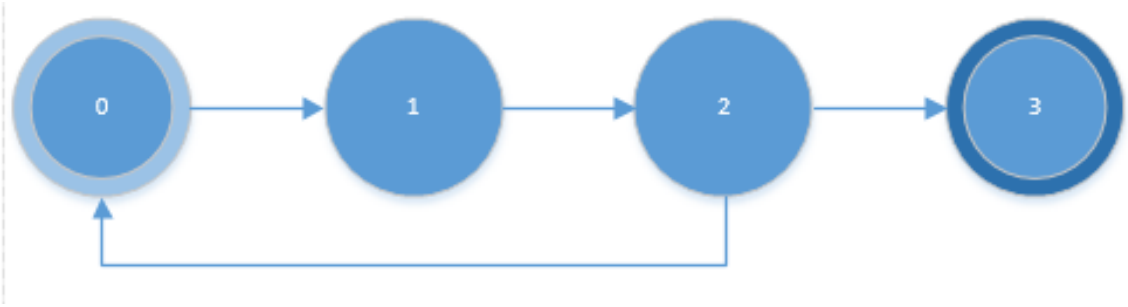

Рисунок 3.5 — Зображення FSM у вигляді графу

Агентам можна задати такі основні параметри як стартова позиція, цільова позиція, швидкість переміщення та розмір його колайдеру. Колайдер помічає точки, що знаходяться поряд з агентом у межах простору колайдеру і не являються частиною шляху агенту, як ті, що належать агенту. Така сама механіка поширюється і на рухомі перешкоди. Це зроблено для того щоб надати можливість об’єктам сцени мати свій власний об’єм. Також колайдери на об’єктах потрібні для того щоб реєструвати непередбачені зіткнення з іншими об’єктами та ініціювати перерахунок шляхів для цих агентів. Агенти мають чотири стани:

– Idle – агент перебуває у стартовій позиції і не рухається;
– Moving – агент рухається за вказаним шляхом;
– Waiting – агент очікує у точці шляху;
– OnGoal – агент очікує у кінцевій точці.

Перешкоди також можна встановлювати і редагувати як під час підготовки до симуляції, так і під час самої симуляції. Окрім таких параметрів як стартова позиція та швидкість руху, перешкоді необхідно задати точки за якими вона буде рухатися в одному з обраних варіантів переміщення:

– Straight – перешкода циклічно рухається за вказаними точками;
– Random – перешкода циклично рухається між випадково обраними точками із вказаних користувачем.

Після запуску симуляції SimulationManager звертається до об’єкта класу Solver, який за допомогою методу BuildAllPath() будує шляхи для усіх встановлених агентів. Після цього за допомогою методу MoveAllObjects() керуючий клас починає переміщення усіх рухомих об’єктів по карті. У разі виникнення колізій або додання на карту нових агентів, керуючий клас знову звертається до об’єкта класу Solver, після чого той шляхом визову методу RebuildConflictPaths() перебудовує шляхи для конфліктуючих агентів.

3.4 Розробка верхнього рівня пошуку шляху

Для створення верхнього рівня системи пошуку шляхів було вирішено використати оптимальний вирішувач MAPF з пошуком на основі конфліктів (CBS). У CBS агенти пов'язані з обмеженнями. Обмеженням для агента ai є кортеж (ai, v, ti),

де агенту ai заборонено займати вершину v під час кроку t. Послідовний шлях для агента ai - це шлях, який задовольняє всі обмеження ai, а послідовне рішення - це рішення, що складається лише з послідовних шляхів. Треба зауважити, що послідовне рішення може бути недійсним, якщо, незважаючи на те, що шляхи узгоджуються з обмеженнями окремих агентів, вони все ще мають конфлікти між агентами.

Високорівнева CBS шукає дерево обмежень (CT). CT - це бінарне дерево, в якому кожен вузол N містить:

- набір обмежень, накладених на агенти (N.constraints),
- єдине рішення (N.solution), яке відповідає цим обмеженням,
- вартість N.solution (N.cost).

Корінь CT містить порожній набір обмежень. Наступник вузла в CT успадковує обмеження батьківського об'єкта і додає єдине нове обмеження для одного агента. N.solution знаходиться шляхом низькорівневого пошуку. Вузол N - це цільовий вузол, коли N.solution є дійсним, тобто набір шляхів для всіх агентів не має конфліктів. Високий рівень CBS виконує найкращий перший пошук на дереві обмежень, де вузли упорядковуються за їхніми витратами (N.cost).

Отримавши вузол CT N, для окремих агентів викликається пошук низького рівня, щоб повернути оптимальний шлях, який узгоджується з їх обмеженнями в N. Будь-який оптимальний алгоритм пошуку шляхів з одним агентом може бути використаний як нижчий рівень CBS. Простий та ефективний вирішувач низького рівня є A* з істинною евристикою найкоротшої відстані (ігноруючи обмеження). Зв’язки між вузлами низького рівня порушуються, віддаючи перевагу шляхам із меншою кількістю конфліктів із відомими шляхами інших агентів.

Після того, як для кожного агента знайдений послідовний шлях (за низьким рівнем), ці шляхи перевіряються щодо інших агентів шляхом імітації руху агентів за їх запланованими шляхами (N.solution). Якщо всі агенти досягають своєї мети без будь-якого конфлікту, N оголошується вузлом цілі, а N.solution повертається. Однак, якщо під час виконання перевірки виявлено конфлікт для двох (або більше) агентів, перевірка зупиняється, а вузол оголошується нецільовим.

Вирішення конфлікту проходить наступним чином. Враховуючи нецільовий вузол CT, N, рішення якого, N.solution, включає конфлікт, (ai, aj, v, ti), відомо, що в

будь-якому дійсному вирішенні щонайбільше один із конфліктуючих агентів, ai або aj, може займати вершину v в момент часу t. Отже, повинно виконуватися принаймні одне з обмежень (ai, v, t) або (aj, v, t). отже, CBS генерує два нових вузли CT як дочірні елементи N, кожен додає одне з цих обмежень до попереднього набору обмежень, N.constraints. Для кожного вузла CT (крім кореневого) пошук на низькому рівні активується лише для одного агента - агента, для якого було додано нове обмеження.

3.5 Обґрунтування вибору алгоритму для низькорівневого пошуку шляху

Одним із найкращих низькорівневих алгоритмів пошуку шляху можна вважати алгоритм A*. Для використання у системах з MAPF даний алгоритм має декілька варіантів.

3.5.1 Виявлення незалежності

Розмір простору MAPF експоненціально залежить від кількості агентів. Стендлі представив структуру виявлення незалежності (ID) [27], щоб зменшити кількість агентів, які беруть участь у фактичному пошуку A*. Дві групи агентів позначаються як незалежні, якщо для кожної групи існує оптимальне рішення, при якому два рішення не конфліктують. Основна ідея ID - розділити агентів на незалежні групи. Спочатку кожен агент поміщається в свою групу. Найкоротші шляхи знаходяться для кожної групи окремо. Отримані шляхи всіх груп виконуються одночасно, поки не відбудеться конфлікт між двома (або більше) групами. Для вирішення конфлікту застосовується кілька евристичних методів. Якщо всі методи не працюють, агенти конфліктуючих груп об'єднуються в нову єдину групу. Всякий раз, коли формується нова група з $k \geq 1$ агентів, ця нова проблема k-агентів оптимально вирішується за допомогою пошуку на основі A*. Цей процес повторюється до тих пір, поки не перестануть виникати конфлікти між групами. Стендлі зауважив, що A*-пошук найбільшої групи домінує в часі вирішення всієї проблеми. Спостерігається експоненційне поліпшення показників часу при використанні ID.

3.5.2 Операторна декомпозиція

Не тільки кількість можливих станів для примірника MAPF експоненціально залежить від кількості агентів (k), а навіть коефіцієнт розгалуження даного стану може бути експоненціальним по k. Припустимо, що структура з 20 агентами знаходиться на 4-хзв'язковій сітці. У кожного агента може бути до 5 можливих переходів (4 сторони і стан очікування). Повне розширення всіх $520 = 9{,}53 \times 1014$ сусідів такого стану з обчислювальної точки зору неможливо. Крім того, кожен з агентів може рухатися до цілі, очікувати або відходити від цілі. Індивідуальне f-значення агента збільшується на нуль, один або два, відповідно. У нашому прикладі з 20 агентами будуть згенеровані дочірні елементи з 41 f-значенням.

Більшість цих дочірніх елементів, можливо, ніколи не знадобиться розширювати, якщо їх значення f більше, ніж вартість оптимального рішення. Вузли з f-значенням, що перевищує вартість оптимального рішення, позначаємо як надлишкові вузли.

Для вирішення цих проблем була введена операторна декомпозиція (OD). Агентам призначається довільний (але фіксований) порядок. Коли звичайний вузол A* розширюється, OD враховує тільки рух першого агента, що призводить до створення так званих проміжних вузлів. На цих вузлах враховуються тільки рух другого агента і генеруються додаткові проміжні вузли. Коли оператор застосовується до останнього агенту, генерується звичайний вузол. Як тільки рішення знайдене, проміжні вузли у відкритому списку не перетворюються в звичайні вузли, та тому кількість регулярних надлишкових вузлів значно скорочується. Цей варіант A * називається ODA*.

3.5.3 Покращене часткове розширення

Надлишкові вузли - це вузли з f-значенням, яке перевищує вартість оптимального рішення. Через величину фактора розгалуження кількість надлишкових вузлів в MAPF може бути дуже великим. Часткове розширення A* (PEA*) вирішує проблему надлишкових вузлів. Коли PEA* розширює вузол n, створюються b дочірніх вузлів, але тільки ті, у яких f = f (n), вставляються у

відкритий список. Решта згенерованих дочірніх елементів відкидаються, n повторно вставляється у відкритий список, але з f-вартістю його найкращого дочірнього елемента, який був відкинутий. Такий вузол може бути потім повторно розширено, але з новим значенням f. Кожен дочірній елемент даного вузла n може бути згенерований багато разів - один раз для кожного повторного розкриття n.

У доменах з великим фактором розгалуження PEA* з базовим розширенням (надалі BPEA*) отримає значне зменшення розміру відкритого списку, що, в залежності від реалізації відкритого списку, також може мати позитивні наслідки для продуктивності.

Нещодавно представлений алгоритм Розширеного Часткового Розширення A* (EPEA*) розвиває BPEA* [9]. BPEA* генерує усіх нащадків n, але тільки ті, у яких f = f (n), вставляються у відкритий список. EPEA* використовує механізм, який генерує тільки дочірні елементи з f = f (n), без створення і відкидання інших дочірніх елементів. Таким чином, кожен вузол генерується тільки один раз в процесі пошуку, і жоден дочірній елемент не створюється повторно, коли його батьківський вузол повторно розширюється. EPEA* використовує знання предметної області, щоб уникнути створення зайвих вузлів. По-перше, проводиться відмінність між звичайним значенням f (g + h) вузла n, званим його статичним значенням що позначається f (n), і значенням, яке в даний час зберігається для n у відкритому списку, званим збережене значення n і позначається F (n). На початку F (n) = f (n). При розширенні вузла n EPEA* генерує тільки дочірні вузли n_c з f (n_c) = F (n). Збережене значення n, F (n), оновлюється до f-вартості наступного кращого дочірнього елемента, і n повторно вставляється у відкритий список.

3.5.4 Використання баз даних шаблонів у MAPF

Бази даних шаблонів (PDB) [28] - це потужний метод для автоматичної побудови допустимої евристики з використанням пам'яті на основі абстракцій предметної області. Основна ідея PDB полягає в тому, щоб спочатку абстрагувати простір станів, розглядаючи тільки підмножини змінних або обмежень. Потім виконується повний пошук в ширину в абстрактному просторі станів (також відомому як простір шаблонів) від абстрактної мети. Відстані до всіх абстрактних

станів (шаблонів) обчислюються і зберігаються в таблиці пошуку (PDB). Ці значення потім використовуються на протязі всього пошуку в якості допустимої евристики для станів у вихідному просторі станів.

PDB “на вимогу” будуються під час пошуку і особливо ефективні в доменах, де абстрактний простір занадто великий, щоб повністю зберігати його в пам'яті. Спочатку спрямований пошук у просторі станів шаблону виконується від цільового шаблону до початкового шаблону (на відміну від звичайних PDB, де виконується повний пошук по ширині). Усі шаблони, знайдені в цьому пошуку, зберігаються в PDB. Потім починається головний пошук у реальному просторі станів. Оскільки генерується більше вузлів, пошук у просторі шаблонів продовжується і знаходить та зберігає все більше значень PDB.

Завдання ефективного застосування PDB до MAPF представляє наступну проблему. PDB ефективні лише тоді, коли записи PDB містять абстрактні стани, відстань до мети яких перевищує базову евристику (наприклад, метрика Манхеттена для пошуку шляху) для цього абстрактного стану. У випадку з MAPF абстрактні стани є проекціями регулярних станів на різні підмножини агентів, тоді як базовою евристикою є сума евристики окремих витрат (SIC). У термінології абстрактні стани - це композиційні агенти зі своїм місцезнаходженням. Розглянемо, як приклад, композиційний агент, що складається з двох агентів у певних місцях. Евристика SIC цього стану буде недооцінювати справжню відстань до мети лише в тому випадку, якщо між агентами існує конфлікт. Тому PDB для MAPF можуть бути ефективними лише в тому випадку, якщо вони створені для агентів, які беруть участь у багатьох конфліктах.

3.5.5 Аналіз продуктивності варіантів

Відповідно Стендлі були протестовані сітки 3x3 і 8x8 без перешкод. Результати показані в таблицях 3.1 та 3.2. Значення усереднюються по усім випадковим екземплярам (з 100), які були вирішені за 5 хвилин усіма алгоритмами (крім позначених «N/A», коли алгоритм не зміг розв’язати задачу пошуку). Кількість таких випадків показані в стовпці Ins.

Таблиця 3.1 – Порівняння по кількості згенерованих вузлів за час розв’язання.

		Кількість згенерованих вузлів			
k	Ins	A*	ODA*	BPEA*	EPEA*
Сітка 3x3					
5	100	780	255	3433	**16**
6	100	2767	1134	14126	**48**
7	100	6634	5425	42205	**144**
8	100	9003	16029	39344	**235**
Сітка 8x8					
5	100	19061	556	36948	**27**
6	100	N/A	1468	353479	**54**
7	95	N/A	2401	1676093	**71**
8	99	N/A	8035	N/A	**182**
9	94	N/A	30707	N/A	**616**
10	96	N/A	N/A	N/A	**2448**
11	81	N/A	N/A	N/A	**2739**
12	89	N/A	N/A	N/A	**11343**

Таблиця 3.2 – Порівняння по часу, витраченому на вирішення задачі.

		Час розрахунку, мс			
k	Ins	A*	ODA*	BPEA*	EPEA*
Сітка 3x3					
5	100	100	7	95	**4**
6	100	660	40	494	**25**
7	100	3593	777	2190	**176**
8	100	5187	3475	4371	**583**

Продовження таблиці 3.2 – Порівняння по часу, витраченому на вирішення задачі.

Сітка 8x8					
5	100	9311	7	743	**6**
6	100	N/A	49	6881	**17**
7	95	N/A	68	33347	**26**
8	99	N/A	593	N/A	**99**
9	94	N/A	6612	N/A	**438**
10	96	N/A	N/A	N/A	**3125**
11	81	N/A	N/A	N/A	**4324**
12	89	N/A	N/A	N/A	**24404**

З результатів обчислень видно, що ODA* швидше, ніж A*. BPEA* також швидший за A*, але він страждає від накладних витрат на генерування надлишкових вузлів та їх відкидання. EPEA* являється найшвидшим та найбільш економним серед усіх заявлених варіантів. Тому доцільно використати його.

3.6 Оптимізація розрахунків під час симуляції

Базовим рішенням вирішення конфліктів для динамічного MAPF є перепланування - створення нового екземпляру MAPF, визначеного поточними розташуваннями та цільовими розташуваннями як існуючих, так і нових агентів, та оновленого середовища, та обчислення цього екземпляра. Хоча перепланування знаходить рішення, якщо воно існує, воно не використовує попередньо обчисленні плани існуючих агентів повторно і може бути обчислювально неефективним.

Для вирішення даної проблеми було вирішено вдатися до почергових перерахунків шляхів конфліктуючих агентів. У разі якщо на карту був добавлений новий агент для нього на нижньому рівні пошуку шляху визначається найоптимальніший шлях. Далі цей шлях звіряється з шляхами інших агентів на наявність конфліктів. Якщо конфлікти знайдені, то усі конфліктуючі агенти поміщаються у конфліктний список передаються до вирішувача для перерахунку їх

шляхів за допомогою алгоритму CBS. Потім усі отримані шляхи знову звіряються зі шляхами неконфліктних агентів. Агенти з конфліктного списку, що більше не конфліктують з іншими агентами, видаляються з цього списку. Для решти конфліктних агентів відбираються агенти з карти, з якими вони конфліктують і переміщуються у список конфліктних агентів.

Дана послідовність повторюється доти, поки кількість ітерацій перерахунків конфліктного списку не перевищить вказаний у конфігураційному файлі ліміт. Якщо на цей момент для решти конфліктуючих агентів не було знайдено оптимального неконфліктного шляху, для них обирається найдешевший неоптимальний неконфліктний шлях. Для ситуації, коли треба вирішити конфлікт між агентом і рухомою перешкодою, алгоритм буде діяти таким же чином як якщо б цей агент вважався новим.

ВИСНОВКИ

У процесі виконання МАР було проаналізовано предметну область мультиагентного пошуку шляху. Були розглянуті сучасні алгоритми та вирішувачі, типи поведінок агентів та види цільових функцій, підходи до тестування методів класичного мультиагентного пошуку. Також були розглянуті методи обчислення MAPF у динамічному середовищі та існуючі аналоги.

Виходячи з аналізу предметної області, було прийнято рішення провести функціональне моделювання компонентів мультиагентної системи пошуку шляхів, побудувати діграми діаграми класів, послідовностей та кооперацій, станів системи та активностей, розробити метод, що оптимізує базовий варіант динамічного мультиагентного пошуку шляху.

Проведене функціональне моделювання системи за допомогою методології IDEF0. Була створена діаграма прецедентів, розроблена діаграма класів, послідовностей та кооперацій, створена діаграма станів системи та активностей.

Проаналізовані шляхи створення програмного продукту, зручні біблиотеки та фреймворки. В результаті був розроблений компонент, що будує граф на основі заданих користувачем текстур. Використовуючи згенерований граф, користувач має змогу розміщувати на ньому рухомі перешкоди та агентів, задавати стартові позиції для рухомих об’єктів та точки, у які ці об’єкти повинні потрапити. Також компонент дає можливість симуляції переміщення рухомих об’єктів з оптимізованим розв’язанням конфліктів, що виникають у динамічному оточенні.

ПЕРЕЛІК ДЖЕРЕЛ

1. Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A., 2016. The increasing cost tree search for optimal multi-agent pathfinding. In IJCAI,

2. Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N.R., 2018 Conflict-based search for optimal multi-agent path finding. Into appear in AAAI.

3. Bartak, R.; Svancara, J.; and Vlk, M. 2018. A scheduling-based approach to multi-agent path finding with weighted and capacitated arcs. In International Conference on Autonomous Agents and Multi Agent Systems (AAMAS).

4. Wagner, G.; Kang, M.; and Choset, H. 2012. Probabilistic path planning for multiple robots with subdimensional expansion. In IEEE International Conference on Robotics and Automation (ICRA), 2886–2892.

5. Wagner, G., and Choset, H. 2017. Path planning for multiple agents under uncertainty. In International Conference on Automated Planning and Scheduling (ICAPS).

6. Li, J.; Surynek, P.; Felner, A.; Ma., H.; Kumar, T. K. S.; and Koenig, S. 2019. Multi-agent path finding for large agents. In AAAI Conference on Artificial Intelligence.

7. Surynek, P. 2020. Multi-Goal Multi-Agent Path Finding via Decoupled and Integrated Goal Vertex Ordering.

8. Ratner, D., and Warrnuth, M. 1986. Finding a shortest solu-tion for the N×N extension of the 15-puzzle is intractable.InAAAI-86.

9. Cohen, L.; Wagner, G.; Chan, D.; Choset, H.; Sturtevant,N.; Koenig, S.; and Kumar, T. 2018b. Rapid randomized restarts for multi-agent path finding: Preliminary results. InInternational Conference on Autonomous Agents and MultiAgent Systems (AAMAS).

10. Felner, A.; Goldenberg, M.; Sharon, G.; Stutervant, N.;Stern, R.; Beja, T.; Schaeffer, J.; and Holte, R. 2012. Partial-expansion a* with selective node generation. Into appear inAAAI.

11. Basem Atiq, Volkan Patoglu, Esra Erdem, 2020: Dynamic Multi-Agent Path Finding based onConflict Resolution using Answer Set Programming

12. Yoshizumi, T.; Miura, T.; and Ishida, T. A* with partial expansion for large branching factor problems. In AAAI/IAAI

13. Fernel, A., Sharon, G., 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem.

14. Bogatarkan, A., Patoglu, V. 2019. A Declarative Method for Multi-Agent Path Finding.

15. Методология IDEF0 [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema6/tema6_2.

16. Нотация IDEF0 [Електронний ресурс] – Режим доступу до ресурсу: http://www.businessstudio.com.ua/bp/bs/overview/notation_idef0.php.

17. DFD - диаграмма потоков данных [Електронний ресурс] – Режим доступу до ресурсу: http://www.itstan.ru/funk-strukt-analiz/dfd-diagramma-potokov-dannyh.html.

18. Основы UML — диаграммы использования (use-case) [Електронний ресурс] – Режим доступу до ресурсу: https://pro-prof.com/archives/2594.

19. Диаграммы классов UML. Логическое моделирование [Електронний ресурс] – Режим доступу до ресурсу: http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=3.

20. Диаграмма последовательности (Sequence diagram) [Електронний ресурс] – Режим доступу до ресурсу: https://flexberry.github.io/ru/fd_sequence-diagram.html.

21. Диаграммы взаимодействия: крупным планом [Електронний ресурс] – Режим доступу до ресурсу: https://flexberry.github.io/ru/fd_sequence-diagram.html.

22. Теория и практика UML. Диаграмма состояний [Електронний ресурс] – Режим доступу до ресурсу: http://it-gost.ru/articles/view_articles/97.

23. Диаграмма деятельности [Електронний ресурс] – Режим доступу до ресурсу: http://studepedia.org/index.php?vol=1&post=2126.

24. C++ - Энциклопедия языков программирования [Електронний ресурс] – Режим доступу до ресурсу: http://progopedia.ru/language/c-plus-plus.

25. Simple OpenGL Image Library 2 [Електронний ресурс] – Режим доступу до ресурсу: https://bitbucket.org/SpartanJ/soil2/src/default/.

26. Understanding State Machines [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/state-machines-basics-of-computer-science-d42855debc66/.

27. Standley, T. Finding optimal solutions to cooperative pathfinding problems. In AAAI,.

28. Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. Journal of Artificial Intelligence Research.