НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВА ОСВІТИ І НАУКИ УКРАЇНИ
Кафедра комп’ютерних інтелектуальних систем та мереж

ЛЮЛЬКО Олександр Андрійович

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
ПРОГРАМНА СИСТЕМА ТРАСУВАННЯ МАРШРУТІВ

Спеціальність 123 – Комп’ютерна інженерія
Спеціалізація – Комп’ютерні системи та мережі

Керівник: Защолкін Костянтин Вячеславович,
доктор технічних наук, професор

Одеса – 2022

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ

Люлько Олександру Андрійовичу
(прізвище, ім'я, по батькові)

1.Тема проекту (роботи) Програмна система трасування маршрутів

керівник проекту (роботи) Защолкін К.В.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ректора ОНПУ від " 06 " 06.2022 № 187-в

2. Строк подання студентом проекту (роботи) 14.06.2022

3. Вихідні дані до проекту (роботи) Виконати розробку програмної системи трасування маршрутів. Виконати відлагодження та тестування розробленої програмної системи

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

1. Аналітичний огляд методів трасування маршрутів
2. Постановка та аналіз завдання кваліфікаційної роботи
3. Розробка програмної підсистеми трасування маршруту на основі методів фронту хвилі

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

1. Реалізовані варіанти процедури поширення фронту хвилі
2. Процедура прокладання маршруту
3. Діаграма використання системи трасування маршрутів
4. Структура розробленої системи трасування маршрутів
5. Діаграма послідовності системи трасування маршрутів
6. Діаграма класів головного модуля системи трасування
7. Діаграма класів модуля інтерфейсу користувача
8. Тестування основної функціональності розробленої системи

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 01.03.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналітичний огляд систем та методів	15.03.22	виконано
2	трасування маршрутів		
3	Специфікація вимог до програмної системи,	25.03.22	виконано
4	що проектується		
5	Проектування програмної системи для	15.04.22	виконано
6	трасування маршрутів		
7	Програмна реалізація системи трасування	10.05.22	виконано
8	маршрутів		
9	Оформлення пояснювальної записки	07.06.22	виконано
10			

Студент _____ Люлько О.А.
(підпис) (прізвище та ініціали)

Керівник проекту (роботи)_____ Защолкін К.В.
(підпис) (прізвище та ініціали)

Відомість кваліфікаційної роботи бакалавра

№ рядка	Найменування	Кільк.	Примітка
1	Пояснювальна записка	61	
2	Задача трасування маршруту	1	
3	Реалізовані варіанти процедури поширення фронту хвилі	1	
4	Процедура прокладання маршруту	1	
5	Діаграма використання системи трасування маршрутів	1	
6	Структура розробленої системи трасування маршрутів	1	
7	Діаграма послідовності системи трасування маршрутів	1	
8	Діаграма класів головного модуля системи трасування	1	
9	Діаграма класів модуля інтерфейсу користувача	1	
10	Розроблена структура JSON об’єкта для зберігання маршруту	1	
11	Тестування основної функціональності розробленої системи	1	
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			

					АМДР.АМ183.2808			
Зм.	Лист	№ докум.	Підпис	Дата				
Розробив		Люлько О.А.			Програмна система трасування маршрутів Відомість кваліфікаційної роботи	Літ.	Лист	Листів
Перевірив		Защолкін К.В.						
Реценз.						Одеська політехніка ІКС, Каф. КІСМ		
Н. Контр.								
Затвердив								

АНОТАЦІЯ

Люлько О.А. Програмна система трасування маршрутів – кваліфікаційна робота бакалавра. Одеса, 2022: 61 с., 10 рис., 3 табл., 1 додаток, 5 джерел.

Метою роботи є розробка програмного забезпечення, що дозволяє виконати трасування маршруту на заданій користувачем двовимірній карті з перешкодами з використанням методів фронту хвилі.

У роботі виконано аналіз існуючих програмних систем, у яких застосовується трасування маршрутів та перелічено їх недоліки. Також проведено огляд розроблених раніше алгоритмів трасування маршруту, зокрема методів фронту хвилі. В результаті одержано висновки про те, що методи фронту хвилі гарантують знаходження оптимального шляху, а також є достатньо простими для розуміння та реалізації.

Метод фронту хвилі побудований на обході вершин графу з ребрами одиничної довжини. Метод складається з двох етапів. На першому з них здійснюється поширення фронту хвилі у чотирьох або восьми напрямках шляхом дослідження сусідніх клітин. Пошук завершується при досягненні кінцевої вершини або констатації факту відсутності шляху між заданими точками. Розроблена програмна система дозволяє здійснити створення карти та трасування маршруту у звичайному або покроковому режимі і відображення процесу трасування. Розроблені формати файлів дозволяють зберегти карту або знайдений маршрут на диск, а також завантажити їх з файлу.

ЗВАЖЕНИЙ ГРАФ, ТРАСУВАННЯ МАРШРУТУ, МЕТОДИ ФРОНТУ ХВИЛІ, ПРОГРАМНА СИСТЕМА, ОБРОБНИКИ ПОДІЙ

ANNOTATION

Lyulko O.A Software system for route tracing – qualification work of bachelor. Odessa, 2022: 61 pp., 10 fig., 3 tables, 1 supplement, 5 sources.

Purpose of the qualification work to develop software that allows tracing the route on a user-defined two-dimensional map with obstacles using the methods of the wave front.

In the analysis of existing software systems, which use pathfinding, were listed their deficiencies. Also, previously developed pathfinding algorithms, in particular of the wave front, were reviewed. As a result, received the opinion that the methods of the wave front guarantee finding the optimal path, and are simple to understand and implement.

The wave front method is based on traversing the vertices of a graph with edges of single weight. The method consists of two steps. The first of these is the spread of the wave front in the four- or eight-way through the exploring of neighboring cells. Search ends when the final vertex found or a statement of fact no path between specified points. The second step is the installation of the route from the endpoint to the starting point. Algorithms of the software were constructed on the basis of used methods.

The developed software system allows maps creating, tracing the route in the normal mode or step mode process and visualization of pathfinding process. Developed file formats allow user to save a map or found route to disk, and load it from a file.

WEIGHTED GRAPH, PATHFINDING, WAVEFRONT METHODS, SOFTWARE SYSTEM, EVENT HANDLERS.

ЗМІСТ

ВСТУП

Підсистеми трасування маршруту на даний момент зустрічаються у багатьох комп'ютерних системах. Найчастіше дана задача використовується в системах автоматизації проектних робіт з розробки друкованих плат для прокладання друкованих провідників, в комп'ютерних іграх для пошуку шляху ігрових одиниць, у робототехніці для побудови маршрутів руху автономних роботів, а також у системах навігації для пошуку оптимального шляху між двома точками на електронній карті.

Пошук маршруту може виконуватись між двома вершинами зваженого графа, а також між точками двовимірної або тривимірної карти з прохідними точками та перешкодами. Така карта також може бути представлена у вигляді зваженого графа з однією вагою для переходів по горизонталі та по вертикалі, та іншою вагою для переходів по діагоналі.

Для трасування можуть використовуватись різноманітні алгоритми, що відрізняються швидкістю роботи, затратами пам'яті, оптимальністю знайденого шляху. Також деякі алгоритми однаково добре працюють як на графах єдиної вартості, так і на графах, переходи яких мають різну вагу, а інші алгоритми працюють лише на графах єдиної вартості.

Умовно алгоритми трасування маршрутів можна розділити на дві групи: ті, що здійснюють пошук у всіх напрямках рівномірно, та ті алгоритми, що виконують пошук, направлений у бік кінцевої точки за допомогою відповідних евристичних функцій. До найбільш популярних алгоритмів з першої групи можна віднести наступні: пошук в ширину, алгоритм Дейкстри, хвильові алгоритми. Серед алгоритмів напрямленого пошуку: Best-First Search, алгоритм A*, Jump Point Search. При цьому в них використовується одна з наступних евристичних функцій: Манхеттенська відстань, Евклідова відстань та відстань Чебишева. За допомогою цих функції обчислюється напрямок руху до фінішної точки.

Методи фронту хвилі використовуються для пошуку мінімального шляху в графі з ребрами одиничної довжини. Найчастіше такий граф представляє деяку площину з прохідними та непрохідними точками. Принцип роботи алгоритму полягає у рівномірному поширенні хвилі у всіх напрямках, що нагадує поширення кілець у воді від кинутого до неї об’єкта. Елементи першого фронту хвилі є джерелами вторинних хвиль, елементи другого фронту генерують хвилю третього фронту і так далі. Процес закінчується тоді, коли фронтом хвилі досягається кінцевий елемент. На наступному етапі відбувається побудова маршруту. Вона може здійснюватись як методом прямого трасування, так і методом зворотного трасування. Пряме трасування виникло історично раніше, проте мало декілька вагомих недоліків, що стосувалися вибору найкоротшого маршруту, а також виходу з глухих кутів. Пізніше виникло декілька оптимізацій цього методу, що дозволяли певним чином уникнути цих недоліків. Потім був створений метод зворотного трасування, що дозволяє знайти лише один маршрут, який буде оптимальним. Суть цього методу полягає у пошуку маршруту від кінцевої точки до початкової шляхом знаходження сусідньої вершини, що має вагу на одиницю меншу, ніж поточна.

Метою дипломної роботи є проектування і розробка програмної системи трасування маршруту на основі методів фронту хвилі, що дає можливість автоматизувати процес пошуку шляху між двома точками на двовимірній карті, що містить перешкоди.

Розроблена програмна система передбачає виконання пошуку маршруту з використанням двох методів трасування, заснованих на поширенні фронту хвилі у чотирьох та восьми напрямках. Це дозволяє оцінити та порівняти якість роботи використовуваних методів. Оскільки у програмній системі передбачено покрокове трасування маршруту, це дозволяє використовувати її для демонстрації механізму роботи хвильових алгоритмів.

1 АНАЛІТИЧНИЙ ОГЛЯД МЕТОДІВ ТРАСУВАННЯ МАРШРУТІВ

1.1 Загальні визначення

Задача трасування маршруту являє собою пошук найкращого маршруту з початкової точки до точки призначення з урахуванням заданих обмежень. Проблема трасування зводиться до задачі знаходження найкоротшого шляху між двома вузлами на зваженому графі, який є математичним подання карти маршрутів.

При цьому під зваженим графом розуміється граф, кожному ребру якого встановлено у відповідність деяке значення, яке ще носить назву вага ребра графа. Ребра пов’язують точки карти між собою та задають шляхи з’єднання на карті.

Шляхом від точки A_1 до точки A_n на графі вважається така послідовність ребер графа, що веде від A_1 до A_n, в якій кожні два сусідніх ребра мають спільну вершину і жодне ребро не зустрічається більше одного разу. При цьому вершина A_1 називається початком шляху, вершина A_n – кінцем шляху.

На практиці область трасування маршруту представляється у вигляді двовимірної карти, розділеної на комірки або клітини певної форми. Розбиття області пошуку на квадратні клітини дозволяє виконати її зберігання у пам’яті у вигляді двовимірного масиву. Перешкода, повз яку будується маршрут, є елементом карти, через який не може бути побудовано маршрут. Маршрут, який необхідно побудувати повинен проходити з початкової позиції до кінцевої, оминаючи при цьому перешкоди.

1.2 Сфери застосування трасування

Алгоритми трасування найчастіше використовуються для вирішення наких практичних задач:

– трасування друкованих плат;

– прокладання маршрутів ігрових об’єктів у відеоіграх;

– визначення маршрутів пересування автономних роботів;

– функціонування апаратно-програмних платформ навігації.

1.2.1 Використання трасування маршрутів у САПР друкованих плат

Трасування друкованих плат є покроковим процесом прокладки провідників, що використовується у системах автоматизованого проектування друкованих плат. Сучасні системи проектування мають складні та ефективні системи автоматичного трасування. При використанні автоматичного трасування програма самостійно виконує прокладання провідників між виводами скомпонованих на платі електронних компонентів, враховуючи обмеження, накладені розробником. Розробник також контролює результат трасування та за необхідності вносить корективи у вихідні параметри задачі.

Відомі алгоритми трасування друкованих плат можна умовно розбитина три великі групи.

Перша група – методи на основі хвильового алгоритму. Вони засновані на ідеях Лі та розроблені Ю.Л. Зіманом і Г.Г. Рябовим. Дані алгоритми одержали широке розповсюдження в існуючих САПР, оскільки вони дозволяють легко враховувати технологічну специфіку друкованого монтажу зі своєю сукупністю конструктивних обмежень. Ці алгоритми завжди гарантують побудову траси, якщо шлях для неї існує.

Друга група – ортогональні алгоритми, що мають більшу швидкодію, ніж алгоритми першої групи. Реалізація їх на комп’ютері вимагає в 75-100 разів менше обчислень у порівнянні з хвильовими алгоритмами. Такі

алгоритми застосовують при проектуванні друкованих плат з наскрізними металізованими отворами. Недоліки цієї групи алгоритмів пов'язані з отриманням великої кількості переходів з шару на шар, відсутністю стовідсоткової гарантії проведення трас, великим числом паралельних провідників.

Третя група – алгоритми евристичного типу. Ці алгоритми частково засновані на евристичному прийомі пошуку шляху в лабіринті. При цьому кожне з'єднання проводиться за найкоротшим шляхом, обходячи перешкоди, що зустрічаються на шляху.

1.2.2 Застосування трасування маршрутів у комп’ютерних іграх

Пошук шляху широко використовується у відеоіграх, де ігрові одиниці рухаються в реальному часі за змінних умов і знаходять маршрут навколо перешкод. Найчастіше задача пошуку шляху виникає в стратегіях реального часу (в яких гравець дає завдання ігровим одиницям рухатися через ігровий рівень, що містить перешкоди). Крім стратегій, завдання пошуку шляху, так чи інакше, зустрічається в більшості сучасних ігрових жанрів. Ігри та їх оточення стають більш складними, отже алгоритми пошуку шляху еволюціонують разом з ними, в результаті чого для вирішення цієї проблеми були розроблені різноманітні програмні пакети штучного інтелекту.

Стратегії реального часу зазвичай містять великі території з відкритим ландшафтом, в яких пошук шляху зазвичай є простим завданням. Однак у більшості випадків по карті переміщується не одна ігрова одиниця, а декілька, що створює потребу в різних набагато складніших алгоритмах пошуку шляху для уникнення заторів у вузьких областях ігрового ландшафту. У стратегіях ігровий рівень ділиться на плитки, які діють як вузли графа в алгоритмі пошуку шляху.

У жанрі 3D-шутерів використовується набагато більше закритих областей простору (або сумішей відкритих та закритих), які не так легко

розділити на вузли. Це призвело до використання сіток навігації. Вони побудовані шляхом розміщення в ігровому світі вузлів, які зберігають відомості про перелік вузлів, до яких можна дістатись від нього.

При великих розмірах карт та значній кількості ігрових одиниць, що рухаються одночасно, реалізація алгоритмів пошуку шляху спричинює високий рівень навантаження на центральний процесор і знижує швидкодію системи. Тому великі зусилля прикладаються для оптимізації алгоритмів трасування маршрутів. Зокрема застосовується попередній розрахунок, що дозволяє програмі звернутись до вже відомих даних для прискорення пошуку конкретного маршруту (наприклад, "Між точками А і Б знаходиться непрохідний рівчак, перейти його можна лише через один міст, тому маршрут обов'язково пролягатиме через міст, а не якусь іншу точку"). Такий метод дозволяє скоротити час виконання алгоритму за рахунок використовуваної пам’яті.

1.2.3 Використання трасування маршрутів у робототехніці

Одною з основних проблем мобільних пристроїв, що переміщаються самостійно, без керування з боку людини, є навігація. Для успішної навігації в просторі бортова система робота повинна вміти будувати маршрут, керувати параметрами руху, правильно інтерпретувати відомості про навколишній світ, отримані від датчиків, і постійно відстежувати власні координати.

Традиційно завдання навігації включає дві підзадачі, які можна розділити у часі: локалізацію у просторі і планування шляху. Локалізація полягає в оцінці поточного становища робота відносно деяких відомих опорних пунктів середовища.

Планування полягає в пошуку, по можливості, найкоротшого маршруту і просуванні в цільове положення. У цілеспрямованої навігації прийнято виділяти мінімум три ієрархічних рівня уявлення проблеми: прохід перешкод, локальну навігацію і глобальне планування маршруту.

При русі мобільного робота з однієї точки в іншу часто виникає проблема обходу перешкод на шляху до кінцевої точки. Серед можливих варіантів пошуку обходу є метод випадкового перебору, але при його використанні на достатньо складній території переміщення з пункту А в пункт Б може зайняти невизначено довгий час. Крім того існують інші, більш ефективні алгоритми пошуку шляху з обходом перешкод.

Алгоритми глобального планування залучають інформацію про всю місцевість, щоб визначити ділянки, по яких можливий рух, і потім обрати оптимальний шлях. Для завдання планування знайдено точні алгоритмічні рішення. Однак точні алгоритми мають велику обчислювальну складність і, крім того, вимагають точних алгебраїчних моделей перешкод. Евристичні методи не гарантують повноти пошуку та оптимальності навіть при глобальному плануванні, коли доступна вся інформація про середовище. Однак евристичні глобальні методи планування різними способами зменшують складність завдання і чутливість до помилок у вхідних даних.

З урахуванням реальних умов руху мобільних роботів задача планування поїздки стає ще більш складною. По-перше, робот виконує тільки оцінку своєї позиції, тому що його датчики не є досконалими. Крім того, якщо у середовищі присутні люди, тварини, або інші рухомі об'єкти, необхідно передбачити, як ці об'єкти будуть рухатися, щоб уникнути їх. Для того щоб врахувати ці невизначеності, необхідно застосовувати модель математичної ймовірності.

1.3 Алгоритми пошуку маршрутів

Найбільш прості алгоритми пошуку шляху оцінюють не весь шлях відразу, а лише один крок. Прикладом такого алгоритму є алгоритм

повороту Креша. Відповідно до нього, об’єкт починає рухатись у напрямку до кінцевої точки. Якщо на його шляху зустрічається перешкода, він починає рухатись праворуч доти, доки не зустрінеться вільний прохід. Використання цього алгоритму потребує значних часових витрат. Крім того, алгоритм повороту Креша не гарантує побудови маршруту навіть якщо він існує, особливо у випадку, коли перешкода має підковоподібну форму.

Більш складні алгоритми спрацьовують швидше і гарантують побудову маршруту, якщо він існує. До таких алгоритмів можна віднести:

– пошук в ширину;
– алгоритм Дейкстри;
– хвильові алгоритми.

Перевагою таких алгоритмів є можливість застосування у випадках, коли координати кінцевої точки є невідомими на момент початку пошуку. До недоліків цих алгоритмів можна віднести те, що пошук відбувається рівномірно у всіх напрямках, а не направлений у бік кінцевої точки.

Інший різновид алгоритмів відзначається ще більшою швидкістю, оскільки у них використовуються евристичні функції, що дозволяють прискорити пошук у напрямку кінцевої точки. До таких алгоритмів відносять:

– алгоритм Best-First Search;
– алгоритм A*;
– алгоритм Jump Point Search.

Недоліком цієї групи алгоритмів є необхідність знання точних координат фінішної точки для застосування евристичних функцій.

1.3.1 Пошук в ширину

Пошук в ширину (breadth-first search) - один з базових алгоритмів, на основі якого побудовано багато інших алгоритмів на графах.

Якщо задано граф G = (V, E) та початкову вершину S, алгоритм пошуку в ширину систематично обходить всі досяжні із S вершини. На першому кроці вершина S позначається, як пройдена, а в список додаються всі вершини, досяжні з S без відвідування проміжних вершин. На кожному наступному кроці всі поточні вершини списку відмічаються, як пройдені, а новий список формується із вершин, котрі є ще не пройденими сусідами поточних вершин списку. Для реалізації списку вершин найчастіше використовується черга. Виконання алгоритму продовжується до досягнення шуканої вершини або до того часу, коли на певному кроці в список не включається жодна вершина. Другий випадок означає, що всі вершини, доступні з початкової, уже відмічені, як пройдені, а шлях до цільової вершини не знайдений.

Алгоритм має назву пошуку в ширину, оскільки «фронт» пошуку (між пройденими та непройденими вершинами) одноманітно розширюється вздовж всієї своєї ширини. Тобто, алгоритм проходить всі вершини на відстані k перед тим як пройти вершини на відстані k+1.

Алгоритм у вигляді послідовності кроків виглядає наступним чином:

– крок 1: почати з довільної вершини S. Включити вершину S у чергу;

– крок 2: розглянути вершину, яка перебуває на початку черги. Нехай це буде вершина X. Якщо для всіх вершин, суміжних із вершиною X, уже визначено номери, то перейти до кроку 4, інакше – до кроку 3;

– крок 3: нехай {X,Y} – ребро, у якому номер не визначено. Призначити цьому ребру наступний номер, включити вершину Y у чергу і перейти до кроку 2;

– крок 4: виключити вершину X з черги. Якщо черга порожня, то зупинитись, інакше – перейти до кроку 2.

Послідовність обходу вершин зображено на рис. 1.1:

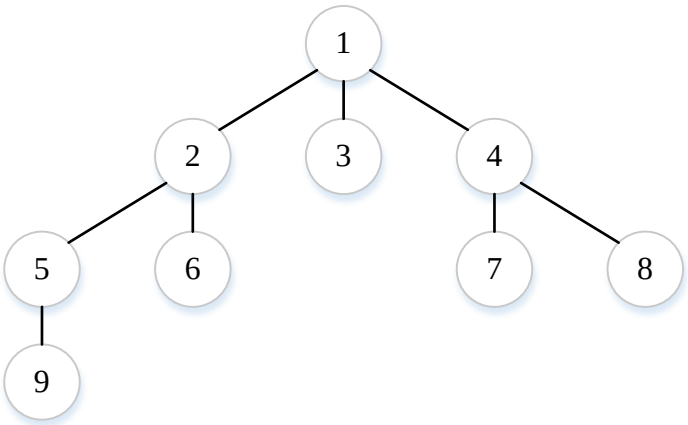

Рисунок 1.1 – Послідовність обходу вершин
графа алгоритмом пошуку в ширину

Перевагою даного алгоритму є те, що пошук в ширину завжди знаходить найкоротший шлях до цільової вершини, оскільки до переходу на рівень k+1 досліджуються всі вершини рівню k.

Недоліком є висока громіздкість такого пошуку у випадку, якщо кінцева вершина є значно віддаленою від початкової, а граф має високий коефіцієнт розгалуження.

1.3.2 Алгоритм Дейкстри

Алгоритм Дейкстри є варіантом оптимізації алгоритму пошуку в ширину. Він дозволяє знайти найкоротші шляхи до всіх вершин графа від заданої початкової вершини.

Нехай існує орієнтований або неорієнтований зважений граф з n вершинами і m ребрами. Вага всіх ребер невід’ємна. Задана деяка початкова вершина s.

Нехай у масиві d[] для кожної вершини v зберігається поточна довжина d[v] найкоротшого шляху з s до v. На початку виконання алгоритму d[s]=0, а для всіх інших вершин ця довжина дорівнює нескінченності (при реалізації на ЕОМ у якості нескінченності зазвичай використовується достатньо велике число, завідомо більше, ніж можлива довжина шляху).

Також у масиві u[] для кожної вершини v зберігається булеве значення, що вказує на те, відмічена вершина чи ні. На початку всі вершини не є відміченими.

Алгоритм Дейкстри складається з n ітерацій. На черговій ітерації обирається вершина v з найменшим значенням d[v] серед ще не відмічених (таким чином, на першій ітерації буде обрано вершину s).

Обрана таким чином вершина вважається відміченою. Потім з вершини v проводяться релаксації: переглядаються усі ребра (v, t), що виходять з вершини v, і для кожної такої вершини t одержується нове значення за формулою:

$$d[t] = \min(d[t], d[v] + len),$$

де len – довжина поточного ребра.

На цьому поточна ітерація закінчується і алгоритм переходить до наступної ітерації. Після n ітерацій всі вершини графа стають відміченими і робота алгоритму закінчується. По завершенні роботи алгоритму в масиві d[] знаходяться довжини найкоротших шляхів з вершини s. Якщо вершина v є недосяжною з вершини s, значення d[v] для неї залишається нескінченним.

Крім довжин найкоротших шляхів зазвичай необхідно знати також послідовності вершин, що відповідають цим маршрутам. Для цього достатньо так званого масиву предків: масиву p[], в якому для кожної вершини v≠s зберігається номер вершини p[v], що є передостанньою в найкоротшому шляху до вершини. Тут використовується той факт, що якщо взяти найкоротший шлях до певної вершини v, а потім видалити з цього шляху останню вершину, то вийде шлях, що закінчується деякою вершиною p[v], і цей шлях буде найкоротшим для вершини. Отже, при наявності масиву предків найкоротший шлях можна буде відновити за

ним, просто кожен раз беручи предка поточної вершини, поки черговим предком не стане стартова вершина - так отримується шуканий найкоротший шлях, записаний у зворотному порядку.

Побудова цього масиву відбувається наступним чином. При кожній успішній релаксації, тобто коли з обраної вершини v відбувається скорочення відстані до деякої вершини t, до масиву заноситься свідчення того, що предком вершини t є вершина v: p[t] = v.

На рис. 1.2 наведено приклад реалізації алгоритму Дейкстри.

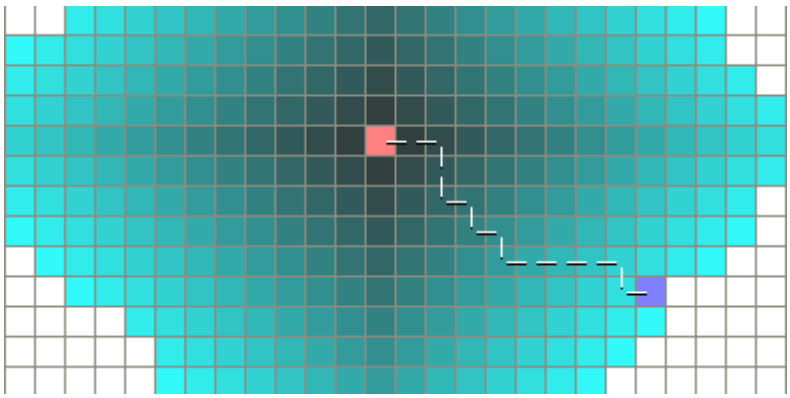

Рисунок 1.2 – Результат функціонування алгоритму Дейкстри

1.3.3 Алгоритм Best-First Search

Алгоритм Best-First Search працює подібно до алгоритму Дейкстри, за винятком того, що він має деякі евристичні оцінки того, як далеко від кінцевої знаходиться будь-яка вершина. Замість вибору вершини найближчої до відправної точки, він обирає вершину найближчу до кінцевої точки. На відміну від алгоритму Дейкстри, Best-First Search не гарантує знаходження найкоротшого шляху. Натомість, він працює набагато швидше, ніж алгоритм Дейкстри, оскільки він використовує евристичну функцію, щоб вести свій шлях до кінцевої вершини дуже швидко. Наприклад, якщо кінцева точка розташована на південь від

стартової позиції, Best-First Search буде, як правило, зосереджений на шляхах, які ведуть на південь. На рис. 1.3, жовтим представлені вузли з високою евристичної цінністю (висока вартість, щоб дістатися до мети) і чорний представляє вузли з низькою евристичної цінністю (низька вартість, щоб дістатися до мети). На цьому зображенні показано, що алгоритм Best-First Search може знайти шлях дуже швидко в порівнянні з алгоритмом Дейкстри.

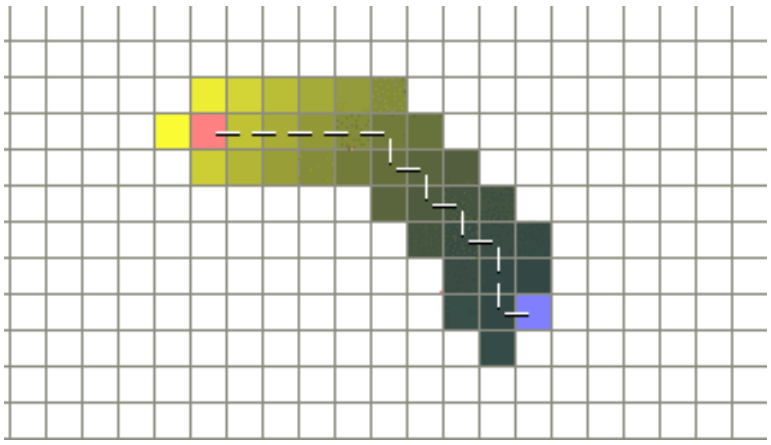

Рисунок 1.3 – Результат функціонування алгоритму Best-First Search

Однією з широко використовуваних евристичних функцій є «Манхеттенська відстань". Манхеттенську відстань можна швидко обчислити, крім того вона дає точну оцінку ймовірності того, що розглядувана клітинка знаходиться на шляху до кінцевої точки. Геометрично Манхеттенська відстань є відстанню між двома точками, при переміщенні тільки по взаємно перпендикулярним лініям (подібно вулицям Манхеттена). Математично, Манхеттенська відстань обчислюється за формулою:

$$d = |cell_x - exit_x| + |cell_y - exit_y|,$$

де ($cell_x$, $cell_y$) – координати розглядуваної точки;

($exit_x$, $exit_y$) – координати кінцевої точки.

На рис. 1.3 наведено перевагу алгоритму Best-First Search над алгоритмом Дейкстри на карті без перешкод. У випадку наявності перешкод, особливо таких, що мають П-подібну форму, ситуація змінюється. На рис. 1.4 та рис. 1.5 наведено результати виконання алгоритму Дейкстри та алгоритму Best-First Search відповідно на одній і тій самій карті.

Алгоритм Дейкстри досліджує більшу частину карти, виконує більшу роботу і, відповідно, працює довше, але результатом його виконання є оптимальний маршрут з початкової точки до кінцевої.

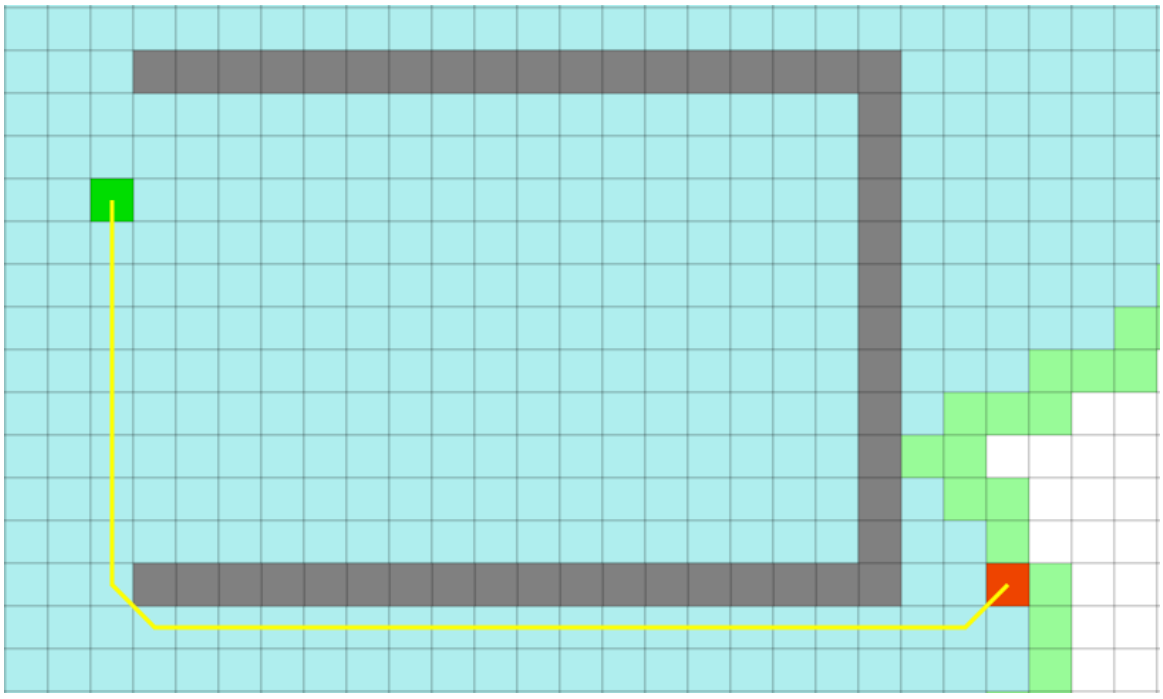

Рисунок 1.4 – Результат функціонування алгоритм Дейкстри при обході П-подібної перешкоди

На відміну від нього, алгоритм Best-First Search працює значно швидше, проте дає значно гірший результат.

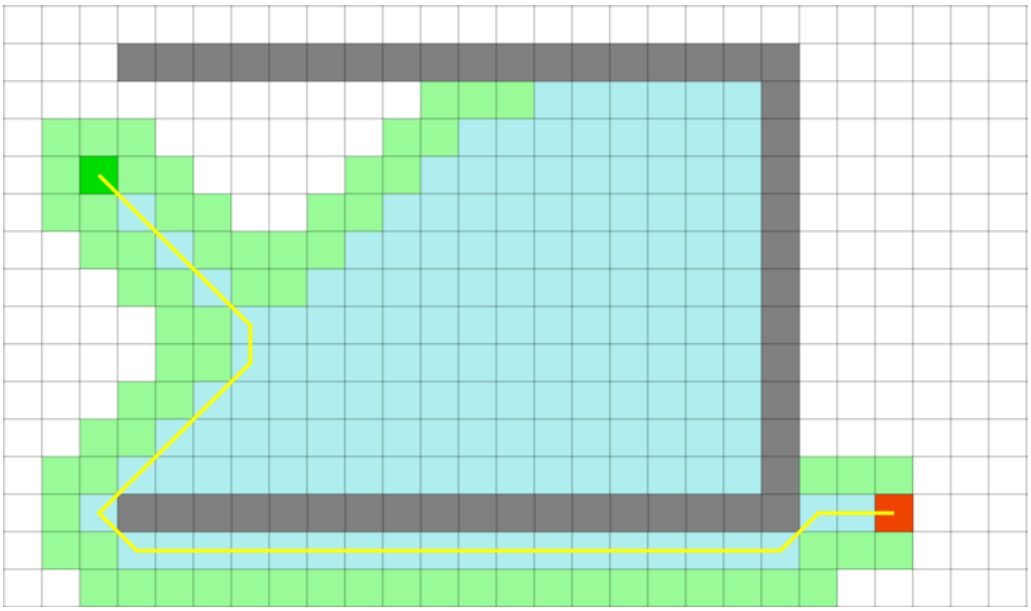

Рисунок 1.5 – Результат функціонування алгоритму Best-First Search при обході П-подібної перешкоди

1.3.4 Алгоритм A*

Алгоритм A* був розроблений в 1968 році для об'єднання евристичного підходу, такого як використовується у алгоритмі Best-First-Search і формального підходу, як у алгоритмі Дейкстри.

Зазвичай, евристичні підходи дають приблизний результат вирішення проблеми і не гарантують, отримання оптимального шляху. Тим не менш, алгоритм A*, побудований на евристичному принципі, може гарантувати отримання найкоротшого шляху.

Алгоритм A* є найбільш популярним рішенням для пошуку шляху, оскільки він є досить гнучким і може бути використаний в широкому діапазоні умов.

Як і інші алгоритми пошуку на графах, A* може потенційно виконувати пошук на величезних за площею картах. Як і алгоритм Дейкстри, він може бути використаний для пошуку найкоротшого шляху. Як і Best-First-Search, він може використовувати евристичні функції для задання напрямку. У простому випадку, він спрацьовує так само швидко, як Best-First-Search:

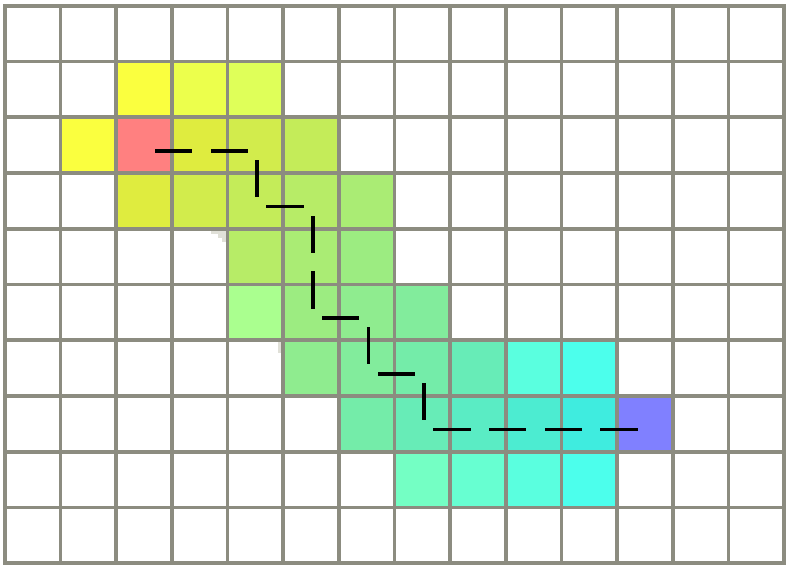

Рисунок 1.6 – Результат функціонування алгоритму A*

У випадку увігнутої перешкоди, що зустрічається на шляху, алгоритм A* спрацьовує майже так само швидко, як і алгоритм Best-First-Search, і при цьому в результаті дає найкоротший шлях. Результат пошуку шляху на тій самій карті, що і для попередніх алгоритмів, наведено на рис. 1.7.

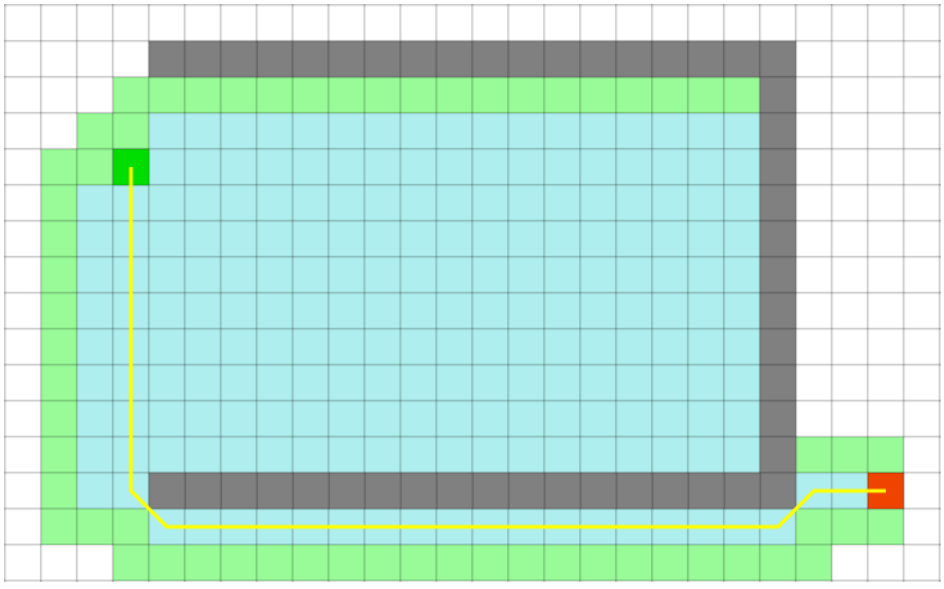

Рисунок 1.7 – Алгоритм A* при обході П-подібної перешкоди

Переваги цього алгоритму пов’язані з тим, що алгоритм A* поєднує фрагменти інформації, використовувані алгоритмом Дейкстри (зокрема, вибір на користь вершин, що є ближчими до стартової точки), та інформацію, котру використовує алгоритм Best-First-Search (вибір на користь вершин, що є ближчими до кінцевої точки). У стандартній термінології, використовуваній коли мова йде про A*, g(n) являє собою точну вартість шляху від початкової точки до будь-якої вершини n, а h(n) являє собою евристичну оцінку вартості шляху з вершини n до кінцевої вершини. Алгоритм A* балансує між цими двома метриками для знаходження оптимального шляху: на кожній ітерації знаходить вершину n, що має найменше значення f(n) = g(n) + h(n).

Алгоритм A* у вигляді послідовності кроків:

1. кожна клітина позначається станом «відкрита» чи «закрита», що характеризує її прохідність;
2. обирається стартова клітина;
3. виконується пошук відкритих сусідів поточної клітини, котрі додаються у відкритий список. Поточна клітина заноситься до закритого списку;
4. серед знайдених клітин обирається та, значення f(n) у якої найменше. Якщо обрана клітина не є кінцевою, перейти до п. 3, інакше завершити роботу алгоритму.

1.3.5 Алгоритм Jump Point Search

Цей алгоритм є покращеним алгоритмом пошуку шляху A*. JPS прискорює пошук шляху, "перестрибуючи" велику кількість місць, що повинні бути переглянуті. На відміну від подібних алгоритмів JPS не вимагає попередньої обробки і додаткових витрат пам'яті.

Алгоритм працює на неорієнтованому графі єдиної вартості. Кожне поле картки має не більше восьми сусідів, які можуть бути прохідними або

непрохідними. Кожен крок по вертикалі або по горизонталі має вартість 1; крок по діагоналі має вартість √2. Рух через перешкоди заборонений.

"Стрибкові точки" дозволяють прискорити алгоритм пошуку шляху, розглядаючи тільки "необхідні" точки. Такі точки можуть бути описані двома простими правилами вибору сусідів при рекурсивному пошуку: одне правило для прямолінійного руху і інше - для діагонального.

Для пошуку необхідних точок виконується рекурсивне відсікання сусідів навколо кожної точки. Таким чином, мета алгоритму полягає у ліквідації "симетрії", шляхом рекурсивного "перестрибування" через всі точки, в які можна потрапити з оптимального шляху, який не проходив через поточну позицію. Рекурсія зупиняється при потраплянні на перешкоду або знаходженні так званої "стрибкової точки-наступника" (англ. jump point successor). Стрибкові точки характеризуються тим, що вони мають сусідів, які не можуть бути досягнуті альтернативним шляхом: оптимальний шлях повинен проходити саме через поточну точку.

Точка y є точкою стрибка точки x в напрямку d, якщо y мінімізує кількість стрибків так, що виконується одна з таких умов:

1. точка y - точка призначення;

2. у точки y є хоча б один сусід, дістатись до якого оптимальним шляхом заважає перешкода;

3. d - рух по діагоналі й існує точка z, яка лежить в k_i кроках у напрямку d_i, такому що z - точка стрибка з y за умови 1 або 2.

Спочатку алгоритм виконує пошук наступника для поточної точки. Для цього обрізається множина сусідів, що безпосередньо прилягають до поточної точки x. Потім замість додавання кожного сусіда n до множини наступників для x, здійснюється спроба "перестрибнути" до точки, яка знаходиться далі, але лежить відносно напрямку x до n. Наприклад, якщо ребро (x; n) являє собою рух по прямій вправо від x, то розглядається точка стрибка безпосередньо праворуч від x. Якщо знаходиться така точка, то

вона додається в набір наступників замість n. Якщо до точки стрибка дійти не вдається, до множини наступників не додається жодна точка. Процес продовжується доти, поки всі сусіди не закінчаться. В результаті одержується список всіх наступників для точки x.

Для пошуку окремих наступників для точки стрибка, алгоритм намагається встановити, чи має x точку для стрибка серед наступників, переміщаючись у напрямку d і перевіряє, чи задовольняє точка n визначенню точки стрибка. У цьому випадку n позначається точкою стрибка. Інакше алгоритм рекурсивно повторюється і рухається знову в напрямку d, з використанням нової точки відліку n. Рекурсія припиняється, коли зустрічається перешкода і не можуть бути зроблені ніякі подальші дії. Перед кожним діагональним кроком алгоритм повинен виявити точки стрибка по прямих напрямах. Ця перевірка має важливе значення для збереження оптимальності алгоритму.

На рис. 1.8 наведено результат пошуку шляху на відкритій місцевості. Алгоритм Jump Point Search дає результат, аналогічний іншим алгоритмам, проте спрацьовує значно швидше.

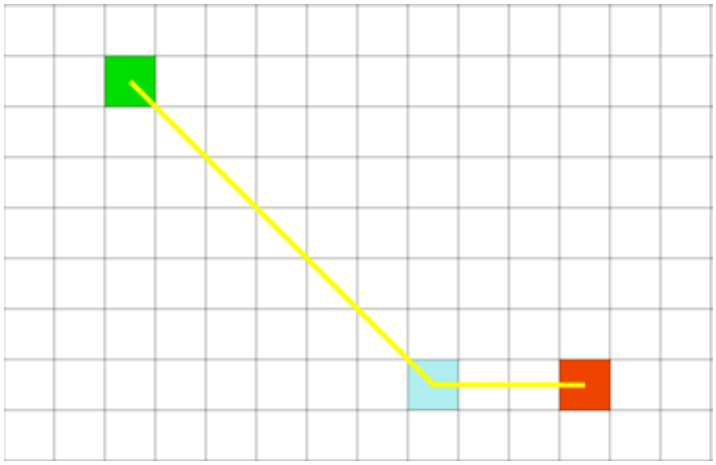

Рисунок 1.8 – Алгоритм Jump Point Search

Ще більше переваги даного алгоритму проявляються на картах з перешкодами. Шлях, зображений на рис. 1.9, є оптимальним, а його пошук виконується швидше, ніж при використанні інших алгоритмів.

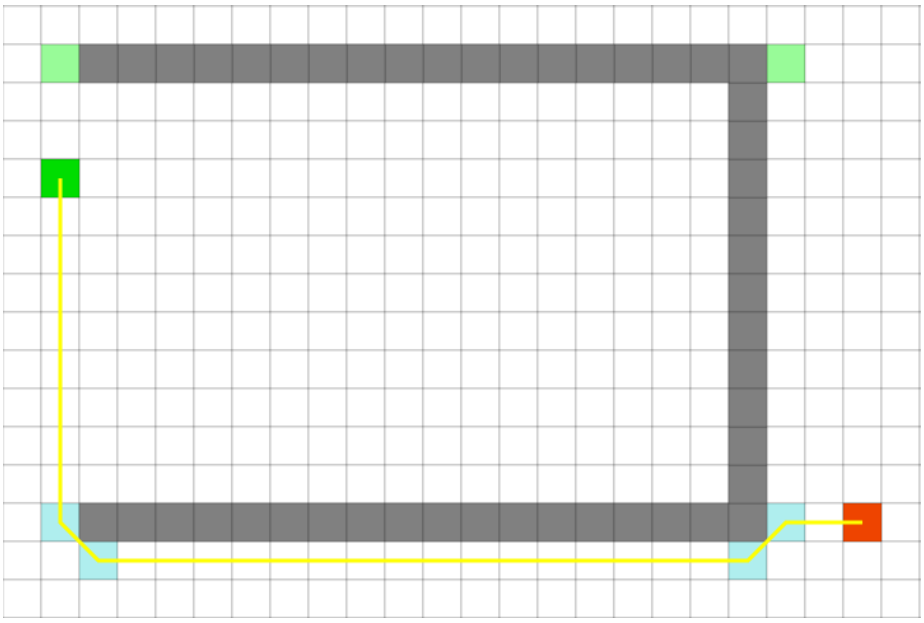

Рисунок 1.9 – Алгоритм Jump Point Search при обході П-подібної перешкоди

1.4 Висновки

У даному розділі кваліфікаційної роботи виконано аналіз предметної області роботи. Також наведено термінологію, що використовується у даній області. Крім того, наведено основні сфери застосування трасування маршрутів. У даному розділі перелічено найчастіше використовувані алгоритми трасування та виконано опис їх основних властивостей та методів роботи. Також проведено порівняння швидкодії алгоритмів та якості пошуку найкоротшого маршруту.

2 ПОСТАНОВКА ТА АНАЛІЗ ЗАВДАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

2.1 Завдання на розробку програмної підсистеми трасування маршруту

2.1.1 Призначення та галузі використання розробки

В кваліфікаційній роботі необхідно розробити програмну систему, призначену для трасування маршрутів на основі методів фронту хвилі. Трасування маршруту є одним із термінів штучного інтелекту, що означає пошук комп’ютерною програмою оптимального шляху між двома точками. На практиці трасування може використовуватись для пошуку маршруту з однієї точки до іншої у двовимірному або тривимірному просторі.

Трасування маршрутів застосовується для розв’язання наступних задач:

– прокладання провідників при трасуванні друкованих плат;

– пошук шляху для ігрових одиниць в комп’ютерних іграх;

– пошук маршруту для переміщення автономних роботів по місцевості, карта якої є наперед заданою, або будується роботом самостійно.

У даній кваліфікаційній роботі необхідно реалізувати систему пошуку маршруту у двовимірному просторі на основі методів фронту хвилі, що поширюється у чотирьох, або восьми напрямках. Також програмна система повинна передбачати можливість створення, редагування, збереження та відтворення карт місцевості, на якій необхідно побудувати маршрут. Крім того, необхідно надати користувачеві можливість проводити трасування у покроковому режимі, що дозволить йому краще зрозуміти принцип дії хвильових алгоритмів пошуку шляху.

Існуючими аналогами розроблюваної системи у певній мірі є підсистеми пошуку шляху у системах автоматизованого проектування друкованих плат та відеоіграх. Недоліком цих систем є закритість вихідного програмного коду, що не дозволяє проводити вивчення використовуваних методів пошуку шляху. Також існує незначна кількість безкоштовних програмних засобів, що дозволяють виконати трасування маршруту. Проте вони дозволяють одержати лише поверхневе розуміння застосовуваних алгоритмів, в них не приділяється уваги пошуку маршрутів на основі методів фронту хвилі. Також ці системи не дозволяють виконати збереження та відтворення карт та одержаних маршрутів.

Розроблена програмна система трасування маршруту може бути використана для вивчення механізму роботи хвильових алгоритмів, а також застосована у якості підсистеми пошуку шляху у комп’ютерних іграх, у складі навігаційної підсистеми автономного мобільного робота.

2.1.2 Мета та задачі розробки

Мета даної кваліфікаційної роботи полягає в створенні програмної системи трасування маршруту на основі методів фронту хвилі. Завдяки використанню розробленої системи передбачається отримання наступних практичних результатів:

– можливість одержати оптимальний маршрут між двома точками на заданій карті;
– можливість наочно продемонструвати методику роботи хвильових алгоритмів пошуку шляху при їх вивченні.

Для досягнення поставленої мети в кваліфікаційній роботі необхідно вирішити наступні задачі:

– провести аналіз методів трасування маршрутів, зокрема методів фронту хвилі;

– розробити алгоритми функціонування програмної системи трасування маршруту;

– обрати платформу та середовище розробки, а також мову програмування, з використанням яких проводитиметься розробка програмної системи, і виконати обґрунтування вибору;

– виконати програмування зазначеної програмної системи;

– провести тестування отриманого програмного забезпечення;

– за результатами вирішення попередніх задач оформити пояснювальну записку та графічні матеріали кваліфікаційної роботи.

2.1.3 Загальні вимоги до розробки

Розроблювана програмна система трасування маршруту повинна забезпечувати виконання набору функцій, необхідних для пошуку шляху між двома точками на заданій користувачем двовимірній карті.

В системі потрібно реалізувати два методи трасування на основі хвильового алгоритму: з поширенням фронту хвилі у чотирьох та у восьми напрямках.

Робота з системою повинна полягати в:

– створенні або завантаженні карти місцевості та зазначенні початкової і кінцевої точок маршруту;

– виборі алгоритму трасування;

– виборі режиму відображення процесу трасування.

Розроблена система повинна забезпечувати наступну функціональність:

– введення та редагування карти шляхом розміщення стартової та фінішної точок, а також розміщення та видалення перешкод з використанням відповідного графічного інтерфейсу;

– можливість збереження карти у файл та її завантаження з файлу;

– вибір алгоритму трасування та режиму відображення (звичайний або покроковий);

– пошук маршруту обраним методом;

– відображення результату пошуку у графічному вигляді та додаткової інформації у текстовому вигляді;

– збереження знайденого маршруту в файл та його відображення з файлу.

До програмного продукту, що розробляється у межах даної кваліфікаційної роботи пред’являються наступні вимоги: операційна система родини Microsoft Windows; платформа – PC. Для реалізації програмної системи використовується середовище розробки Microsoft Visual Studio, мова програмування – C# та платформа .NET Framework.

2.1.4 Вхідні та вихідні дані системи, що розробляється

Вхідними даними системи трасування, що розробляється є:

– дані про конфігурацію карти: розмір, координати перешкод;

– параметри алгоритму трасування

Вхідними даними системи трасування, що розробляється є:

– дані про координати комірок карти по яким прокладено маршрут.

Зазначені вхідні данні можуть завантажуватися в систему користувачем або зовнішньою інформаційною системою.

2.2 Хвильовий алгоритм

Хвильовий алгоритм – алгоритм, що дозволяє знайти мінімальний шлях в графі з ребрами одиничної довжини. Базується на алгоритмі пошуку в ширину. Застосовується для знаходження найкоротшого шляху в графі. В загальному випадку знаходить лише його довжину.

Хвильовий алгоритм – один з основних при автоматизованому трасуванні друкованих плат. Він має досить велику кількість різноманітних модифікацій, що дозволяють покращити знайдений розв'язок з точки зору затрат пам'яті та швидкодії. Також одне з характерних застосувань хвильового алгоритму – пошук найкоротшої відстані на карті в стратегічних комп'ютерних іграх.

Використовуючи хвильовий алгоритм, можна знайти трасу в лабіринті з будь-якою кількістю стін. У цьому й полягає перевага його використання. Недолік алгоритму полягає в тому, що при побудові маршруту потрібен великий об’єм пам'яті.

2.2.1 Поширення хвилі

Вхідними даними для алгоритму є двовимірна карта, що складається з «прохідних» і «непрохідних» (перешкоди) комірок. На ній позначені стартова та фінішна точки. Прохідні комірки мають вагу «0», перешкоди – «-1», стартова комірка – «1», комірка фінішу має високу вагу, завідомо вищу за максимальну можливу довжину маршруту. Метою алгоритму є прокладання найкоротшого шляху від стартової до фінішної точки, якщо такий шлях існує.

Точка отримує поточне значення (маркер), яке означає відстань від стартової комірки карти. При досягненні в результаті поширення хвилі фінішної точки, здійснюється перехід до другого етапу алгоритму – зворотне трасування з пошуком шляху. Якщо ж на певному етапі для всіх комірок в околиці не залишилось вільних сусідніх комірок, і кінцевої точки не знайдено, то шлях між заданими точками прокласти неможливо.

Якщо комірка має вагу «0», їй присвоюється чергове значення, що вказує на відстань до стартової точки. Комірки з іншою вагою (заповнені на попередніх кроках), а також комірки, відмічені як «непрохідні», залишаються без змін. Якщо одна з комірок є точкою фінішу, алгоритм переходить на наступний етап – побудову шляху. Якщо ж на певному етапі для всіх точок в околі не залишилось вільних комірок, і кінцевої точки не знайдено, то вважається, що шлях між заданими точками прокласти неможливо.

На рис. 2.1 та рис. 2.2 зображено поширення фронту хвилі у чотирьох та восьми напрямках відповідно.

5	6	7	8	9	10	11	12	13	14						
4	-1	-1	-1	-1	-1	12	13	14							
3	2	1	2	3	-1	13	14								
4	3	2	3	4	-1	14									
5	4	3	4	5	-1	13	14								
6	5	4	5	6	-1	12	13	14							
7	6	5	6	7	-1	11	12	13	14						
8	7	6	7	8	9	10	11	12	13	14					
9	8	7	8	9	10	11	12	13	14						
10	9	8	9	10	11	12	13	14							
11	10	9	10	11	12	13	14								
12	11	10	11	12	13	14									
13	12	11	12	13	14										
14	13	12	13	14											
	14	13	14												
		14													

Рисунок 2.1 – Поширення фронту хвилі
у чотирьох напрямках

4	4	5	6	7	8	9									
3	-1	-1	-1	-1	-1	9									
3	2	1	2	3	-1										
3	2	2	2	3	-1										
3	3	3	3	3	-1	9	9	9							
4	4	4	4	4	-1	8	8	9							
5	5	5	5	5	-1	7	8	9							
6	6	6	6	6	6	7	8	9							
7	7	7	7	7	7	7	8	9							
8	8	8	8	8	8	8	8	9							
9	9	9	9	9	9	9	9	9							

Рисунок 2.2 – Поширення фронту хвилі
у восьми напрямках

Про ефективність алгоритму свідчить те, що після досягнення хвилею фінішу значна частина карти залишилась непереглянутою. Метод з поширенням хвилі у восьми напрямках має більшу швидкодію, оскільки потребує меншої кількості кроків та переглядає менше клітин на карті.

2.2.2 Зворотне трасування

Для підвищення ефективності побудови шляху використовується принцип зворотного трасування, що полягає в пошуку шляху від кінцевої точки до початкової. Оскільки шлях зі стартової точки може поширюватися у декілька напрямків, постає задача вибору такого маршруту, що приведе до кінцевої точки. Початок побудови маршруту з кінцевої точки гарантує потрапляння до стартової точки найкоротшим шляхом. Також цей метод дозволяє уникнути глухих кутів, у які можна потрапити при використанні прямого трасування при виборі невірного напрямку.

Метод пошуку маршруту полягає в наступному. Для кінцевої точки, що має вагу k, обирається комірка з її околу (з вагою k-1). Для обраної комірки виконується пошук суміжної з вагою k-2. Таким чином продовжується доти, доки не буде знайдено комірку з вагою «1», що відповідатиме вазі стартової точки. В результаті одержується маршрут, що належить до множини найкоротших.

На рис. 2.1 та рис. 2.2 маршрути, знайдені методом зворотного трасування, позначено блакитним кольором.

2.3 Висновки

У даному розділі виконано постановку завдання на кваліфікаційну роботу та його аналіз. У рамках постановки завдання наведено перелік функцій, які повинна виконувати програмна система, обрано програмну та апаратну платформи, для яких виконується розробка, а також середовище розробки та мову програмування. Також виконано опис використовуваного алгоритму трасування маршруту та наведено приклади його роботи.

3 РОЗРОБКА ПРОГРАМНОЇ ПІДСИСТЕМИ ТРАСУВАННЯ МАРШРУТУ НА ОСНОВІ МЕТОДІВ ФРОНТУ ХВИЛІ

3.1 Загальна структура розроблюваного програмного забезпечення

Програмна система повинна складатися з двох модулів: основного модуля та модуля, що забезпечує інтерфейс взаємодії з користувачем та візуалізацію трасування маршруту. Основний модуль приймає на вхід масив даних про карту, що включають в себе інформацію про стартову та кінцеву точку, а також місця розташування перешкод. Також на вхід основного модуля надходять дані про використовуваний метод трасування. Результатом його роботи є масив даних з зазначенням знайденого маршруту.

Інтерфейсний модуль приймає на вхід текстовий файл карти місцевості та виконує її відображення на екрані комп’ютера. Також цей модуль надає користувачеві інтерфейс для редагування карти та задання параметрів трасування. Одержані від користувача дані про маршрут інтерфейсний модуль передає на вхід основного модуля. Після завершення роботи основного модуля інтерфейсний одержує від нього результат пошуку маршруту та виконує його відображення.

Для відображення загальної структури програмної системи доцільно навести діаграму класів аналізу (рис. 3.1). Керуючий клас Tracer входить до складу основної підсистеми і виконує пошук оптимального шляху між двома точками на заданій карті методом побудови фронту хвилі. За допомогою подій (англ. event) клас повідомляє інтерфейсному модулю про виникнення помилок у ході трасування, завершення чергового кроку трасування, а також завершення пошуку маршруту.

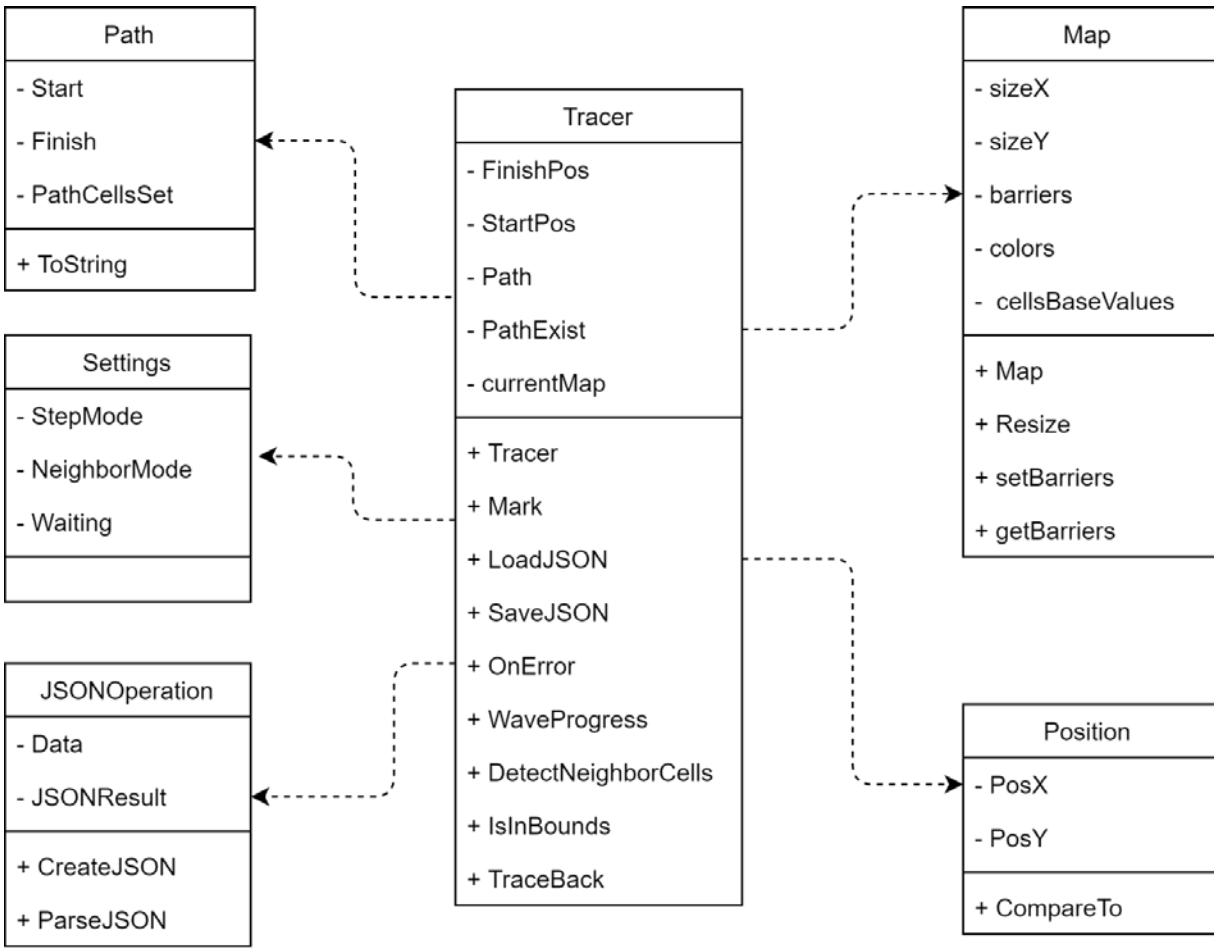

Рисунок 3.1 – Діаграма класів основного модуля системи, що розробляється

Клас сутності Position являє собою структуру, що містить координати точки на карті і використовується для зручного представлення оброблюваних позицій карти в основному та інтерфейсному модулях. Статичний клас сутності Data використовується для обміну даними між формами інтерфейсного модуля та форм з основним модулем. Граничні класи MainForm та NewMapForm наслідують стандартний клас Form інтерфейсу Windows Forms для реалізації віконного графічного інтерфейсу користувача.

3.2 Розробка інтерфейсу користувача

3.2.1 Розробка інтерфейсу головного вікна

Інтерфейс організований з застосуванням стандартних та розроблених програмни класів Діаграма класів наведена на рис. 3.2. Для відображення карти на екрані, а також для її редагування пропонується застосовувати стандартний візуальний компонент PictureBox бібліотеки Windows Forms. Для зменшення витрат пам’яті розмір компонента pbField типу PictureBox та зображення у ньому є фіксованим, тому в загальному випадку в кожен момент часу активною є лише частина карти.

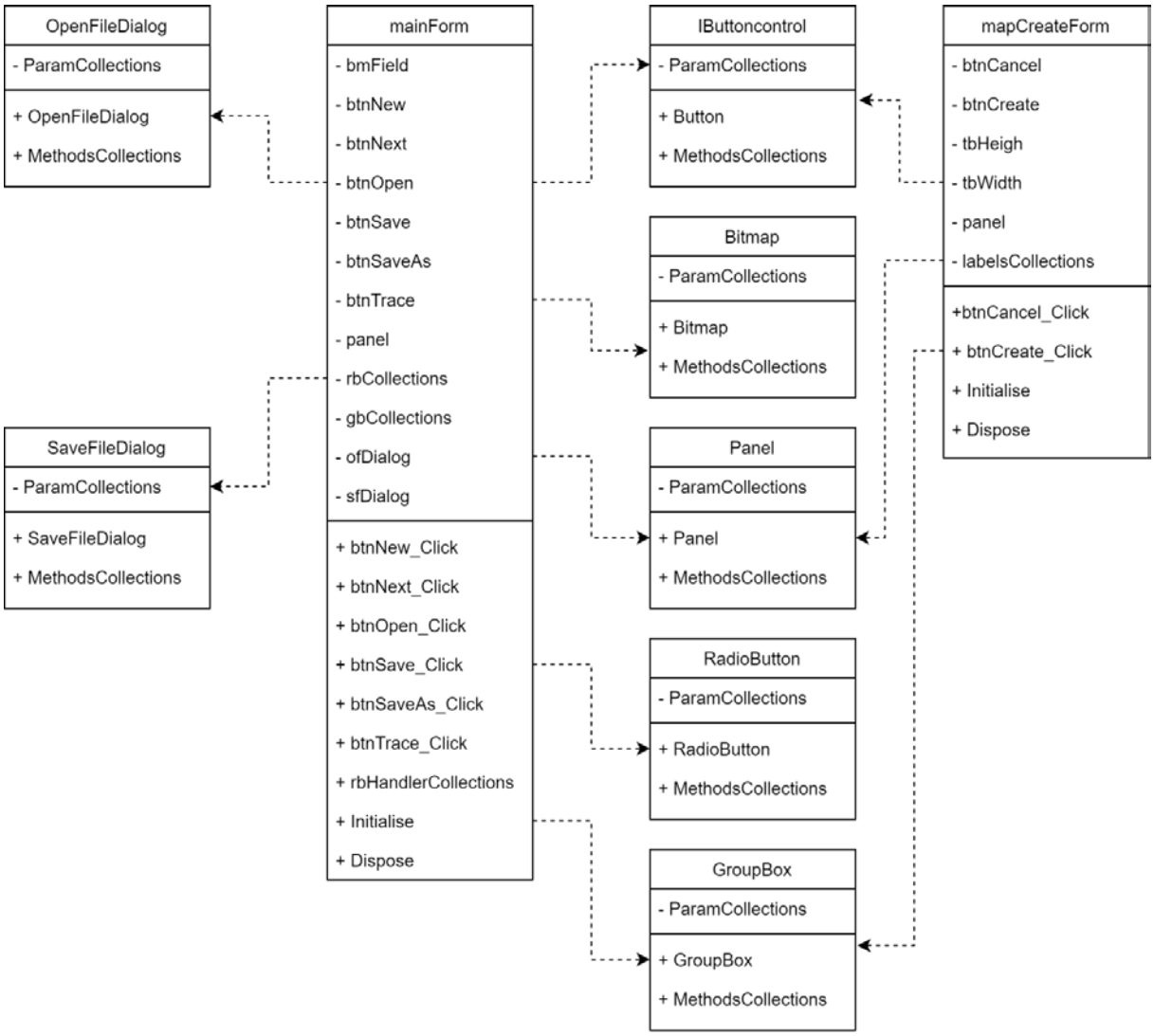

Рисунок 3.1 – Діаграма класів інтерфейсного модуля

Оскільки у програмі передбачено зміну масштабу відображення карти, доцільно виводити на екран інформацію про поточний масштаб. Для цього використовуються компонент StatusStrip, що являє собою рядок стану, розташований на нижній межі головного вікна програми. В межах цього рядку можуть бути розташовані компоненти ToolStripStatusLabel, в яких відображається текст, заданий властивістю Text, що має тип даних String. Саме ці компоненти використовуються для виводу поточних значень масштабу та відстані фронту хвилі від стартової точки.

В програмі передбачено чотири режими редагування карти:

– розміщення стартової точки;
– розміщення фінішної точки;
– розміщення перешкод;
– видалення перешкод.

Кожному з цих режимів відповідає один компонент типу RadioButton, що являє собою фіксовану кнопку з двома станами: відмічена або невідмічена. При розміщенні цих елементів в межах компонента GroupBox вони об’єднуються в масив кнопок, для яких задане загальне правило поведінки. Це правило полягає у тому, що в один момент часу може бути натиснутою лише одна кнопка, натискання іншої кнопки призводить до відпускання попередньої кнопки.

Аналогічним чином, на базі елементів RadioButton та GroupBox, побудований механізм перемикання режиму трасування: з поширенням хвилі у чотирьох або восьми напрямках.

У межах компоненту GroupBox з назвою «Трассировка» містяться компоненти, що відповідають за початок трасування маршруту та режим його відображення. Увімкнення та вимкнення покрокового режиму трасування здійснюється за допомогою компонента CheckBox. Цей компонент являє собою фіксовану кнопку з двома станами, але, на відміну від компонента RadioButton, поточний стан цього компонента може бути змінений шляхом повторного натискання. Для керування процесом

трасування передбачено компоненти Button, що являють собою кнопки, при натисканні яких виконується певна функція. При натисканні на кнопку з написом «Старт» програма запускає на виконання основний модуль, що виконує процес трасування. У випадку, якщо обрано звичайний режим трасування, розповсюдження фронту хвилі виконується без участі користувача і кнопка «Дальше» є неактивною. Інакше кожен крок трасування ініціюється натисканням на кнопку «Дальше». У будь-якому випадку перед початком зворотного трасування пошук маршруту призупиняється, що дозволяє користувачеві переглянути процес зворотного трасування покроково, або одразу одержати результат.

У головному вікні розташовані кнопки типу Button, за допомогою яких користувач може швидко створити карту або завантажити її з файлу, а також зберегти карту до нового файлу або зберегти зміни карти в поточному файлі.

Підпункти пункту «Карта» дозволяють створити, відкрити або зберегти карту, а також завершити роботу з програмою. Підпункти меню «Маршрут» дозволяють виконати збереження знайденого маршруту до файлу, а також переглянути знайдений маршрут, завантаживши його з файлу.

При завантаженні програми заголовок головного вікна містить назву програмної системи – «Tracer». Створення нової карти призводить до того, що відбувається зміна заголовка на «Новая карта – Tracer». При збереженні карти до файлу заголовок вікна програми змінюється на таке, що містить у собі назву файлу карти та назву програми. Зміна заголовку здійснюється шляхом доступу до властивості Text класу MainForm.

Клас MainForm також має події DragEnter та DragDrop, що дозволяють виконати відкриття файлу шляхом перетягування його піктограми на площину вікна програми.

Всі використовувані компоненти мають властивість Enabled, що дозволяє зробити деякі компоненти неактивними, коли виконання відповідної їм дії не є можливим.

Також взаємодія з користувачем реалізується за допомогою діалогових вікон, що реалізуються класом MessageBox. За допомогою цих повідомлень програма пропонує користувачеві зберегти зміни у карті, інформує його про необхідність встановити початкову і кінцеву точки маршруту перед початок трасування, а також повідомляє про помилки, які виникли у ході трасування або про успішно знайдений маршрут.

На рис. 3.3 зображено інтерфейс головного вікна системи.

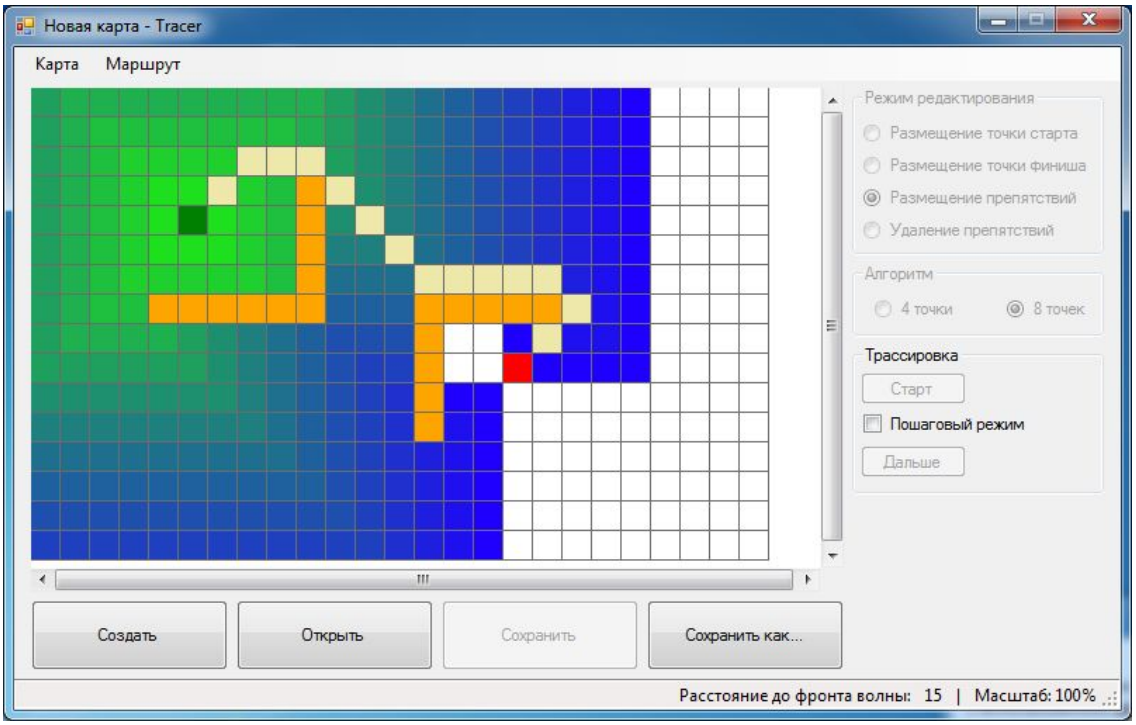

Рисунок 3.3 – Головне вікно системи

3.2.2 Розробка інтерфейсу вікна створення нової карти

Вікно створення нової карти викликається шляхом натискання на кнопку «Создать» в головному вікні, або вибору пункту «Создать» головного меню. На діаграмі класів це вікно представлене граничним гласом NewMapForm. Також інтерфейс створення нової карти передбачає використання двох кнопок – компонентів типу Button. Кнопка «Отмена»

скасовує операцію створення нової карти та повертає фокус до головного вікна програми. При натисканні кнопки «Создать» виконується створення нового поля з заданими розмірами та передача фокусу головному вікну програми.

Одночасно з активацією головного вікна діалогове вікно створення нового поля закривається і вивантажується з оперативної пам’яті комп’ютера.

3.3 Розробка головного модуля

Структуру головного модуля наведено на рис. 3.4. Основним програмним класом цього модуля є клас Tracer. Він забезпечує функціональність побудови маршруту та додаткові функціональності інтерфейсного плану відповідно до діаграми послідовності рис. 3.5.

3.3.1 Конструктор класу Tracer

Для обробки кожної нової карти створюється власний об’єкт класу Tracer. Для цього конструктору передаються два цілочисельних параметри, що задають розміри карти – кількість клітин в ширину та в висоту. Оскільки на початку трасування невідомо, чи існує шлях між заданими вершинами, то змінній, що вказує на його існування, присвоюється значення true, яке потім змінюється на false у випадку відсутності маршруту. Також на етапі створення нового об’єкта класу Tracer ініціалізуються два стеки, призначені для зберігання позицій на карті, що розглядаються на поточному та наступному кроках. Таке розбиття використовується для спрощення реалізації покрокового режиму.

3.3.2 Ініціалізація об’єкта класу Tracer

Після створення нового засобу пошуку маршруту необхідно виконати його ініціалізацію – задання інформації про карту. Це здійснюється за допомогою функції Initialize. Ця функція приймає своїм

аргументом адресу у пам’яті масиву з даними про карту. Вказівнику, що є властивістю об’єкта класу Tracer, присвоюється значення адреси переданого масиву.

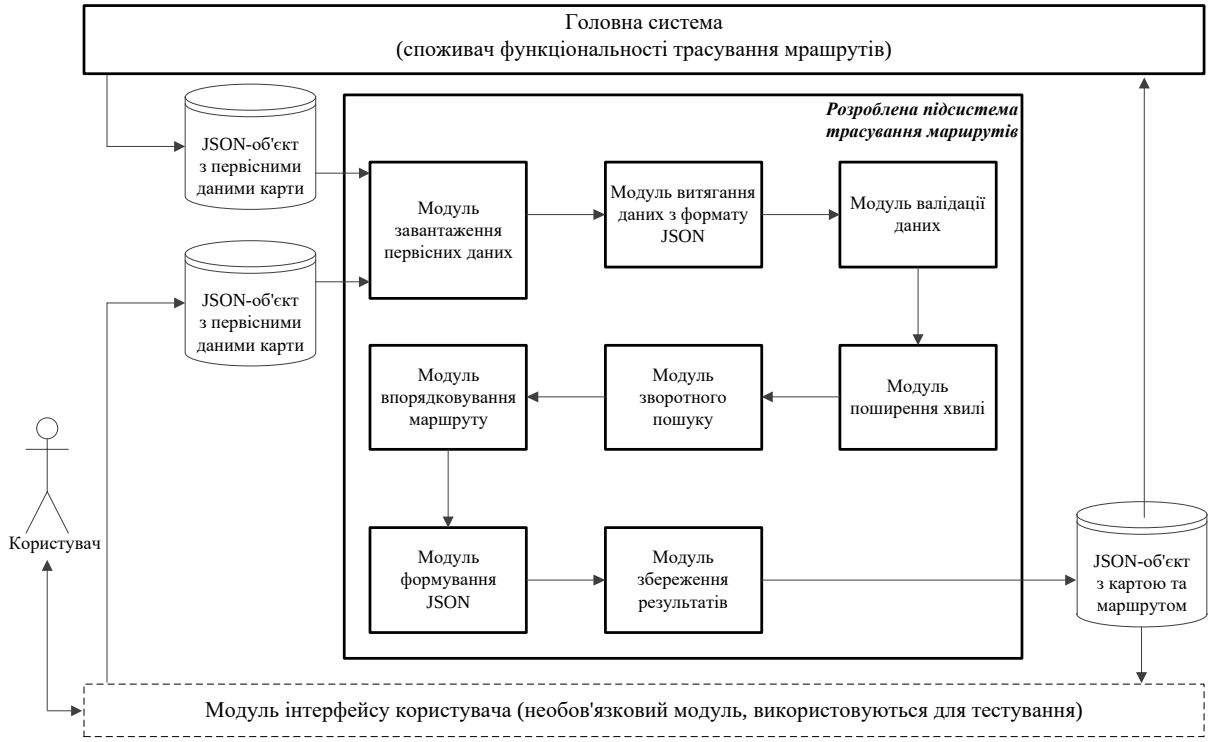

Рисунок 3.4 – Структура головного модуля системи, що розробляється

3.3.3 Реалізація заповнення карти градієнтом

Одразу після ініціалізації масиву виникає необхідність виконати прямий хід хвильового алгоритму. Цим займається функція Fill. Якщо маршрут на даному кроці існує, функція продовжує своє виконання. Відбувається збільшення лічильника кроків. Оскільки другий стек використовується для поточного кроку, а перший – для наступного, дані з першого стеку переносяться до другого. Потім дані починають по черзі вилучатись зі стеку і відбувається перевірка околу відповідних точок за допомогою функції Explore. Нові точки з околу, знайдені цією функцією, додаються до першого стеку.

Коли будуть переглянуті всі точки на поточному кроці, якщо обрано покроковий режим, створюється подія OnNewStep. Після її обробки починається нова ітерація циклу. Якщо на певній ітерації хвиля досягла фінішної точки, генерується подія OnError з відповідним кодом, а функція Fill повертає значення true.

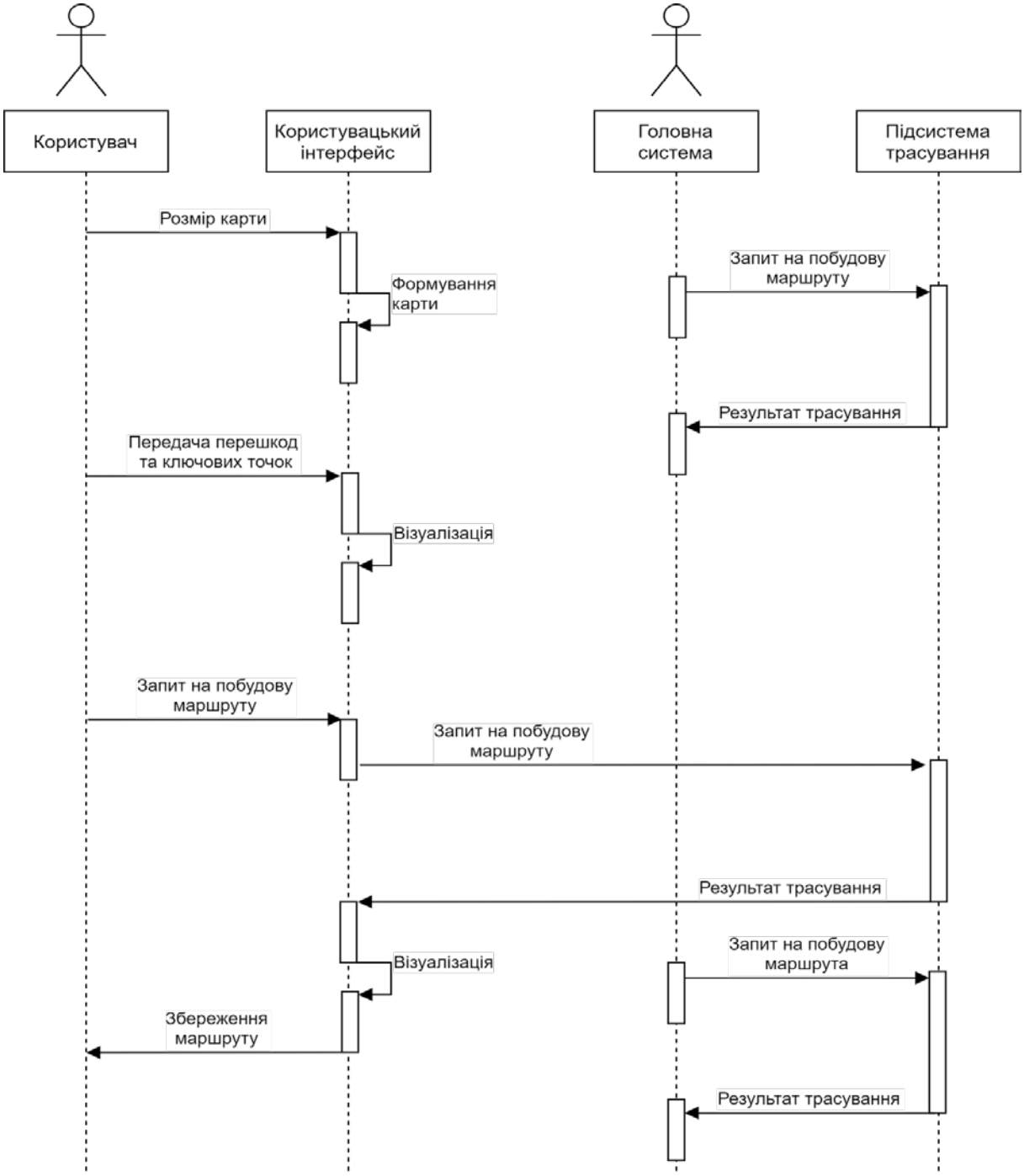

Рисунок 3.5 – Діаграма послідовності функціонування системи трасування

3.3.4 Розробка функції дослідження сусідніх клітин

Під час побудови фронту хвилі необхідно перевіряти окіл розглядуваних точок та встановлювати в них відповідне значення ваги. Для цього призначена функція Explore. Якщо обрано трасування з поширенням хвилі у восьми напрямках, спочатку перевіряються точки, що є сусідніми по діагоналі. Для точки з координатами (cx;cy) це такі точки: (cx-1;cy-1), (cx-1;cy+1), (cx+1;cy+1) та (cx+1;cy-1). Незалежно від обраного режиму трасування перевіряються точки, що знаходяться поряд по вертикалі та горизонталі: (cx-1;cy), (cx;cy-1), (cx;cy+1) та (cx+1;cy). Наведені вище точки по черзі перевіряються за допомогою функції Mark.

3.3.5 Програмна реалізація функції перевірки значення точки карти

При поширенні хвилі необхідно перевіряти сусідні точки на можливість проходу хвилі через них та встановлювати їм відповідну вагу. Для цього призначена функція Mark. Якщо задана точка не є фінішною, але також не є непрохідною або вже пройденою, тобто на відповідну позицію в масиві записаний «0», до цієї точки може поширитись хвиля. При цьому нуль замінюється на поточне значення лічильника кроків. Координати цієї точки заносяться до стеку щоб бути розглянутими на наступному кроці. Оскільки для наступного кроку була знайдена принаймні одна точка для дослідження, до змінної DoPathExist заноситься значення true.

У всіх інших випадках вважається, що точка відповідає позиції перешкоди, або уже була розглянута, отже функція припиняє роботу.

3.3.6 Визначення функції перевірки точки на належність карті

Функція IsInBounds реалізована для запобігання виникненню виключної ситуації звертання до пам’яті, що знаходиться поза межами масиву. Також можливим є використання механізму перехоплення виключних ситуацій, але перевірка умов перед звертанням до пам’яті спрацьовує швидше, ніж перехоплення виключень.

Для реалізації зворотного ходу алгоритму розпочинається цикл, що продовжується доти, доки поточна розглядувана точка не співпаде з точкою старту. В межах циклу у першу чергу відбувається розгляд сусідніх точок по горизонталі та вертикалі шляхом занесення їх координат до змінних cX та cY. Для цього використовується функція DetectPathPoint, результатом роботи якої є інформація про те, чи є дана точка наступною точкою шляху. Якщо наступну точку знайдено, вона стає розглядуваною позицією і розпочинається нова ітерація циклу. Інакше перевіряється інша точка з околу заданої позиції.

Якщо використовується метод трасування з поширенням хвилі у восьми напрямках і наступну точку маршруту не було знайдено серед сусідів по вертикалі та горизонталі, пошук продовжується серед сусідніх точок по діагоналі.

Коли пошук досягає початкової вершини, робота циклу завершується. Якщо активовано покроковий режим, генерується подія OnNewStep. Потім у будь-якому разі створюється подія OnError, аргументом якої є код успішного завершення процедури пошуку маршруту.

3.3.7 Визначення наступної точки маршруту

Коли наступну точку шляху знайдено, виконується встановлення значення позиції для розгляду на наступній ітерації. У випадку, якщо ця точка не співпадає зі стартовою точкою, на відповідну позицію у масиві заноситься значення 0xFFFE, що відповідає точці маршруту. Потім значення, що повинна мати наступна точка маршруту, зменшується на одиницю і, у випадку обраного покрокового режиму, генерується подія OnNewStep. Оскільки наступну точку маршруту знайдено, функція повертає значення true.

3.3.8 Розробка функцій повідомлення про виникнення подій

У класі Tracer передбачено два види подій. Одна з них виникає при завершенні алгоритмом чергового кроку, а інша – при виникненні помилки

в процесі трасування. Функція OnNewStep спочатку перевіряє наявність об’єктів, підписаних на виникнення подій і, якщо такі об’єкти існують, генерує нову подію. Після цього функція змінює значення властивості Waiting статичного класу Data. Якщо покроковий режим все ще є обраним, функція знаходиться у режимі очікування, доки підписаний об’єкт не змінить значення властивості Waiting, що означатиме перехід до нового кроку.

Функція OnError приймає у якості вхідного параметра код помилки, про яку необхідно сповістити користувача. Так само, як і у функції сповіщення про завершення кроку алгоритму, спочатку відбувається перевірка наявності підписаних об’єктів. Потім відповідному аргументу події присвоюється значення коду помилки та відбувається створення цієї події.

3.4 Розробка функціональності головного вікна програми

3.4.1 Визначення властивостей класу MainForm

Властивість MapModified містить значення true, якщо поточна карта була завантажена з файлу та користувачем в неї були внесені зміни. Також це значення робить активними кнопку та пункт меню, що дозволяють зберегти зміни до відкритого файлу.

Властивість PathFound містить значення true коли в результаті трасування між двома заданими точками було знайдено шлях. У такому випадку також стає активним пункт меню, що відповідає за збереження знайденого маршруту до файлу.

Властивість MapSafeFileName містить коротке ім’я поточного файлу без зазначення повного шляху. Зміна цієї властивості спричиняє зміну заголовку головного вікна на такий, що містить ім’я файлу та назву програми.

3.4.2 Реалізація конструкторів головного вікна

Конструктор головного вікна за замовчуванням не має вхідних параметрів та виконує лише функцію ініціалізації компонентів. На початку його роботи також проводиться виклик функції ініціалізації компонентів. Потім повне ім’я файлу зберігається у відповідній змінній. Також виконується знаходження короткого імені файлу шляхом відтинання початкової частини, що містить повне ім’я каталогу, в якому міститься відкритий файл. Змінній FileOpened присвоюється значення true, що означає, що на даний момент ведеться робота з картою, відкритою з файлу.

3.4.3 Реалізація обробників подій

3.4.3.1 Обробка події завантаження головного вікна

Дана функція викликається після спрацювання конструктора головного вікна. Змінним, призначеним для зберігання розмірів тієї області карти, що виводиться на екран, присвоюються значення ширини та висоти відповідного компоненту PictureBox. Крім того, створюється нове растрове зображення, розміри якого співпадають з розмірами компонента PictureBox. Також створюється пов’язаний з зображенням об’єкт класу Graphics, котрий використовується для редагування зображення шляхом використання стандартних функцій рисування графічних примітивів. Створене зображення присвоюється властивості Image компонента PictureBox. Потім виконується зазначення обробника події MouseWheel головного вікна програми. Це дозволяє використовувати колесо миші для переміщення активної області карти, а також для зміни масштабу. Якщо файл було відкрито шляхом передачі параметрів командного рядка, викликається функція завантаження карти з файлу.

3.4.3.2 Програмна реалізація обробки подій натискання елементів головного меню та відповідних їм кнопок

При виборі пункту меню «Створити» або натисканні відповідної кнопки здійснюється виклик функції New, з використанням якої здійснюється створення нової карти. Наступною командою є відображення

створеного діалогу та очікування вибору користувачем каталогу, до якого необхідно зберегти файл, а також зазначення імені файлу. Якщо взаємодія користувача з діалогом завершилася натисканням кнопки «Сохранить», функція ShowDialog повертає значення DialogResult.OK. У цьому випадку здійснюється виклик допоміжної функції SaveMap, одним з аргументів якої стає повне ім’я файлу, обране користувачем.

Обробником події натискання на пункт меню, що відповідає за завантаження з файлу знайденого раніше маршруту, є функція miOpenPath_Click. Перед відкриттям файлу з маршрутом програма пропонує користувачеві зберегти зміни у карті, що редагується програмою на даний момент. Це відбувається шляхом виклику функції AskForSave. Потім створюється діалог відкриття файлу OpenFileDialog, а його поле Filter заповнюється розширенням шуканих файлів («*.tpf») та його описом. Діалог відображається на екрані та дозволяє користувачеві обрати бажаний файл. Якщо користувач успішно завершує роботу з діалогом і функція ShowDialog повертає значення DialogResult.OK, викликається функція завантаження карти з файлу OpenMap, одним з параметрів якої передається повне ім’я обраного користувачем файлу. Якщо обраний файл вдається відкрити, у змінних MapFileName та MapSafeFileName відбувається збереження повного та короткого імені файлу відповідно, що одержуються з полів діалогу. Наостанок відбувається виклик функції DisableAll, що робить неактивними всі елементи керування, що стосуються процесів редагування карти та трасування маршруту.

Натискання на пункт меню «Закрыть» призводить до завершення роботи програми шляхом виклику функції Application.Exit.

При натисканні на кнопку «Дальше» відбувається зміна значення поля Waiting статичного класу Data на false. Таким чином, об’єкт класу Tracer припиняє очікування та переходить до наступного кроку.

Елемент керування типу CheckBox, що вмикає або вимикає покроковий режим, також має подію CheckedChanged. Під час її обробки виконується присвоєння змінним StepMode та Waiting класу Data значення поля Checked цього елемента керування. Таким чином, натискання цієї кнопки вмикає покроковий режим, а її відпускання вимикає покроковий режим та вимикає очікування дозволу на початок наступного кроку. Крім того, цей елемент впливає на активність кнопки «Дальше»: вона є активною лише у випадку, якщо розпочато процес трасування у покроковому режимі.

Переміщення по карті та її масштабування за допомогою колеса миші відбуваються шляхом обробки події MouseWheel. Її обробником є функція WheelHandler. Якщо прокручування здійснюється з натиснутою клавішею Ctrl, відбувається зміна масштабу відображення карти. Значення масштабу може змінюватись у межах від 30% до 1000%. Якщо аргумент Delta події прокручування колеса миші є меншим від нуля, масштаб зменшується на 10%, інакше – збільшується на 10%. Оскільки при зміні масштабу змінюється розмір клітини на карті, необхідно викликати процедуру створення нового курсора. Крім того, необхідно виконати процедуру перерисування активної області карти. Це здійснюється за допомогою функції ImageChanged.

Якщо під час прокрутки колеса миші є натиснутою клавіша Shift, реалізується переміщення активної області карти по горизонталі. Створюється змінна max, у яку заноситься максимальне можливе значення горизонтальної смуги прокручування. Потім обчислюється нове значення позиції повзунка на смузі прокручування шляхом додавання або віднімання (залежно від напрямку прокручування колеса миші) до поточного значення позиції повзунка, числа обчисленого за формулою 1+max/30. Якщо одержане число є меншим від нуля, смуга прокручування зсувається на нульову позицію. Інакше, якщо обчислене значення є більшим, ніж різниця максимального значення і розміру повзунка,

повзунок зсувається на максимальну позицію. Якщо ж нове значення знаходиться у припустимому діапазоні, повзунок зміщується до цієї позиції. У цьому випадку перерисування карти викликати непотрібно, оскільки функція ImageChanged є підписаною на події ValueChanged обох смуг прокручування.

Якщо при прокручуванні колеса миші не є натисненою жодна клавіша, здійснюється зміщення активної області карти вздовж вертикальної осі. Переміщення відбувається аналогічно тому, як це здійснюється при горизонтальному переміщені, єдиною відмінністю є робота з вертикальною смугою прокручування замість горизонтальної.

Аргумент події ErrorCode містить у собі код помилки, що виникла під час трасування маршруту. Якщо ця змінна має значення «0», алгоритм успішно завершив свою роботу і шлях між заданими точками знайдено. У такому випадку змінюється доступність необхідних елементів керування. Також значення властивості PathFound змінюється на true. Потім програма виводить на екран повідомлення про успішний пошук шляху та пропонує виконати трасування тієї ж карти іншим методом. Якщо користувач погоджується, програма відкриває тимчасовий файл з картою, що була збережена перед початком трасування. Інакше програма закриває доступ до елементів керування кроками трасування та до елементу pbField.

Якщо код помилки має значення «1», це свідчить про те, що шляху між заданими точками не існує. Програма викликає перерисування активної області карти та робить активними необхідні елементи керування. Потім програма виводить на екран повідомлення про невдалий пошук шляху та пропонує виконати трасування тієї ж карти іншим методом. Реакція на рішення користувача є такою ж, як і під час обробки помилки з кодом «0».

Якщо код помилки має значення «2», прямий хід хвильового алгоритму успішно завершено. У цьому випадку примусово вмикається

покроковий режим. Таким способом програма дає користувачеві можливість переглянути результат поширення фронту хвилі.

3.4.4 Програмна реалізація процедур відображення карти

Якщо активна область карти та її властивості були змінені, викликається функція ImageChanged. На початку її роботи виконується обчислення розміру всієї карти у пікселях. Для цього кількість клітин у відповідному вимірі множиться на розмір однієї клітини, що обчислюється шляхом множення початкового розміру клітини на масштаб карти та додавання до результату одиниці. Потім обчислюються максимальні значення смуг прокручування. Якщо розмір карти є меншим, ніж відповідний розмір елементу pbField, максимальне значення смуги прокручування є нулем. Інакше воно обчислюється шляхом віднімання від розміру карти кількості клітин та додавання розміру повзунка. Також необхідно розрахувати номер клітини, що знаходиться у лівому верхньому куті активної зони карти, по горизонталі і вертикалі – змінні xPos та yPos. Це здійснюється шляхом ділення значення відповідної смуги прокручування на розмір клітини у пікселях. Крім того, потрібно знати величину зсуву у пікселях від лівого верхнього кута цих клітин відносно кута карти – xShift та yShift. Їх знаходження проводиться шляхом обчислення остачі від ділення позиції відповідної смуги прокручування на розмір клітини у пікселях. Також доцільно знати номер останньої клітини, що належить активній області карти: xLast та yLast. Якщо відповідний розмір карти є меншим, ніж розмір pbField, доданий до позиції смуги прокручування, останньою відображається фактично остання клітина. Інакше номер останньої відображуваної клітини обчислюється шляхом ділення номера останнього відображуваного пікселя карти на розмір однієї клітини. Після обчислення усіх необхідних метрик відбувається виклик функції DrawField для рисування поля.

Спочатку до змінних xShift1 та yShift1 заноситься кількість пікселів по горизонталі та вертикалі, що залишилися від клітини у лівому

верхньому куті. До змінної movement заноситься значення розміру клітини та використовується для зазначення відстані між лініями сітки карти. До змінних maxX та maxY записуються клієнтські координати останньої точки відображуваної області карти. Якщо розмір карти менший, ніж розмір pbField, до змінної заноситься розмір карти, інакше – розмір зображення.

3.6 Висновки

В даному розділі кваліфікаційної роботи на підставі наведених у попередніх розділах теоретичних матеріалів виконані наступні етапи практичної розробки кваліфікаційної роботи: розроблено загальну структуру програмної системи, призначеної для трасування маршруту методом фронту хвилі. Вона складається з двох модулів: основного модуля трасування та модуля візуального відображення процесу трасування, а також файлів карти та файлів знайденого маршруту; розроблено інтерфейс користувача, що базується на використанні інтерфейсу розробки графічних додатків Windows Forms середовища розробки Microsoft Visual Studio; виконано розробку та тестування зазначених модулів; виконано розробку та впровадження в програмну систему форматів файлів для зберігання карти та знайденого маршруту.

ВИСНОВКИ

В межах дипломної роботи виконано проектування, розробку, тестування та документування програмної системи трасування маршруту у двовимірному просторі на основі методів фронту хвилі, що поширюється у чотирьох або восьми напрямках. Для досягнення поставленої мети проведено аналіз методів трасування маршрутів, зокрема методів фронту хвилі. На основі цього аналізу було розроблено алгоритми функціонування програмної системи трасування маршруту.

Після цього здійснено наступні етапи практичної розробки програмної системи трасування маршрутів:

– розроблено загальну структуру програмної системи, призначеної для трасування маршруту методом фронту хвилі. Вона складається з двох модулів: основного модуля трасування та модуля візуального відображення процесу трасування, а також файлів карти та файлів знайденого маршруту;

– розроблено інтерфейс користувача, що базується на використанні інтерфейсу розробки графічних додатків Windows Forms середовища розробки Microsoft Visual Studio;

– виконано розробку та тестування зазначених модулів;

– виконано розробку та впровадження в програмну систему форматів файлів для зберігання карти та знайденого маршруту.

В середовищі розробленої програмної системи проведено експерименти для конкретних екземплярів карти з використанням різних методів трасування маршруту, а також порівняння якості роботи цих алгоритмів.

Розроблений модуль трасування маршруту може бути використаним у якості підсистеми пошуку маршруту у створюваних САПР друкованих плат та системах навігації мобільних роботів. Загалом програмна система може використовуватись для демонстрації методів роботи хвильового алгоритму.

ПЕРЕЛІК ПОСИЛАНЬ

1. Amit Patel. Amit’s A Trace Pages. Stanford Theory Group. 2011. URL: http://theory.stanford.edu/~amitp/GameProgramming. (Дата звертання: 20.06.2022).

2. Harabor D. Jump Point Search. 2011. URL: http://harablog.wordpress.com/2011/09/07/jump-point-search. (Дата звертання: 20.06.2022).

3. Шилдт Г. Полный справочник по C#. Издательский дом «Вильямс», 2014. 752 с.

4. Нейгел К. C# 4.0 и платформа .NET 4 для профессионалов. ДМК», 2011. 1440 с.

5. Иванов Д.В. Алгоритмические основы растровой графики. Бином, 2018. 283с.

ДОДАТОК А

Програма розробленої системи

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace Tracer
{
   static class Program
   {
      private void EnableGroupBox(GroupBox gb, bool en)
      {
         if (gb.InvokeRequired)
         {
            TRACEREnable TRACER = new TRACEREnable(EnableGroupBox);
            gb.Invoke(TRACER, new object[] { gb, en });
         }
         else
         {
            gb.Enabled = en;
         }
      }

      private void ChangeCheckBox(CheckBox cb, bool en,bool
ChangeCheck,bool ch)
      {
         if (cb.InvokeRequired)
         {
            DCBChange TRACER = new DCBChange(ChangeCheckBox);
            cb.Invoke(TRACER, new object[] { cb, en, ChangeCheck, ch });
         }
         else
         {
            cb.Enabled = en;
            if (ChangeCheck) cb.Checked = ch;
         }
```

```
    }

    private void ChangeRadioButton(RadioButton rb, bool en, bool
ChangeCheck,
                                   bool ch)
    {
      if (rb.InvokeRequired)
      {
        DRBChange TRACER = new DRBChange(ChangeRadioButton);
        rb.Invoke(TRACER, new object[] { rb, en, ChangeCheck, ch });
      }
      else
      {
        rb.Enabled = en;
        if (ChangeCheck) rb.Checked = ch;
      }
    }

    private void btnSaveAs_Click(object sender, EventArgs e)
    {
      SaveAs();
    }

    private void SaveAs()
    {
     SaveFileDialog sfDialog = new SaveFileDialog();
     sfDialog.Filter = "Файл карты (*.tmf)|*.tmf";
     if (sfDialog.ShowDialog() == DialogResult.OK)
     {
      SaveMap(sfDialog.FileName, false, false);
      MapFileName = sfDialog.FileName;

MapSafeFileName=MapFileName.Substring(MapFileName.LastIndexOf('\\')+1);
      MapModified = false;
     }
    }

    private void SaveMap(string FileName, bool silent, bool path)
    {
```

```
XmlDocument doc = new XmlDocument();
XmlElement newField = doc.CreateElement("field");
XmlElement newSize = doc.CreateElement("size");
XmlElement newSizeX = doc.CreateElement("x");
newSizeX.InnerText = fX.ToString();
newSize.AppendChild(newSizeX);
XmlElement newSizeY = doc.CreateElement("y");
newSizeY.InnerText = fY.ToString();
newSize.AppendChild(newSizeY);
newField.AppendChild(newSize);
XmlElement newStart = doc.CreateElement("start");
XmlElement newStartX = doc.CreateElement("x");
newStartX.InnerText = StPos.PosX.ToString();
newStart.AppendChild(newStartX);
XmlElement newStartY = doc.CreateElement("y");
newStartY.InnerText = StPos.PosY.ToString();
newStart.AppendChild(newStartY);
newField.AppendChild(newStart);
XmlElement newFinish = doc.CreateElement("finish");
XmlElement newFinishX = doc.CreateElement("x");
newFinishX.InnerText = FinPos.PosX.ToString();
newFinish.AppendChild(newFinishX);
XmlElement newFinishY = doc.CreateElement("y");
newFinishY.InnerText = FinPos.PosY.ToString();
newFinish.AppendChild(newFinishY);
newField.AppendChild(newFinish);
XmlElement newBarrier = doc.CreateElement("barrier");
XmlElement newPath = null;
if (path) newPath = doc.CreateElement("path");
for (int row = 0; row < fY; row++)
{
   for (int col = 0; col < fX; col++)
   {
      if (arrField[row, col] == -1)
      {
         XmlElement newBarrierEl = doc.CreateElement("el");
         XmlElement newBarrierX = doc.CreateElement("x");
         XmlElement newBarrierY = doc.CreateElement("y");
         newBarrierX.InnerText = col.ToString();
```

```
                newBarrierY.InnerText = row.ToString();
                newBarrierEl.AppendChild(newBarrierX);
                newBarrierEl.AppendChild(newBarrierY);
                newBarrier.AppendChild(newBarrierEl);
            }
            else if (path && arrField[row, col] == 0xFFFE)
            {
                XmlElement newPathEl = doc.CreateElement("pt");
                XmlElement newPathX = doc.CreateElement("x");
                XmlElement newPathY = doc.CreateElement("y");
                newPathX.InnerText = col.ToString();
                newPathY.InnerText = row.ToString();
                newPathEl.AppendChild(newPathX);
                newPathEl.AppendChild(newPathY);
                newPath.AppendChild(newPathEl);
            }
        }
    }
    newField.AppendChild(newBarrier);
    if (path) newField.AppendChild(newPath);
    doc.AppendChild(newField);
    try
    {
        doc.Save(FileName);
    }
    catch (UnauthorizedAccessException)
    {
        if (!silent) MessageBox.Show("Отказано в доступе");
    }
    doc.RemoveAll();
}

private void btnOpen_Click(object sender, EventArgs e)
{
    Open();
}

private void Open()
{
```

```
    AskForSave();
    OpenFileDialog ofDialog = new OpenFileDialog();
    ofDialog.Filter = "Файл карты (*.tmf)|*.tmf";
    if (ofDialog.ShowDialog() == DialogResult.OK)
    {
        if (OpenMap(ofDialog.FileName, false))
        {
            MapFileName = ofDialog.FileName;
            MapSafeFileName = ofDialog.SafeFileName;
            FileOpened = true;
        }
    }
}

private bool OpenMap(string FileName, bool path)
{
    int x;
    int y;
    XmlDocument doc = new XmlDocument();
    try
    {
        doc.Load(FileName);
        XmlNode xn = doc["field"];
        xn = xn["size"];
        int.TryParse(xn["x"].InnerText, out Data.Width);
        int.TryParse(xn["y"].InnerText, out Data.Height);
        CreateField();
        xn = doc["field"];
        xn = xn["start"];
        int.TryParse(xn["x"].InnerText, out StPos.PosX);
        int.TryParse(xn["y"].InnerText, out StPos.PosY);
        arrField[StPos.PosY, StPos.PosX] = 1;
        DrawCell(StPos.PosX, StPos.PosY, Color.Green);
        xn = doc["field"];
        xn = xn["finish"];
        int.TryParse(xn["x"].InnerText, out FinPos.PosX);
        int.TryParse(xn["y"].InnerText, out FinPos.PosY);
        arrField[FinPos.PosY, FinPos.PosX] = 0xFFFF;
        DrawCell(FinPos.PosX, FinPos.PosY, Color.Red);
```

```
            StartLocated = true;
            FinishLocated = true;
            XmlNodeList xnList = doc.GetElementsByTagName("el");
            foreach (XmlNode node in xnList)
            {
               int.TryParse(node["x"].InnerText, out x);
               int.TryParse(node["y"].InnerText, out y);
               arrField[y, x] = -1;
               DrawCell(x, y, Color.Orange);
            }
            if (path)
            {
               xnList = doc.GetElementsByTagName("pt");
               foreach (XmlNode node in xnList)
               {
                  int.TryParse(node["x"].InnerText, out x);
                  int.TryParse(node["y"].InnerText, out y);
                  arrField[y, x] = 0xFFFE;
                  DrawCell(x, y, Color.PaleGoldenrod);
               }
            }
            else PathFound = false;
            ChangeRadioButton(rbEditBarrier, true, true, true);
            EditMode = 2;
            MapModified = false;
         }
         catch (Exception) { MessageBox.Show("Не удалось открыть карту");
                                    return false; };
         return true;
      }

      private void ChangeCursor()
      {
         if (pbField.InvokeRequired)
         {
            GeneralInvokeDelegate                                    TRACER=new
GeneralInvokeDelegate(ChangeCursor);
            pbField.Invoke(TRACER);
         }
```

```
    else
    pbField.Cursor = CursorMaker.CreateCursor((int)(iCellSize*multiplier),
                                                    EditMode);
}

private void rbEditStart_CheckedChanged(object sender, EventArgs e)
{
    if (((RadioButton)sender).Checked)
    {
        EditMode = 0;
        ChangeCursor();
    }
}

private void rbEditFinish_CheckedChanged(object sender, EventArgs e)
{
    if (((RadioButton)sender).Checked)
    {
        EditMode = 1;
        ChangeCursor();
    }
}

private void rbEditBarrier_CheckedChanged(object sender, EventArgs e)
{
    if (((RadioButton)sender).Checked)
    {
        EditMode = 2;
        ChangeCursor();
    }
}

private void rbEditSpace_CheckedChanged(object sender, EventArgs e)
{
    if (((RadioButton)sender).Checked)
    {
        EditMode = 3;
        ChangeCursor();
    }
```

```
    }

    private void btnSave_Click(object sender, EventArgs e)
    {
        Save();
    }

    private void Save()
    {
        SaveMap(MapFileName, false, false);
        MapModified = false;
    }

    private bool AskForSave()
    {
        if (MapModified)
        {
            switch (MessageBox.Show("Сохранить изменения?", "Карта изменена",
                                    MessageBoxButtons.YesNoCancel))
            {
                case DialogResult.Yes:
                    if (FileOpened)
                        SaveMap(MapFileName, false, false);
                    else SaveAs();
                    break;
                case DialogResult.Cancel:
                    return true;
                default:
                    break;
            };
        }
        return false;
    }

    private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        e.Cancel = AskForSave();
```

```
    if (!e.Cancel)
        System.IO.File.Delete("tempmap.tmf");
}

private void miExit_Click(object sender, EventArgs e)
{
    Application.Exit();
}

private void miNew_Click(object sender, EventArgs e)
{
    New();
}

private void miOpen_Click(object sender, EventArgs e)
{
    Open();
}

private void miSave_Click(object sender, EventArgs e)
{
    Save();
}

private void miSaveAs_Click(object sender, EventArgs e)
{
    SaveAs();
}

private void MainForm_DragDrop(object sender, DragEventArgs e)
{
  string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);
  string FileName = files[0];
  if (btnOpen.Enabled)
  {
    if (FileName.EndsWith(".tmf"))
    {
      MapFileName = FileName;
      MapSafeFileName=FileName.Substring(FileName.LastIndexOf('\\')+1);
```

```
            FileOpened = true;
            OpenMap(FileName, false);
          }
        }
      }

      private void MainForm_DragEnter(object sender, DragEventArgs e)
      {
        e.Effect = DragDropEffects.Copy;
      }

      private void miSavePath_Click(object sender, EventArgs e)
      {
        SaveFileDialog sfDialog = new SaveFileDialog();
        sfDialog.Filter = "Файл маршрута (*.tpf)|*.tpf";
        if (sfDialog.ShowDialog() == DialogResult.OK)
        {
          SaveMap(sfDialog.FileName, false, true);
        }
      }

      private void miOpenPath_Click(object sender, EventArgs e)
      {
        AskForSave();
        OpenFileDialog ofDialog = new OpenFileDialog();
        ofDialog.Filter = "Файл маршрута (*.tpf)|*.tpf";
        if (ofDialog.ShowDialog() == DialogResult.OK)
        {
          if (OpenMap(ofDialog.FileName, true))
          {
            MapFileName = ofDialog.FileName;
            MapSafeFileName = ofDialog.SafeFileName;
            DisableAll();
          }
        }
      }
}
```