**Зміст**

[Вступ 2](#_Toc8636943)

[Розділ 1. Огляд алгоритму роботи програми та інструментарію для візуалізації  
завдання 5](#_Toc8636944)

[1.1. Правила комп’ютерної гри “Комахи” 5](#_Toc8636945)

[1.2. Вибір оптимального алгоритму для створення гри “Комахи” 5](#_Toc8636946)

[1.3. Вибір структури даних для реалізації алгоритму гри “Комахи” 6](#_Toc8636947)

[1.4. Вибір графічної бібліотеки 7](#_Toc8636948)

[1.5. Висновок до першого розділу 8](#_Toc8636949)

[Розділ 2. Проектування візуальної гри “Комахи” 9](#_Toc8636950)

[2.1. Правила гри та функції для реалізації візуальної гри “Комахи” 9](#_Toc8636952)

[2.2. Блок-схема оптимального алгоритму гри “Комахи” 9](#_Toc8636953)

[2.3. Створена діаграма класів 12](#_Toc8636954)

[2.4. Висновок до другого розділу 14](#_Toc8636955)

[Розділ 3. Реалізація візуальної гри “Комахи” 15](#_Toc8636956)

[3.1. Реалізація основних функцій програмного продукту 15](#_Toc8636958)

[3.2. Реалізація користувацького інтерфейсу 18](#_Toc8636959)

[3.3. Реалізація ієрархії класів 21](#_Toc8636960)

[3.4. Реалізація принципів об’єктно-орієнтованого програмування 23](#_Toc8636961)

[3.5. Висновок до третього розділу 23](#_Toc8636962)

[Висновки 25](#_Toc8636963)

[Список інформаційних джерел 26](#_Toc8636964)

**Вступ**

Всі ми грали в ігри з жанром “Tower Defense”, зазвичай вороги виходять з однієї точки, у них є прешкоди які вони не можуть проходити, та

в них є ціль, - дійти до зарані підготовленої точки. Шлях який вони повинні пройти можна написати самому для заготовленої карти, але якщо вона генерується комп’ютером? Для цього можна використати алгоритм пошуку в ширину.

**Мета цієї курсової роботи -** написати програму для візуалізації алгоритму пошуку в ширину.

**Пошук в ширину** - це метод обходу графа, який працює як з орієнтованими, так і не орієнтованими графами.

**Граф** – це сукупність об’єктів зв’язаних між собою. Ці об’єкти називають вершинами (або вузли), зв’язані вони між собою ребрами.

Ребра графа можуть бути напрямлені, або не напрямлені. Наприклад у нас є місто **А** і місто **B –** це наші вершини, ребра у міст можна розглянути як дороги.

Якщо з міста А в місто Б веде дорога, в зворотню сторону якої їхати не можна, то це напрямлений граф, а коли ми можемо по тій самій дорозі їхати і в одну, і в зворотню сторону - ненапрямлений. Якщо дві вершини з’єднані вони називаються суміжними, в інакшому разі – не суміжними.

**Матриця суміжності** – це спосіб подання графу в виді матриці, тобто двовимірниного масиву, де номер колонки та рядка є номерами вершини.

Графи використовуються в різних сферах науки, можна навести декілька прикладів:

1. Файлова система комп'ютера. Ієрархія файлів в багатьох операційних системах має вигляд дерева, яке є окремим випадком графу.
2. Молекули всіх хімічних речовин можна зобразити у вигляді графу, де атоми є вершинами, а зв'язки між ними – ребрами
3. Соціальну мережу також можна подати у вигляді графу, де кожна людина чи соціальна група є вершиною, а зв'язки між ними — ребрами
4. А ось розробка програмного забезпечення та комп'ютерні науки є однією з тих галузей, де графи застосовуються найчастіше. Графи також є зручними для зображення структур даних, блок-схем, потоків даних, схем баз даних та баз знань, скінченних автоматів, схем комп'ютерних мереж та окремих сайтів, схем викликів підпрограм тощо. Також графи широко використовуються в багатьох алгоритмах пошуку та сортування.

**Розділ 1. Огляд алгоритмів роботи програми та інструментарію для візуалізації пошуку в ширину**

Щоб виконати роботу нам потрібно вибрати засіб для розробки програм.

Для написання даної програми було використано Microsoft Visual Studio 2019. Це лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення та низку інших інструментів. Дані продукти дозволяють розробляти як консольні програми, так і ігри та програми з графічним інтерфейсом, у тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-програми, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Windows.

* 1. **Вибір структур даних**

В ході реалізації програми було використано декілька структур даних:

Масив – це впорядкована структура однотипних даних з фіксованою кількістю, які зберігаються послідовно в комірках оперативної пам’яті, мають порядковий номер, який начинається з нуля. Дана структура ефективна для швидкого доступу до елемента по номеру, але не ефективні при додаванні та видаленні елементів.

Зв`язний список - це структура даних, в якій елементи лінійно впорядковані, але порядок визначається не номерами елементів (як в масивах), а вказівниками, що входять у склад елементів списку та вказують на наступний елемент.

Черга - динамічна структура даних, що працює за принципом «перший прийшов — перший пішов» (англ. FIFO — first in, first out). У черги є голова (англ. head) та хвіст (англ. tail). Елемент, що додається до черги, опиняється в її хвості. Елемент, що видаляється з черги, знаходиться в її голові.

* 1. **Вибір графічної бібліотеки**

При виборі графічної бібліотеки я вирішив, що найоптимальніший

варіант - Windows Forms, оскільки для візуалізації пошуку в ширину достатньо інструментарію, що надається цим інтерфейсом програмування додаткі.

Windows Forms - Це інтерфейс программування додатків (API), що відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework. Цей інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за рахунок створення обгортки для існуючого Win32 API в керованому коді. Причому керований код класи, що реалізують API для Windows Forms, не залежать від мови розробки. Тобто програміст однаково може використовувати Windows Forms як при написанні програмного забезпечення на C#, С++.

* 1. **Опис алгоритму**

В основі реалізації завдання лежить алгоритм пошуку в ширину.

По́шук у ширину́ — алгоритм пошуку на графі. 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 та принцип

FIFO (перший прийшов, перший вийшов). 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.

**1.4. Висновок до першого розділу**

В цьому розділі було обрано структури даних (масив, список, черга), графічну бібліотеку (Windows Forms) та описано головний алгоритм для реалізації ((Breadth-first Search).

**Розділ 2. Проектування програми візуалізіції пошуку в ширину**

**2.1. Функції для реалізації**

Наш програмний продукт повинен виконувати свою роботу так:

Програма читає з текстового файлу символьно-графічну версію лабіринту. Ми цей лабіринт перетворюємо в матрицю суміжностей, щоб далі працювати з цим лабіринтом в головному алгоритмі – пошук в ширину. Пошу в ширину відмітить довжину кроку, та взагалі можна чи туди йти. Пісня відмітки нам потрібно пройтися по краям лабіринту та виявити координати всіх виходів.

Щоб намалювати лабіринт графічно, ми повині зберегти історію кроків в чергі, для того щоб відображати його послідовно як це відбувається в BFS.

Спочатку ми малюэмо рамку лабиринту, наші стіни, вони позначені в матриці кроком -1, це означає, що там немає проходу. Ми малюємо кроки послідновно, відмічаючи найкоротший шлях червоним кольором, а всі інші – чорним. Коли ми дісталися виходу ми закінчуємо малювання.

**2.2. Блок схема алгоритму візуалізації пошуку в ширину**

Отже, вся програма складається з створення потрібних змінних, формування матриці для обробки лабіринту в алгоритмі BFS, пошуку найкоротшого шляху та відображення цього графічно.

Загальна блок-схема програмного продукту зображена на малюнку 1.

Початок

Створення потрібних змінних

Створення матриці суміжностей

Обробка матриці в алгоритмі BFS

Пошук всіх виходів з лабіринту

Малювання лабіринту на формах

Кінець

Малюнок 1. Узагальнена блок-схема програмного продукту

**2.3. Висновок до другого розділу**

У другому розділі був узагальнено описаний функціонал, який нам потрібно реалізувати, та зроблена блок-схема до його реалізації.

**Розділ 3. Реалізація візуалізації пошуку в ширину.**

1-й етап:

Початок роботи починається з класу Graph.

Створення зммінних які ми будемо використовувати для роботи з матрицею на малюнку лістинг 1.

//координати гравця, звідки буде почато пошук(старт)

private Point \_player;

//graph[i,j] - index це вершина

//значення це довжина кроку від старту

private int[,] \_matrix;

//координати виходів

private List<Point> \_exits = new List<Point>();

//історія роботи алгоритму пошуку в ширину

//потрібно щоб правильно відмалювати

private Queue<Point> \_BFSHistory = new Queue<Point>();

public Graph(string side, string freeWave, string player, string filePath)

{

CreateAdjacencyMatrix(side, freeWave, player, filePath);

}