Міністерство освіти і науки України

Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Програмування інтелектуальних інформаційних систем

**ЗВІТ**

до лабораторних робіт

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Виконав**  **студент** |  | ІТ-92, Бондаренко Дмитро Сергійович |  |  |
|  |  | (№ групи, прізвище, ім’я, по батькові ) |  |  |
|  |  |  |  |  |
| **Прийняв** |  | ас. Баришич Л.М. |  |  |
|  |  | (посада, прізвище, ім’я, по батькові ) |  |  |

Київ 2021

# **Завдання лабораторної роботи**

Темою цієї роботи є написання алгоритмів пошуку в ширину (breadth-first search), пошуку в глибину (depth-first search) та Uniform-Cost Search із подальшою візуалізацією цих алгоритмів на полі гри а також із часом виконання цих алгоритмів.

# **Опис використаних технологій**

У цій роботі буде використана мова програмування Python, оскільки вона має зручну бібліотеку для відтворення графічного інтерфейсу, а й власне гри, tkinter.

# **Опис програмного коду**

Робота базується на грі, яку ми написали у минулій роботі. У цій лабораторній роботі були додані 2 файли: search.py та searchAgents.py, де ми реалізуєм алгоритми пошуку та їх загальну візуалізацію.

Потрбіно зауважити, що у минулій роботі, ми реалізували можливість введення параметрів через консоль за допомогою агрументів. Для даної роботи ми будемо використовувати такі аргументи:

-l smallMaze -p SearchAgent -a fn=bfs

-l smallMaze -p SearchAgent -a fn=dfs

-l smallMaze -p SearchAgent -a fn=ucs

Де -l – використовується для обирання деякого лабіринту, -p – використовується для того, щоб обрати агента для переміщення Pac-man, -a – додаткові аргументи, у даному випадку використовуються для обирання алгоритму пошуку.

Також були представлені абревіатури назв алгоритмів для більш зручного використання функцій:

bfs = breadthFirstSearch  
dfs = depthFirstSearch  
ucs = uniformCostSearch

Розберемо сам код:

Були додані класи:

* Stack – структура даних, яка представляє стек.
* Queue – структура даних, яка представляє чергу.
* PriorityQueue – структура даних, яка представляє пріорітетну чергу.
* SearchProblem – абстрактний клас, який описує структуру проблеми пошуку.
* SearchAgent - загальний пошуковий агент, який знаходить шлях за допомогою наданого пошуку алгоритм для заданої проблеми пошуку, а потім повертає дії, щоб слідувати цьому шлях.
* PositionSearchProblem - Проблема пошуку визначає простір стану, стартовий стан, перевірку мети, функцію наступника та функцію витрат.

Були додані такі методи:

У класі SearchAgent:

* registerInitialState – це перший випадок, коли агент бачить макет ігрового поля. Тут ми обираємо шлях до мети. На цьому етапі агент повинен обчислити шлях до мети та зберегти його у локальній змінній.
* getAction – повертає наступну дію на шляху, обраному раніше.

У класі PositionSearchProblem:

* getStartState – повертає стартову позицію Pac-man.
* isGoalState – повертає булеве значення, яке характеризує, чи буде досягнута перемога, при переході у деякий стан.
* getSuccessors – повертає можливі наступні стани для Pac-man.

depthFirstSearch – алгоритм пошуку в глибину. Базується на структурі даних стек.

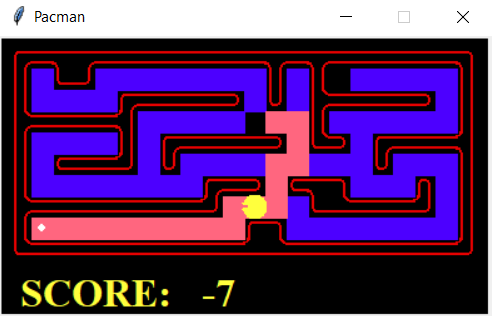
breadthFirstSearch – алгоритм пошуку в ширину. Базується на структурі даних черга.

uniformCostSearch – алгоритм пошуку вузла із найменшою загальною вартістю. Базується на структурі даних пріорітетна черга.

visualiseWay – метод для візуалізації роботи алгоритмів та шляхів.

# **Скріншоти роботи програмного застосунку**

Розберемо пошук в ширину:



Даний алгоритм перебирає всі можливі місця для пересування, а отже і шляхи. Завдяки цьому, цей алгоритм гарантовано поверне найкоротший шлях до заданої точки.

Ми також отримали наступний вивід у консоль (для замірів візуалізація буде вимкнена):

Алгоритм працював: 0:00:00.000511

Шлях:

East; East; South; South; West; South; South; West; West; South; West; West; West; West; West; West; West; West; West;

Шлях із координатів:

(12, 6); (13, 6); (13, 5); (13, 4); (12, 4); (12, 3); (12, 2); (11, 2); (10, 2); (10, 1); (9, 1); (8, 1); (7, 1); (6, 1); (5, 1); (4, 1); (3, 1); (2, 1); (1, 1);

Розберемо пошук в глибину:



Даний алгоритм перебирає лише наступні можливі шляхи, відкидаючи альтернативні варіанти, тобто цей алгоритм повертає лише перший можливий шлях, який приведе до перемоги. Тобто, цей алгоритм не гарантує отримання найкоротшого шляху. По тій самій причині, алгоритм має працювати швидше ніж пошук в ширину.

Ми також отримали наступний вивід у консоль (для замірів візуалізація буде вимкнена):

Алгоритм працював: 0:00:00.000071

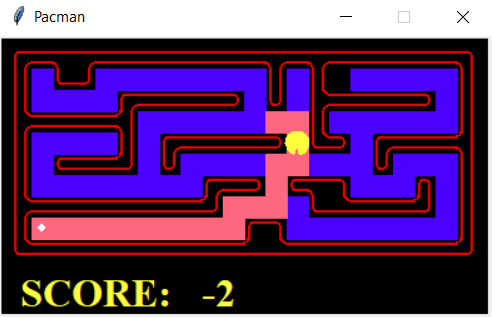
Шлях:

West; West; West; West; West; South; South; South; East; East; North; East; East; East; East; East; East; East; East; South; East; East; North; East; East; South; South; South; West; West; West; West; West; West; West; North; West; West; West; South; West; West; West; West; West; West; West; West; West;

Шлях із координатів:

(10, 6); (9, 6); (8, 6); (7, 6); (6, 6); (6, 5); (6, 4); (6, 3); (7, 3); (8, 3); (8, 4); (9, 4); (10, 4); (11, 4); (12, 4); (13, 4); (14, 4); (15, 4); (16, 4); (16, 3); (17, 3); (18, 3); (18, 4); (19, 4); (20, 4); (20, 3); (20, 2); (20, 1); (19, 1); (18, 1); (17, 1); (16, 1); (15, 1); (14, 1); (13, 1); (13, 2); (12, 2); (11, 2); (10, 2); (10, 1); (9, 1); (8, 1); (7, 1); (6, 1); (5, 1); (4, 1); (3, 1); (2, 1); (1, 1);

Розберемо Uniform-Cost Search:



На відміну від інших алгоритмів, цей алгоритм повертає шлях, який буде найменше затратним. Тобто, цей алгоритм також не гарантує найкоротший шлях. Варто зауважити, що ціна переміщення Pac-man у ішний стан випадкова від 1 до 10.

Вивід у консоль (візуалізація також вимкнена):

Алгоритм працював: 0:00:00.000999

Шлях:

East; East; South; South; West; South; South; West; West; South; West; West; West; West; West; West; West; West; West;

Шлях із координатів:

(12, 6); (13, 6); (13, 5); (13, 4); (12, 4); (12, 3); (12, 2); (11, 2); (10, 2); (10, 1); (9, 1); (8, 1); (7, 1); (6, 1); (5, 1); (4, 1); (3, 1); (2, 1); (1, 1);

Загальна ціна шляху 90.

**Висновок:** У цій роботі ми реалізували 3 алгритми пошуку: пошук в ширину, пошук в глибину та Uniform-Cost Search. Зробили заміри часу виконання кожного алгоритму, та зрозуміли, який із алгоритмів буде давати найкоротший шлях, а який накоротший час виконання. А також реалізували візуалізацію роботи цих алгоритмів завдяки можливостям бібліотеки tkinter.