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 алгоритму A\* а також відповідні для цього евристики (у цій роботі буде використані Мангеттенський відстань, Евклідова відстань та Евклідова квадратична відстань). Також ми реалізуємо проблеми пошуку шляху через всі монети, а також проблему пошуку шляху через кути (4 точки) за допомогою алгоритму А\* з використанням вище зазначенних евристиків. Також ми реалізуємо алгоритм генерації випадкового ігрового поля для роботи вищезазначених 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 минулих роботах. У цій лабораторній роботі були відредактовано 3 файли: search.py, searchAgents.py та layout.py. У перших двох ми напишемо реалізацію проблем пошуку, евристики, а також безпосередньо сам алгоритм А\*. У файлі layout.py ми напишемо алгоритм генерації поля для гри і в залежності від того, який аргумент у консолі ми будемо передавати для поля, такий і лабіринт буде генеруватися, наприклад:

randomMaze – випадкове поле для проблеми пошуку однієї точки.

randomCorners – випадкове поле для проблеми пошуку чотирьох точок.

randomSearch – випадкове поле для проблеми пошуку всіх точок.

Розберемо сам код:

Були додані класи:

* CornersProblem – ця проблема пошуку знаходить шляхи через усі чотири кути поля (4 точки).
* FoodSearchProblem – ця проблема пошуку, пов'язана з пошуком шляху, який збирає всю їжу на полі.

Загалом, ці два класи подібні, оскільки наслідуються від одного класу SearchProblem, який ми розібрали у минулих лабораторних роботах, окрім одного методу – getCostOfActions. Цей метод повертає вартість послідовності дій.

Були додані такі методи:

У фалі layout.py:

# Генерація поля для для проблеми пошуку точки  
def randomMaze():  
 layout\_2d = initRandMaze(20, 20)  
 layout\_2d[18][1] = '.'  
 layout\_2d[1][18] = 'P'  
 return Layout(flattedMaze(layout\_2d))  
  
  
# Генерація поля для проблеми пошуку 4 точок  
def randomCorners():  
 layout\_2d = initRandMaze(20, 20, 7)  
 layout\_2d[18][1] = '.'  
 layout\_2d[1][18] = '.'  
 layout\_2d[1][1] = '.'  
 layout\_2d[18][18] = '.'  
 layout\_2d[10][10] = 'P'  
 return Layout(flattedMaze(layout\_2d))  
  
  
# Генерація поля для проблеми пошуку всієї їжі  
def randomSearch():  
 height = 20  
 width = 20  
 layout\_2d = initRandMaze(height, width, 25)  
  
 num\_of\_rand\_dots = 7  
  
 while num\_of\_rand\_dots != 0:  
 y = int(random.random() \* height)  
 x = int(random.random() \* width)  
 if x != height - 1 and x > 2 and y != width - 1 and y > 2:  
 if layout\_2d[x][y] != '%':  
 layout\_2d[x][y] = '.'  
 num\_of\_rand\_dots -= 1  
 layout\_2d[10][10] = 'P'  
 return Layout(flattedMaze(layout\_2d))  
  
  
# Конвертація двувимірної матриці у список  
def flattedMaze(layout\_2d):  
 layout = []  
 for i in range(len(layout\_2d)):  
 flat\_line = ''  
 for j in range(len(layout\_2d[i])):  
 flat\_line += layout\_2d[i][j]  
 layout.append(flat\_line)  
 return layout  
  
  
# Алгоритм генерації випадкового лабіринту (за випадковим алгоритмом Прима)  
def initRandMaze(width, height, num\_of\_rand\_holes=120)

У файлі search.py:

* aStarSearch – алгоритм пошуку а-зірочка.
* restartGame – метод для перезапуску гри. Цей метод потрібен у тому випадку, коли алгоритм генерації випадкового поля дає збій (відбувається дуже рідко).

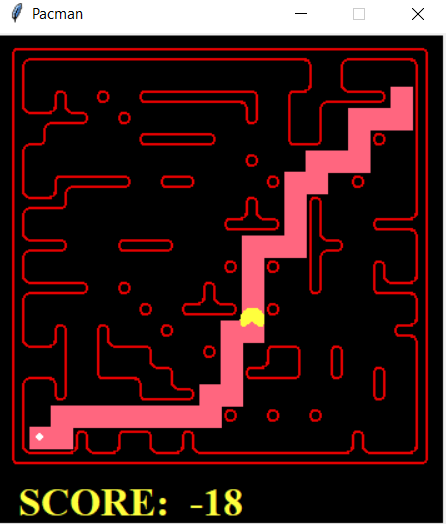
У файлі searchAgents.py:

* manhattan – мангеттенська відстань.
* Euclidean – евклідова відстань.
* euclideanSquared – евклідова квадратична відстань.
* manhattanHeuristic – мангеттенська евристика.
* euclideanHeuristic – евклідова евристика.
* euclideanHeuristicSquared – евклідова квадратична евристика.
* cornersManhattanHeuristic – мангеттенська евристика для проблеми пошуку кутів (4 точок).
* cornersEuclideanHeuristic – евклідова евристика для проблеми пошуку кутів (4 точок).
* cornersEuclideanSquaredHeuristic – евклідова квадратична евристика для проблеми пошуку кутів (4 точок).
* foodHeuristicManhattan – мангеттенська евристика для проблеми пошуку всієї їжі на полі.
* foodHeuristicEuclidean – евклідова евристика для проблеми пошуку всієї їжі на полі.
* foodHeuristicEuclideanSquared – евклідова квадратична евристика для проблеми пошуку всієї їжі на полі.

# **Скріншоти роботи програмного застосунку**

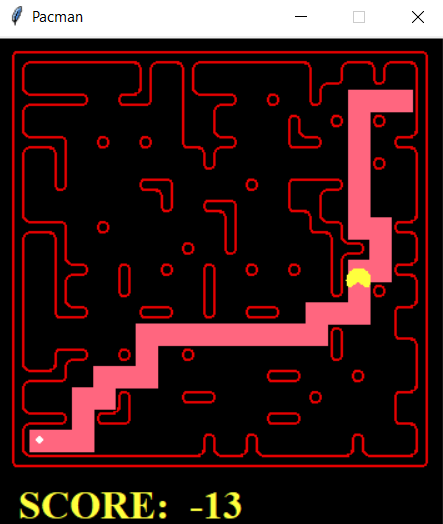
***Розберемо проблему пошуку однієї точки*** за допомогою алгоритму А\* та різних евристиків:

**Мангеттенський шлях:**



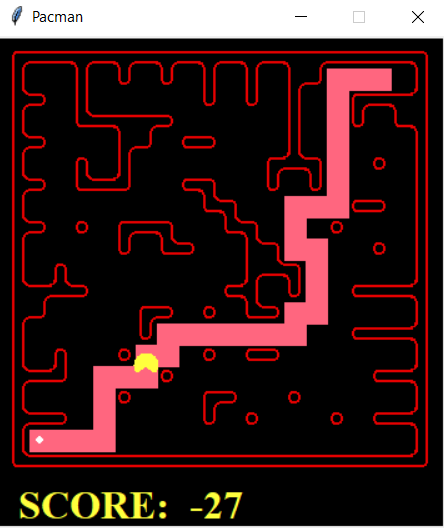
Данна евристика показує суму всіх дистанцій між початком та кінцем. При цьому розширено пошукових вузлів: 212. Це означає, що програма пройшла 212 циклів алгоритма, перед тим, як був знайдений шлях. Алгоритм працював: 0:00:00.003989. Загальна ціна шляху 143.

**Евклідовий шлях:**



Дана евристика показує показує найкоротший шлях між початком та кінцем, при цьому, це буде пряма лінія. Розширено пошукових вузлів: 214. Алгоритм працював: 0:00:00.003989. Загальна ціна шляху 145

**Евклідовий квадратичний шлях:**

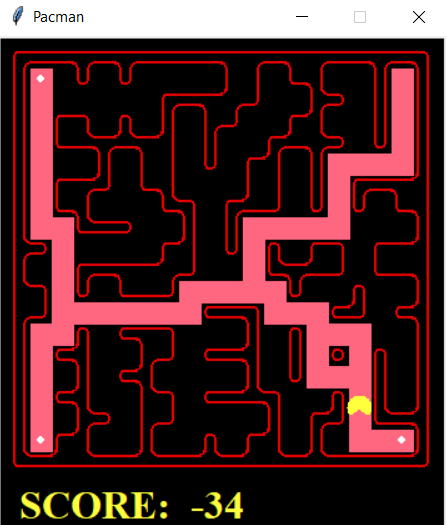


Даний алгоритм позиціонує себе як пришвидшенна версія евклідового шляху, коли ми відкидаємо працеємку операцію піднесення до кореня. Але при цьому, буде відбуватися переоцінка евристики. Тобто, цей алгоритм здеградує до жадібного і ми отримаємо не гарантовано найкоротший і не гарантовано найвигідніший шлях. Але ми отримаємо цей шлях швидко. Розширено пошукових вузлів всього 45, що показує, що ця евристика відпрацьовує найшвидше. Алгоритм працював: 0:00:00.000998. Загальна ціна шляху 186.

Із трьох вищенаведених евристиків Евклідова та Мангеттенська показали себе майже на однаковому рівні. На мою думку, варто обрати Мангеттенську відстань, адже вона вирішує саме ту проблему, яку ми маємо, не дивлячись на те, що Евклідову відстань простіше зрозуміти та реалізувати. Якщо ж треба швидкість відпрацювання алгоритму, а не ефективність, то можна взяти Евклідову квадратичну відстань.

***Розберемо проблему пошуку 4 точок на ігровому полі*:**

**Мангеттенський шлях:**



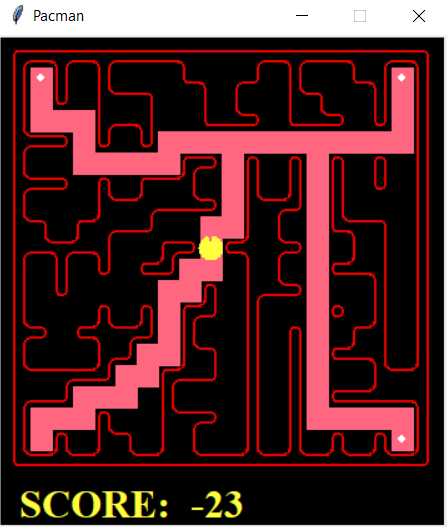
При цьому, ми отримали наступний вивід у консоль:

Алгоритм працював: 0:00:00.116678.

Розширено пошукових вузлів: 1779.

Загальна ціна шляху 98.

**Евклідовий шлях:**



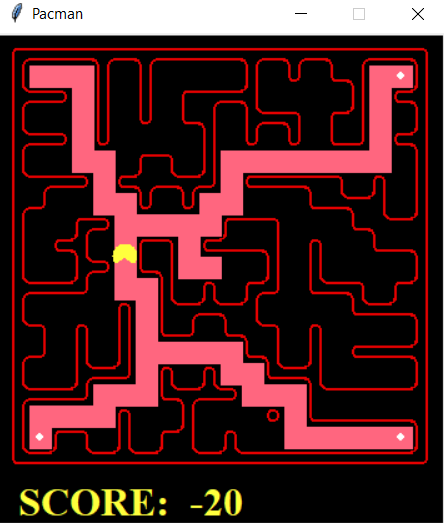
При цьому, ми отримали наступний вивід у консоль:

Алгоритм працював: 0:00:00.129686.

Загальна ціна шляху 104.

Розширено пошукових вузлів: 1863.

**Евклідовий квадратичний шлях:**



При цьому, ми отримали наступний вивід у консоль:

Алгоритм працював: 0:00:00.068843

Загальна ціна шляху 124

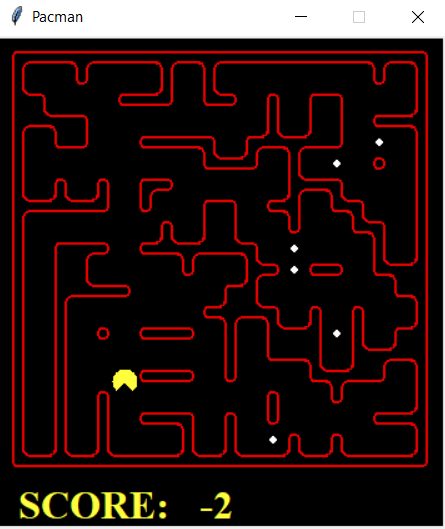
Розширено пошукових вузлів: 1358

Висновок можна зробити ідентичний із попереднім експериментом. Вибір евристики буде грунтуватися на тому, чи потрібна нам швидкість, ефективність або простота реалізації.

***Розберемо проблему пошуку всіх точок на ігровому полі:***

Варто зауважити, що кількість точок була обрана 7, тому, що алгоритм починав працювати досить довго.

**Мангеттенський шлях:**

****

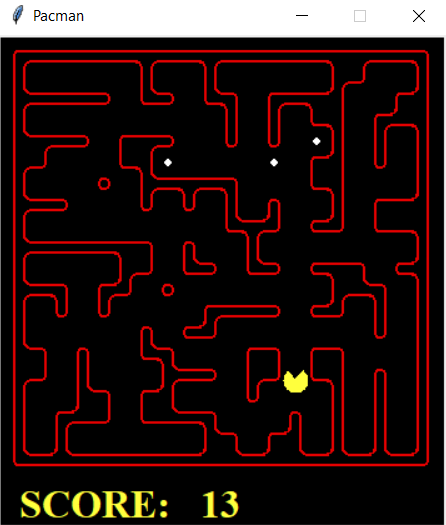
При цьому, ми отримали наступний вивід у консоль:

Алгоритм працював: 0:00:02.498424

Загальна ціна шляху 102

Розширено пошукових вузлів: 6104

**Евклідовий шлях:**



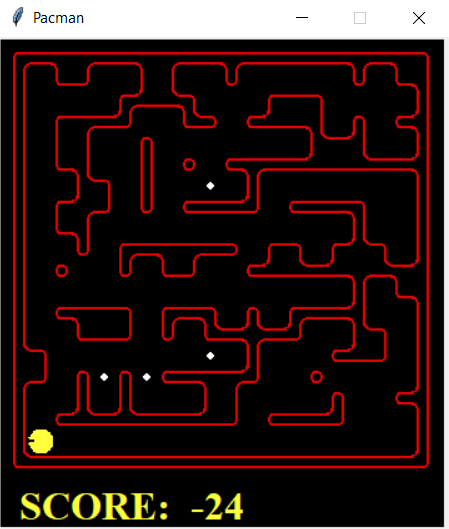
При цьому, ми отримали наступний вивід у консоль:

Алгоритм працював: 0:00:01.693365

Загальна ціна шляху 48

Розширено пошукових вузлів: 4317

**Евклідовий квадратичний шлях:**



При цьому, ми отримали наступний вивід у консоль:

Алгоритм працював: 0:00:00.166582

Загальна ціна шляху 89

Розширено пошукових вузлів: 887

Висновок можна зробити подібний зі попередніми експериментами. У цій проблемі між Мангеттенським та Евклідовим шляхом виникли деякі переваги у часі та кількості циклів, але, більш ймовірно, що це похибка того, як вони будуть працювати на різних ігрових полях. Евклідовий квадратичний шлях веде себе у цій проблемі дуже не оптимізовано. Тобто відпрацьовує він знову швидко, але шлях при цьому дуже довгий та неоптимізований.

**Висновок:** У цій роботі ми реалізували алгоритм пошуку А\* а також 3 евристики: Мангеттенський шлях, Евклідовий шлях а також Евклідовий квадратичний шлях. Перевірили ці евристики на 3 різних проблемах. Із них можна зрозуміти, що між Евклідовим шляхом та Мангеттенським шляхом майже нема різниці. Один простіший у реалізації (Евклідовий) але інший є більш підходящим для вирішення нашої проблеми (Мангеттенський). Евклідовий квадратичний шлях показав себе як найшвидший із усіх, але, оскілки він є жадібним аглоритмом, то шлях не є коротким та оптимальним.