|  |
| --- |
| https://lh6.googleusercontent.com/QcftzNtI05T0Y6fjdSh1Rr2rt8oqZ1IvnLvbn1jLJ7CCyteVir3k-xBLv4SL1wAgWJsRhmmJSR0UW-RP63_GQenE4vVWv05BRoZTsmIcBccVTnfxwmsnNMvjg599x9SqZd8E3dkd |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **«МИРЭА** - **Российский технологический университет»**  **РТУ МИРЭА** |

Институт искусственного интеллекта (ИИИ)

Кафедра проблем управления

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1** | |
| **по дисциплине** | |
| «Агентно-ориентированные системы автономного управления» | |
|  | |
| Выполнил студент группы КРМО-01-23 | Галанин В.А. |
| Принял | Голубов В.В. |

Практические работы выполнены «\_\_» \_\_\_\_\_ 2023 г.

(подпись студента)

«Зачтено» «\_\_» \_\_\_\_\_ 2023 г.

(подпись преподавателя)

Москва 2023

Код программы, написанный на языке Python, представлен в Листинге 1.

В коде используются библиотеки networkx для построения графов, matplotlib для визуализации, numpy для некоторых операций линейной алгебры и time для замера времени выполнения алгоритмов.

В списке agents задаются начальные координаты и целевые точки агентов. В переменных map\_width и map\_height задаются размеры карты. В данном случае карта имеет размер 10 на 10. В словаре agent\_colors задаются цвета для разных агентов. В 65, 66 строках можно выбрать алгоритм A\* или алгоритм Дейкстры. Выполнение алгоритмов замеряется и выводится в консоль.

Листинг 1 — Код программы

import networkx as nx

import matplotlib.pyplot as plt

import numpy as np

import time

*# Создание графа*

G = nx.Graph()

*# Размер карты*

map\_width = 10

map\_height = 10

*# Параметры для алгоритма потенциальных полей*

k\_att = 0.2 *# Коэффициент притяжения к целевой точке*

k\_rep = 1.0 *# Коэффициент отталкивания от других агентов*

rep\_radius = 1 *# Радиус воздействия отталкивания*

*# Список начальных координат агентов и их целевых точек*

agents = [

{"start": (8, 4), "target": (2, 1)},

{"start": (9, 1), "target": (5, 6)},

{"start": (8, 2), "target": (1, 9)},

]

*# Словарь для хранения путей агентов и их цветов*

agent\_paths = {}

*# Задание цветов агентам*

agent\_colors = {

"Agent 1": "olive",

"Agent 2": "lightsalmon",

"Agent 3": "mediumpurple",

}

*# Добавление вершин (узлов) графа для каждой координаты на карте*

for x in range(map\_width):

Продолжение Листинга 1

for y in range(map\_height):

G.add\_node((x, y))

*# Добавление ребра (связи) между соседними вершинами (координатами)*

for x in range(map\_width):

for y in range(map\_height):

*# Добавляем ребра соседних вершин, ограничиваясь манхэттенским расстоянием 1*

for dx in [-1, 0, 1]:

for dy in [-1, 0, 1]:

if abs(dx) + abs(dy) == 1: *# Манхэттенское расстояние 1*

new\_x, new\_y = x + dx, y + dy

if 0 <= new\_x < map\_width and 0 <= new\_y < map\_height:

G.add\_edge((x, y), (new\_x, new\_y))

*# Функция для расчета потенциала отталкивания от других агентов*

def repulsive\_potential(agent\_pos, other\_agent\_pos):

distance = np.linalg.norm(np.array(agent\_pos) - np.array(other\_agent\_pos))

if distance <= rep\_radius:

return 0.5 \* k\_rep \* (1 / distance - 1 / rep\_radius) \*\* 2

else:

return 0

*# Расчет пути для каждого агента с использованием A\* или алгоритма дейкстры*

start\_time = time.time() *# Замер времени*

for agent in agents:

start = agent["start"]

target = agent["target"]

path = nx.astar\_path(G, start, target) *# A\**

*#path = nx.shortest\_path(G, start, target) # Дейкстра*

agent\_paths[start] = path

end\_time = time.time() *# Замер времени*

alg\_1\_time = end\_time - start\_time *# Запись результата времени вычисления*

*# Визуализация путей агентов с использованием A\* или Дийкстры*

for agent\_name, path in zip(agent\_colors.keys(), agent\_paths.values()):

x\_values, y\_values = zip(\*path)

plt.plot(x\_values, y\_values, color=agent\_colors[agent\_name])

*# Расчет пути для каждого агента с использованием алгоритма потенциальных полей*

start\_time = time.time() *# Замер времени*

for agent in agents:

start = agent["start"]

target = agent["target"]

path = [start]

current\_pos = start

max\_iterations = 100 *# Максимальное количество итераций для безопасности*

while current\_pos != target:

Окончание Листинга 1

gradient = np.zeros(2)

for neighbor in G.neighbors(current\_pos):

gradient += k\_att \* (np.array(target) - np.array(current\_pos))

gradient -= repulsive\_potential(current\_pos, neighbor) \* (np.array(current\_pos) - np.array(neighbor))

new\_pos = tuple(np.round(np.array(current\_pos) + gradient).astype(int))

if new\_pos == current\_pos or max\_iterations <= 0:

break

path.append(new\_pos)

current\_pos = new\_pos

max\_iterations -= 1

agent\_paths[start] = path

end\_time = time.time() *# Замер времени*

alg\_2\_time = end\_time - start\_time *# Запись результата времени вычисления*

*# Визуализация путей агентов с использованием алгоритма потенциальных полей*

for agent\_name, path in zip(agent\_colors.keys(), agent\_paths.values()):

x\_values, y\_values = zip(\*path)

plt.plot(x\_values, y\_values, color=agent\_colors[agent\_name], linestyle='--')

*# Визуализация итоговых координат агентов*

for agent in agents:

target = agent["target"]

agent\_name = f"Agent {agents.index(agent) + 1}"

plt.scatter(\*target, color=agent\_colors[agent\_name], marker="x", s=150, label=f'{agent\_name} {target}')

print(f"Время выполнения алгоритма A\* (Дейкстры): {alg\_1\_time} секунд")

print(f"Время выполнения алгоритма потенциальных полей: {alg\_2\_time} секунд")

*# Визуализация легенд, сетки, координат*

plt.legend()

plt.grid(True)

plt.xticks(range(map\_width + 1))

plt.yticks(range(map\_height + 1))

plt.show()

Зададим для агентов начальные точки (8, 4), (9, 1), (8, 2) и целевые точки (2, 1), (5, 6), (1, 9). Выберем алгоритм A\*. Визуализация путей представлена на Рисунке 1. Путь по алгоритму A\* отмечены сплошной линией, а путь по алгоритму потенциальных полей отмечены пунктиром.

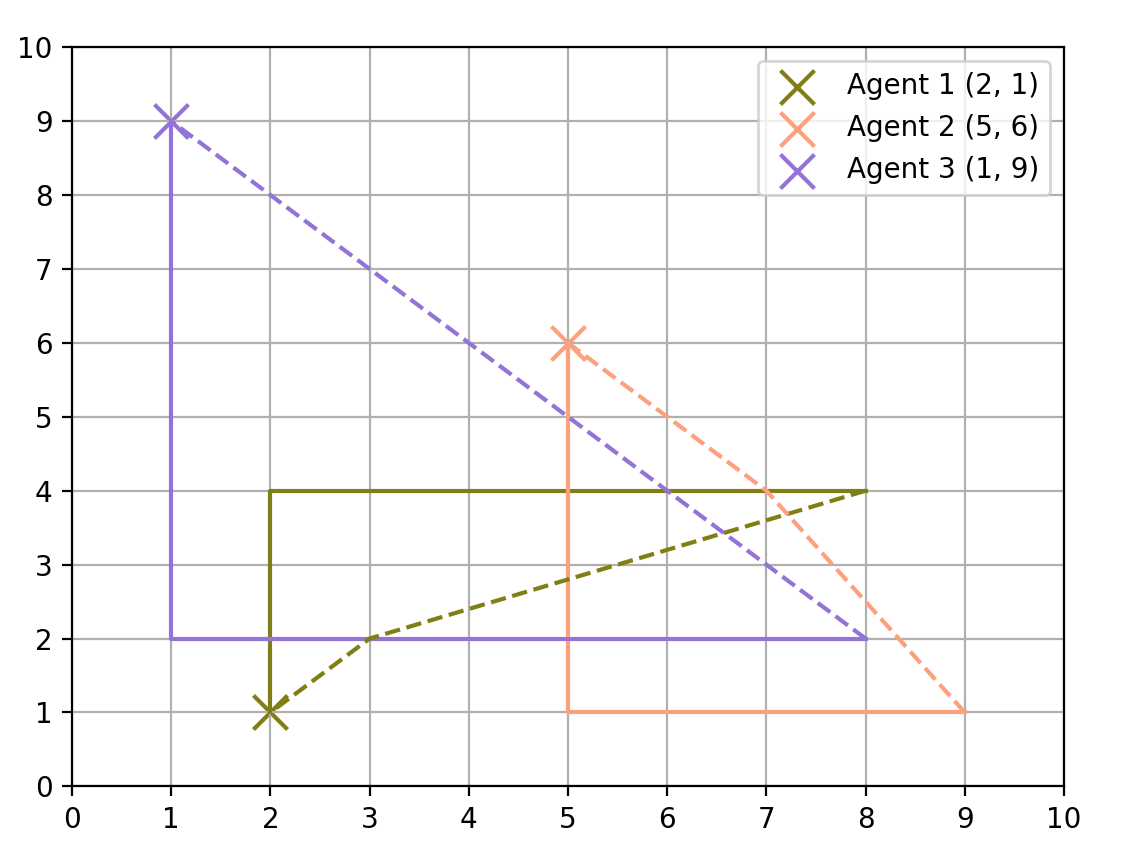


Рисунок 1 — Визуализация путей

Время выполнения алгоритмов показано на Рисунке 2.

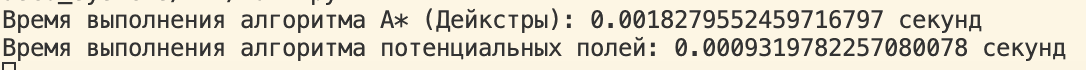


Рисунок 2 — Время выполнения алгоритмов

Как видим, алгоритм потенциальных полей построен оптимальнее, а время его выполнения быстрее.