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.

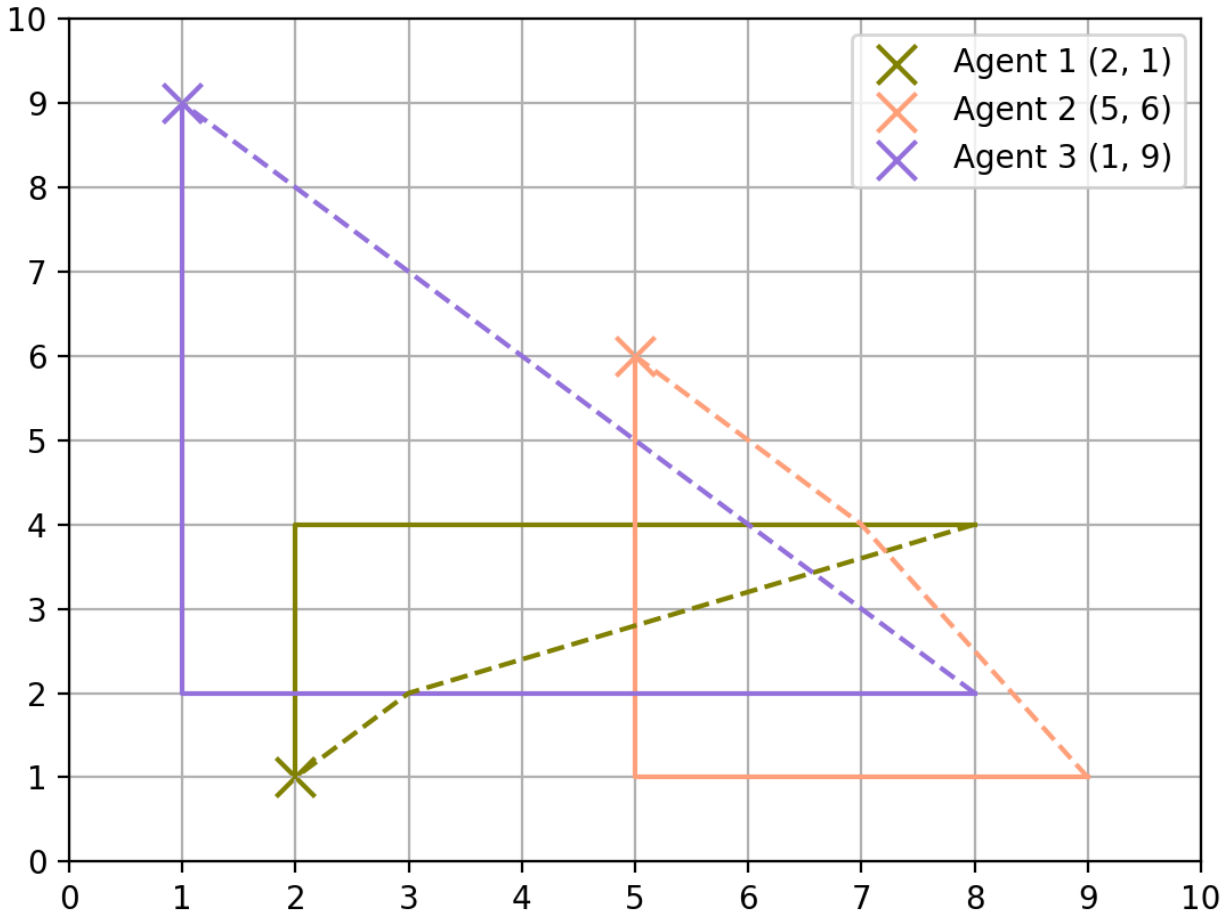

Рисунок 1 — Визуализация путей

Время выполнения алгоритмов показано на Рисунке 2.

```
Время выполнения алгоритма A* (Дейкстры): 0.0018279552459716797 секунд
Время выполнения алгоритма потенциальных полей: 0.0009319782257080078 секунд
```

Рисунок 2 — Время выполнения алгоритмов

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