2180153 - Tumi

2332155 - Tao Yuan

2424161- Shakeel Malagas

A graph of a line graph

Description automatically generatedA graph with red and blue dots

Description automatically generated

Greedy Agent

Random Agent

A blue rectangle with yellow squares

Description automatically generated

## Main.py

from World import World,movelist

from agent import RandomAgent, SmartAgent

from policyGrid import GRID

import matplotlib.pyplot as plt

def debugBoards(agent):

    path = agent.getPath()

    y,x = zip(\*agent.getPath())

    plt.scatter(x,y,color='blue', marker='o')

    for i in range(len(x) - 1):

        plt.annotate(

            '',

            xy=(x[i+1], y[i+1]),

            xytext=(x[i], y[i]),

            arrowprops=dict(arrowstyle='->', color='red', lw=1.5)

        )

    plt.xlabel('X-axis')

    plt.ylabel('Y-axis')

    plt.xlim(0, 6)

    plt.ylim(0, 6)

    plt.gca().invert\_yaxis()

    plt.show()

START = (6,0)

GOAL = (0,0)

obstacleList = [(2,0), (2,1), (2,2), (2,3), (2,4), (2,5)]

world = World((obstacleList), START, GOAL)

grid = GRID()

policy=grid.gridWorld

board = world.getBoard()

print(board)

def run():

    agent1 = RandomAgent(position=START)

    agent2 = SmartAgent(position=START)

    world = World((obstacleList), START, GOAL)

    # ================== AGENT 1 ====================

    print(f"AgentOne initial position: {agent1.position}")

    for \_ in range(50):

        move1 = agent1.chooseMove(movelist)

        agent1.makeMove(move1, world)

    print(f"AgentOne final position: {agent1.position}")

    print(f"AgentOne world final position: {world.getPath()[-1]}")

    # ================== AGENT 2 ====================

    world = World((obstacleList), START, GOAL)

    print(f"Agent Two initial position: {agent2.position}")

    while agent2.currentReward!=20:

        move1 = agent2.chooseMove(movelist,policy)

        agent2.makeMove(move1, world)

    print(f"AgentTwo final position: {agent2.position}")

    return agent1,agent2

def many\_runs(runs:int):

    sum1 = 0

    sum2 = 0

    for i in range(runs):

        agent1,agent2 = run()

        sum1 +=agent1.reward

        sum2 += agent2.reward

    return sum1/runs,sum2/runs

agent1,agent2 = run()

print(agent1.getPath())

print(agent1.reward)

print(agent2.getPath())

print(agent2.reward)

debugBoards(agent1)

debugBoards(agent2)

average1,average2 = many\_runs(20)

numbers = [average1, average2]

labels = ['Random Agent', 'Greedy Agent']

# Plot

plt.bar(labels, numbers, color=['blue', 'orange'])

plt.xlabel('Labels')

plt.ylabel('Values')

plt.title('Reward Bar Graph')

plt.show()

## agent.py

import random

import numpy as np

class Agent:

    def \_\_init\_\_(self, position=(0, 0)):

        self.position = position

        self.reward=0

        self.currentReward=0

        self.path=[position]

    def chooseMove(self,movelist):

        raise NotImplementedError("This method should be overridden by subclasses")

    def makeMove(self, move, world):

        self.position, move\_reward  = world.takeAction(move)

        self.currentReward=move\_reward

        self.reward+=move\_reward

        self.path.append(tuple(self.position))

    def getPath(self):

        return self.path

class RandomAgent(Agent):

    def chooseMove(self,movelist):

        return random.choice(movelist)

class SmartAgent(Agent):

    def chooseMove(self, movelist, policy):

        scores = []

        for move in movelist:

            possible\_move = self.position + np.array(move)

            if 0 <= possible\_move[0] < policy.shape[0] and 0 <= possible\_move[1] < policy.shape[1]:  # Check boundaries

                scores.append((policy[tuple(possible\_move)], move))

            else:

                scores.append((float('-inf'), move))  # Out of bounds moves get the lowest score

        max\_score = max(scores, key=lambda x: x[0])

        best\_moves = [move for score, move in scores if score == max\_score[0]]

        return best\_moves[0]

## World.py

import numpy as np

#constants

UP = (-1,0)

DOWN = (1,0)

LEFT = (0,-1)

RIGHT = (0,1)

movelist=[UP,DOWN,LEFT,RIGHT]

class World:

    def setObstacles(self, obstacles):

        for obstacle in obstacles:

            self.grid[obstacle[0]][obstacle[1]] = 0

    def \_\_init\_\_(self, obstacles, start, goal): #obstacles is a list of tuple coordinates,  start and goal are tuple coordinates

        self.grid = np.zeros((7,7))

        self.grid = self.grid - 1

        self.AgentCoordinates = np.array(start)

        self.PathTravelled = [start]

        self.obstacleList = obstacles

        self.goal = goal

        #adding goal and obstacle to grid for visual

        self.grid[goal[0]][goal[1]] = 20

        self.setObstacles(obstacles)

    def takeAction(self, move):

        #move is integer, 1 2 3 4 (up down left right respectively)

        projectedCoordinates = self.AgentCoordinates + move

        if(projectedCoordinates[0] < 0 or projectedCoordinates[1] < 0 or projectedCoordinates[0] > 6 or projectedCoordinates[1] > 6): #out of bounds check

            self.PathTravelled.append(tuple(self.AgentCoordinates))

            return self.AgentCoordinates, -1

        #hitting obstacle check

        if any(np.array\_equal(projectedCoordinates, obstacle) for obstacle in self.obstacleList):

            self.PathTravelled.append(tuple(self.AgentCoordinates))

            return self.AgentCoordinates, -1

        #goal check

        if np.array\_equal(projectedCoordinates, self.goal):

            self.PathTravelled.append(tuple(projectedCoordinates))

            self.AgentCoordinates = projectedCoordinates

            return self.AgentCoordinates, 20

        #normal move

        self.PathTravelled.append(tuple(projectedCoordinates))

        self.AgentCoordinates = projectedCoordinates

        return self.AgentCoordinates, -1

    def getBoard(self):

        return self.grid

    def getPath(self):

        return self.PathTravelled

# example usage

# start = (6,0)

# goal = (0,0)

# obstacleList = [(2,0), (2,1), (2,2), (2,3), (2,4), (2,5)]

# world = World((obstacleList), start, goal)

# board = world.getBoard()

# print(board)

# position, reward = world.takeAction(UP)

# print(position, reward) #position is returned as np array, if you want coordinates then cast with tuple(position)

# path = world.getPath()

# print(path)

## policyGrid.py

import numpy as np

# up down left right

class GRID:

    def \_\_init\_\_(self) -> None:

        self.gridWorld = np.array([[20,17,16,15,14,13,12],

                                   [18,17,16,15,14,13,12],

                                   [0 ,0 ,0 ,0 ,0 ,0 ,11],

                                   [4 ,5 ,6 ,7 ,8 ,9,10],

                                   [3 ,4 ,5 ,6 ,7 ,8,9],

                                   [2 ,3 ,4 ,5 ,6 ,7,8],

                                   [1 ,2 ,3 ,4 ,5 ,6,7]

                                   ])