SEA400- Assignment 3

|  |  |
| --- | --- |
| Total Mark: | 10 marks |
| Submission file(s): | * Asg3.docx * multiagent.py with your solution |

Please work in **groups** to complete this lab. This lab is worth 9% of the total course grade and will be evaluated through your written submission, as well as the lab demo. During the lab demo, group members are *randomly* selected to explain the submitted solution. Group members absent during the lab demo will lose the demo mark.

Please submit the submission file(s) through Blackboard. Only one person must submit for the group and only the last submission will be marked.

Please cite any resources you use.

**Part I: Multi-Agent Search**

* In this assignment, you will be implementing uninformed and heuristic search algorithms. You will be using code from

[Project 3 | CS 188 Spring 2023 (berkeley.edu)](https://inst.eecs.berkeley.edu/~cs188/sp23/projects/proj3/)

Download and unzip: <https://inst.eecs.berkeley.edu/~cs188/sp23/assets/projects/multiagent.zip>

* Open Anaconda Navigator, activate SEA400 environment and launch VS Code (or your editor). Open above unzipped folder. Use Anaconda terminal to run code, if needed.
* Follow instructions to solve and test questions 1 to 5 from above link (Project 3).

For each question,

* write the algorithm (pseudo-code) you implemented.
* Explain how you implemented each step of the algorithm.

**Part II: Run Tests**

Paste screenshots of the results of the following commands here:

python autograder.py -q q1 –no-graphics

Result:

Determine ghost position

use the manhattan distance to get a distance between the ghost and pacman

Penalize if the host is too close

Calculate food distance using the included function to determine food position

subtract points based on distance to the food.

This moves the agent to want to get as close to food as possible while being as far away from the ghosts as possible.

Pseudocode  
def evaluationFunction

for the state of each ghost

if the manhattandistance is less than 1:

subtract 100 from the score

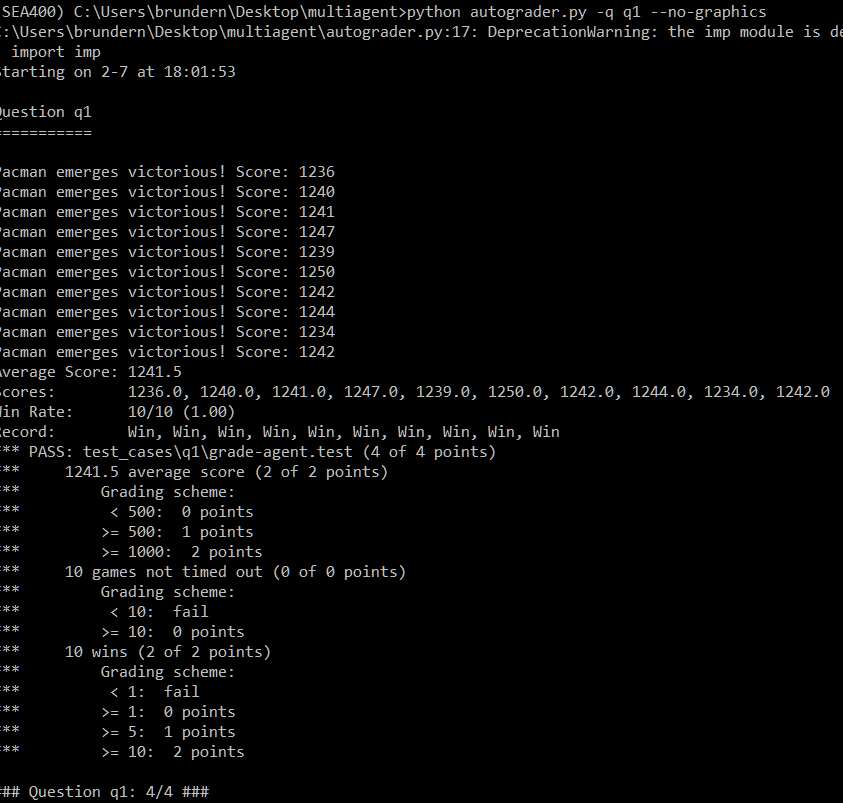
if fooddistance is not empty:

subtract score based on the distance between the agent and the food #this incentivizes the agent to move toward the food

return the final score

A computer screen shot of a program

Description automatically generated



python autograder.py -q q2 –no-graphics

Result:

function minimaxSearch(gameState, depth, maximizingPlayer):

Base case: If depth is 0 or game is in terminal state, return the evaluation function value

return evaluationFunction(gameState)

// Recursive case

if maximizingPlayer:

// Pacman's turn to maximize the score

maxScore = negative infinity

for action in gameState.getLegalActions(0): // Pacman's index

successorState(0, action)

score = minimaxSearch(successorState, depth - 1, false)

maxScore = max(maxScore, score)

return maxScore

else:

// Ghosts' turn to minimize the score

minScore = positive infinity

for i in range(1, gameState.getNumAgents()): // Iterate over ghost agents

for action in gameState.getLegalActions(i):

successorState(i, action)

score = minimaxSearch(successorState, depth - 1, true)

minScore = min(minScore, score)

return minScore

A computer screen shot of a program

Description automatically generated

A screenshot of a computer

Description automatically generated

python autograder.py -q q3 –no-graphics

Result:

function minimaxSearch(gameState, depth, alpha, beta, maximizingPlayer):

Base case: If depth is 0 or game is in terminal state, return the evaluation function value

if maximizingPlayer:

Pacman's turn to maximize the score

maxScore = negative infinity

for action in gameState.getLegalActions(0): // 0 represents Pacman's index

successorState(0, action)

score = minimaxSearch(successorState, depth - 1, alpha, beta, false)

maxScore = max(maxScore, score)

alpha = max(alpha, maxScore)

if beta <= alpha:

break // Beta pruning

return maxScore

else:

// Ghosts' turn to minimize the score

minScore = positive infinity

for i in range(1, gameState.getNumAgents()): // Iterate over ghost agents

for action in gameState.getLegalActions(i):

successorState(i, action)

score = minimaxSearch(successorState, depth - 1, alpha, beta, true)

minScore = min(minScore, score)

beta = min(beta, minScore)

if beta <= alpha:

break // Alpha pruning

return minScore

**A computer screen shot of a program

Description automatically generated**

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

python autograder.py -q q4 –no-graphics

Result:

Base case: If reached the specified depth or the game is over, return the evaluation of the current state

Generate successor states based on legal actions for the current agent

If the current agent is Pacman (maximizing player)

if agentIndex == 0:

Return the maximum score among the successor states

else:

If the current agent is a Ghost (chance player)

Update the agent index and depth for the next iteration

Return the average score of successor states

Calculate scores for each legal action of Pacman using the Expectimax function

Choose the action with the highest score

A screenshot of a computer program

Description automatically generated

python autograder.py -q q5 –no-graphics

Result:

Better score where pacman is near food.

Better score where pacman is far from ghost.

Favor food more than distance from ghost when distance is above a certain limit.

When pacman is in a range of 1 around ghost, state will be very negative.

The lesser food on the map, the better the score.

Same rule for capsules.

When a ghost is scared, states where pacman is near the ghost has significantly better score.

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

python submission\_autograder.py

Result:

A screenshot of a computer

Description automatically generated

**Part III: Group work**

* Complete this declaration by adding your names:

We, Chet, Kasra, Nahaeli, Atem-Ako, declare that the attached assignment is our own work in accordance with the Seneca Academic Policy. We have not copied any part of this assignment, manually or electronically, from any other source including web sites, unless specified as references. We have not distributed our work to other students.

1. Specify what each member has done towards the completion of this work:

|  |  |  |
| --- | --- | --- |
|  | Name | Task(s) |
| 1 | Sthapanavichet Long | All parts |
| 2 | Atem-Ako Eyong Atem | All parts |
| 3 | Nahaeli Brunder | All parts |
| 4 | Kasra Bina | All parts |