Agents and Search

In this homework, you are required to answer some questions (Q1 and Q2), and implement a solution to a described search problem (Q3). To do so, you are going to implement some of the search algorithms that have been covered in the class. You are going to work on an existing game environment, and its written in Python 3 with pygame library. You can obtain the game files from github repository. The program can be started with the following example commands:

- python main.py <RUNNING_MODE> <LEVEL> - The first command line argument is the running mode (examples below), the second argument is the level index (there are 4 levels given to you) being played. Possible running modes are BFS, DFS, ASTAR. There is also a HUMAN mode so you can play the game while thinking about assignment.
  - python main.py BFS 1 - Starts level 1 with BFS agent.
  - python main.py ASTAR 4 - Starts level 4 with A* agent.
  - python main.py HUMAN <LEVEL> - Allows you to play the game yourself in specified level.

Problem 1 - PEAS Description of Agents (20 points)

For each of the following agents, develop a PEAS description of the task environment:

(a) Warehouse robot which carries boxes from one point to another

(b) Home service robot

(c) Activity recognition and anomaly detection software agent in an airport

(d) An agent that classifies tweets

For each of these agent types characterize the environment according to the properties of the environment (observability, dynamism, etc.), and determine the appropriate type of the agent architecture with reasonable arguments.

Problem 2 - Admissible but Inconsistent Heuristics (20 points)

Construct a state space in which the graph search version of A* ends up with a sub-optimal solution using an admissible but inconsistent heuristic function $h(n)$.

Problem 3 - Problem Solving with Search Algorithms (60 points)

In this section, you are required to implement a solution to a game using various search algorithms. In this game, you need to control a character whose aim is collecting all apples **with minimum amount of moves** in a grid world environment. Figure 1 illustrates two example snapshots of world states taken from the game:

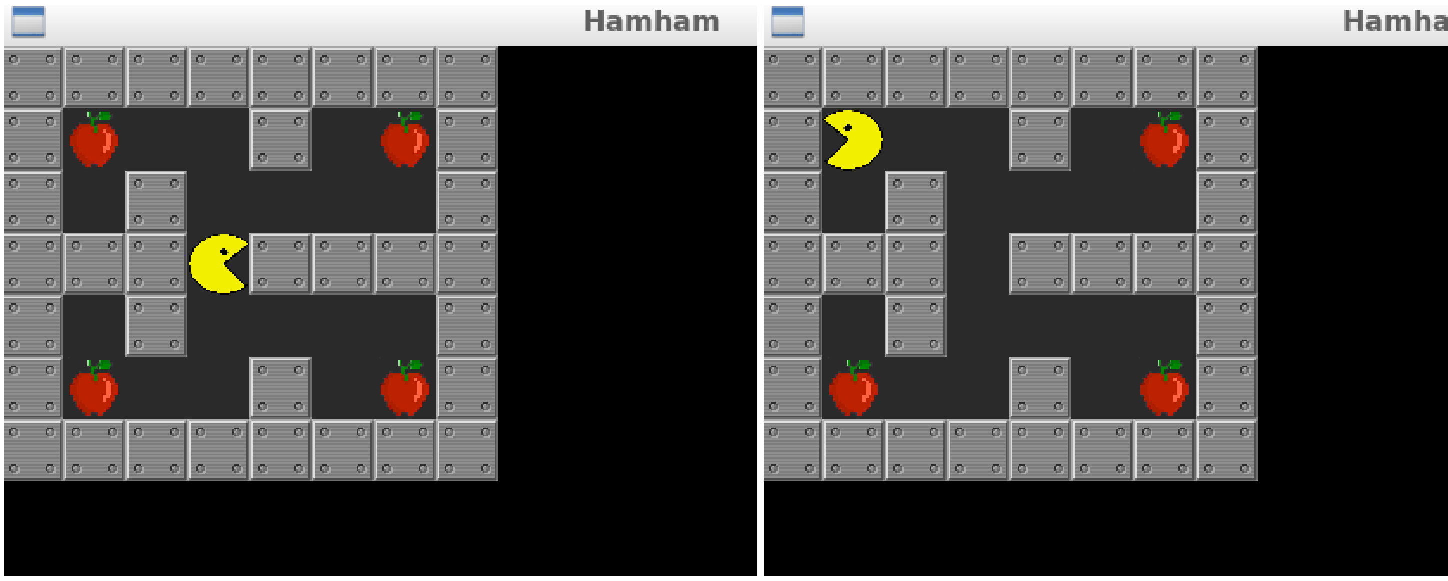

Figure 1: Example snapshots of the described game.

The game has the following rules:

- The character can move in 4 directions: right, up, left, down.
- The character can only move from 1 grid rectangle to another at a single time step. There is not a move such as "go right until hitting a wall".
- The character collects an apple when its on the same position with that apple, and collected apple is removed from the maze.
- A game level is completed when all apples in the level are collected.

You need to:

(a) Formulate this problem in a well-defined form. Explain your state and action representations in detail. If you have used different representations in your implementation, explain them all.

(b) Run appropriate versions of Breadth-First Search (BFS) and Depth-First Search (DFS). Analyze the results in terms of:

- Number of nodes generated
- Number of nodes expanded
- Maximum number of nodes kept in the memory
- Running time

If any of the algorithms does not last, please specify the reason.

(c) Run A* Algorithm with an admissible and consistent heuristic function. Experiment with at least two different tie breaking (deciding which node to expand when f values of the nodes are same) approach, and give a detailed analysis of the results in your report in terms of:

- Number of nodes generated
- Number of nodes expanded
- Maximum number of nodes kept in the memory
- Running time

Some important points about your implementations:

- Required BFS, DFS and A* implementations must be done in their respective files: *bfs_agent.py*, *dfs_agent.py*, *astar_agent.py*.
- You are not allowed to edit any file other than *agent.py*, *bfs_agent.py*, *dfs_agent.py*, *astar_agent.py*. If you would like to use any other programming language, you may invoke your code from the python file and use its result, or write the result to a file and read that file. Other option can be implementing whole assignment with that language you choose..
- You can check explanations in the *agent.py* to understand input and output format of the function *solve*. You can complete the homework by only using agent files. Understanding other game files is not necessary.
- You can import and use any file from standard python library. Any other file inclusion is not allowed.
- You are given 4 sample levels. Level 4 is given to solve with A* only, you do not need to solve it with the uninformed search algorithms. For the other levels, if any of the algorithms can not find the result, please specify the reason.

Important: In Q3, your solution can rely on existing BFS, DFS or A* algorithms. However, you need to explain how the algorithms work in this problem and perform the requested analyses above with sufficient explanations in your report. **Code usage without relevant references will be considered as plagiarism.**

Submission

Submit your homework files through Ninova. Please zip and upload all your files using filename BLG435E_HW_1_STUDENTID.zip. You are going to submit 5 files:

1. (Whether you modified it or not) *agent.py*
2. *bfs_agent.py*
3. *dfs_agent.py*
4. *astart_agent.py*
5. A pdf file containing answers of the first two questions of the homework and required explanations/analyses about the third question of the homework.
6. If you used any other language, include their code files as well.

Note that submitted homeworks may also be tested with different levels of mazes other than those that are given to you. In case of any questions, feel free to send an e-mail to unlut@itu.edu.tr.