BLG 435E – Artificial Intelligence
2019-2020 Fall, Assignment 1 Report

Submission Deadline: 30.10.19, 12:00

Kadir Emre Oto (150140032)

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
   - Observable: partially
   - Deterministic: strategic
   - Episodic: sequential
   - Static: dynamic
   - Discrete: continuous
   - Agents: Multi

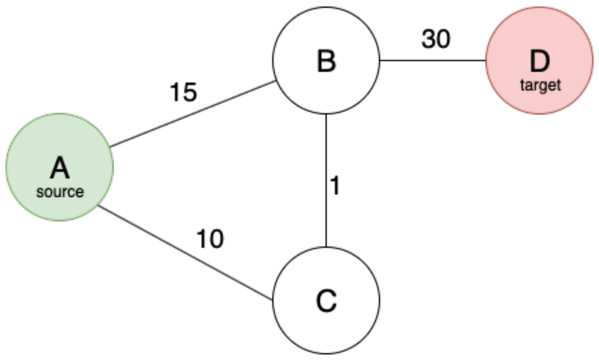
b) Home service robot
   - Observable: partially
   - Deterministic: stochastic
   - Episodic: sequential
   - Static: dynamic
   - Discrete: continuous
   - Agents: Multi

c) Activity recognition and anomaly detection software agent in an airport
   - Observable: partially
   - Deterministic: stochastic
   - Episodic: sequential
   - Static: dynamic
   - Discrete: continuous
   - Agents: Multi

d) An agent that classifies tweets
   - Observable: fully
   - Deterministic: stochastic
   - Episodic: sequential
   - Static: static
   - Discrete: discrete
   - Agents: Multi

For each of these agent types characterize the environment according to 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 $h(n)$.



$$h(A) = 45, \quad h(B) = 0, \quad h(C) = 40, \quad h(D) = 0$$
$$C(A, B) = 15, \quad C(A, C) = 10, \quad C(B, C) = 1, \quad C(B, D) = 30$$

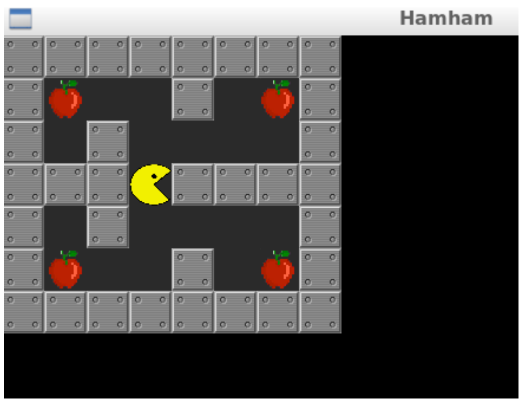
The optimal path from A to D is $A \to C \to B \to D$, but A* finds $A \to B \to D$ because of the inconsistency at the heuristic function. The function is not consistent because:

$$h(C) + C(A, C) > h(A)$$

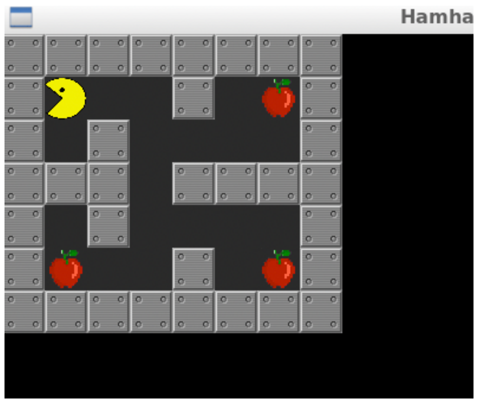
Problem 3 – Problem Solving with Search Algorithms (60 Points)
Originally, this problem can be associated with *Travelling Salesman* problem, which has non-deterministic polynomial (NP) time complexity.

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

Formulation:
State: position of player and positions of uncollected apples. If a state does not have any uncollected apple, it means this state is one of the target states.
Action: At every state, the four neighbor cells of player position should be checked and if a cell is reachable, new state (node) need to be created with new player position and updated apple positions, and should be added to suitable data structure according to the search algorithm (queue for BFS, stack for DFS, priority queue for A*). Mainly all three search algorithms run with same logic except the processing order, the data structures determine which node should be considered at next step.
Heuristic Function $h(n)$: Manhattan distance of farthest uncollected apple. (instead closest uncollected apple can be used but farthest gave better results). This function is used in A* algorithm to select following states heuristically.



Player position: (3, 3)
Apple positions: ((1, 1), (1, 6), (5, 1), (5, 6))



Player position: (1, 1)
Apple positions: ((1, 6), (5, 1), (5, 6))

Figure 1: Example state representations

Implementation:
First of all, I did not use the “Node” class provided in homework template, instead I implemented mine in agent.py file. This class holds two information about the current state: ***position of player*** and ***positions of uncollected apples***. In this way, we do not have to hold the entire map or visited cells. Other attributes of the “*Node*” class are:

- **depth:** denotes the number of steps to reach this state
- **path**: denotes the move sequence to reach this state
- **__hash__()**: hash function to represent a node with an integer
- **f()**: cost function that returns $depth + h(n)$. $h(n)$ is heuristic function, that returns the Manhattan distance to farthest apple.
- **f_cache**: holds the value of f() function to prevent unnecessary calculations.

The logic of all three search algorithms (BFS, DFS, A*) is the same but the state processing order varies because of used data structures (BFS: queue, DFS: stack, A*: priority queue or heap), so it is possible to generalize the main logic with this pseudo-code:

```
states = Queue([initial_state]) or Stack([initial_state]) or PriorityQueue([initial_state])
visited = {hash(initial_state)}
answer = None

while states:
    node = queue.pop()
    if node.is_finish():
        answer = node
        break
    for next_state in node.children:
        if next_state not in visited:
            states.add(next_state)
            visited.add(hash(next_state))
```

Normally, DFS algorithm does not give the optimal solution for this problem, so we need to change the algorithm a little. The first reached target state may not be the optimal answer; therefore, the algorithm should continue to run until no state is left and return the best answer. Because this would be extremely slow and take up memory excessively, some optimizations can be done like that: it the depth of current state is greater than or equal to current answer, then stop branching from here.

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

Level	Label	DFS	BFS	A*
Level 1	elapsed time step	17	17	17
	number of generated node	82	60	37
	number of expanded nodes	75	54	31
	maximum number of nodes kept in memory	4	5	6
	elapsed solve time	0,00087404	0,00063181	0,00068831
Level 2	elapsed time step	27	27	27
	number of generated node	360	266	238
	number of expanded nodes	273	202	176
	maximum number of nodes kept in memory	13	20	20
	elapsed solve time	0,00297523	0,00222826	0,01227927
Level 3	elapsed time step	16	16	16
	number of generated node	864	326	190
	number of expanded nodes	607	232	118
	maximum number of nodes kept in memory	23	48	41
	elapsed solve time	0,0080018	0,00313139	0,00537205
Level 4	elapsed time step	96	96	96
	number of generated node	6988	2977	2148
	number of expanded nodes	6285	2757	1924
	maximum number of nodes kept in memory	27	72	59
	elapsed solve time	0,05107069	0,02648759	0,08330464

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 node 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

Level	Label	A* - 1	A* - 2
Level 1	elapsed time step	17	17
	number of generated node	37	47
	number of expanded nodes	31	41
	maximum number of nodes kept in memory	6	6
	elapsed solve time	0,00068831	0,00071049
Level 2	elapsed time step	27	27
	number of generated node	238	240
	number of expanded nodes	176	180
	maximum number of nodes kept in memory	20	19
	elapsed solve time	0,01227927	0,00303268
Level 3	elapsed time step	16	16
	number of generated node	190	230
	number of expanded nodes	118	139
	maximum number of nodes kept in memory	41	41
	elapsed solve time	0,00537205	0,00375485
Level 4	elapsed time step	96	96
	number of generated node	2148	2662
	number of expanded nodes	1924	2423
	maximum number of nodes kept in memory	59	66
	elapsed solve time	0,08330464	0,03783298

Heuristic Function – 1: Manhattan distance of farthest apple
Heuristic Function – 2: Manhattan distance of closest apple