Artificial Intelligence
Homework 1

Ahmed Burak Gulhan
150160903

Question 1:

1)

Task Environment: Warehouse robot which carries boxes from one point to another

Observable:	Partial
Determinism:	Deterministic
Episodism:	Episodic
Dynamism:	Dynamic (boxes change the environment)
Discretization:	Continuous
Agents:	Multi (if more than one robot)

Performance:	How quickly boxes are moved between two points.
Environment:	A warehouse.
Actuators: forward	Arms of robot (for lifting), steering wheel, accelerator (for and reverse movement), brakes
Sensors:	Camera, GPS (or similar device for location inside warehouse), weight sensors (for lifting boxes)

2)

Task Environment: Home service robot (assuming used for cleaning)

Observable:	Partial
Determinism:	Stochastic
Episodism:	Sequential
Dynamism:	Dynamic
Discretism:	Continuous
Agents:	Multi

Performance:	Percentage of clean area in house and detection/response time for cleaning.
Environment:	A house
Actuators:	Steering wheel, accelerator(forward and reverse movement), cleaning brushes, brakes
Sensors:	Forward camera(detecting obstacles), downward camera(detecting unclean areas and downward drops)

3)

Task Environment:	Activity recognition and anomaly detection software agent in an airport
Observable:	Partial
Determinism:	Stochastic
Episodism:	Sequential
Dynamism:	Static
Discretism:	Discrete
Agents:	Single

Performance:	consistently detecting anomalies without misdetecting non anomalies, F1 score (where where detecting an actual anomaly correctly is a true positive and detecting something as having no anomaly correctly is a true negative)
Environment:	An airport.
Actuators:	Monitor screen (to show if an anomaly is detected), keyboard
Sensors:	Cameras, weight detectors, metal detectors

4)

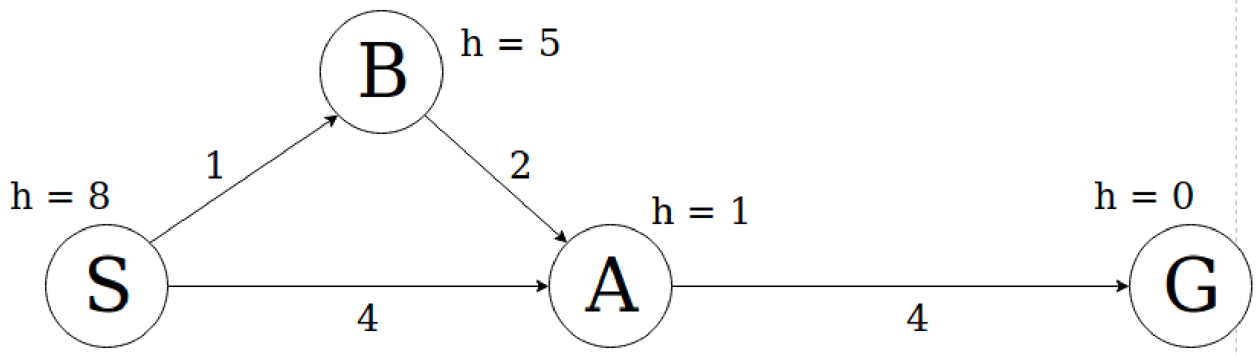
Task Environment:	An agent that classifies tweets
Observable:	Fully
Determinism:	Deterministic
Episodism:	Episodic
Dynamism:	Static
Discretism:	Continuous
Agents:	Single

Performance:	Classification accuracy
Environment:	Twitter
Actuators:	Monitor screen (to show results), keyboard
Sensors:	String arrays (to store tweets)

Question 2:

If h(n) is not consistent then the optimal path is not guaranteed. h(n) is consistent if for every node n, for every successor node n’ due to action a, h(n) <= c(n,a,n’) + h(n’)

An example state space for an admissible but inconsistent heuristic.



Here by using an inconsistent heuristic we get the path S→A→G which has a length of 8. However this is not the optimum path. The optimum path is S→B→A→G which has a length of 7.

Question 3:

a)

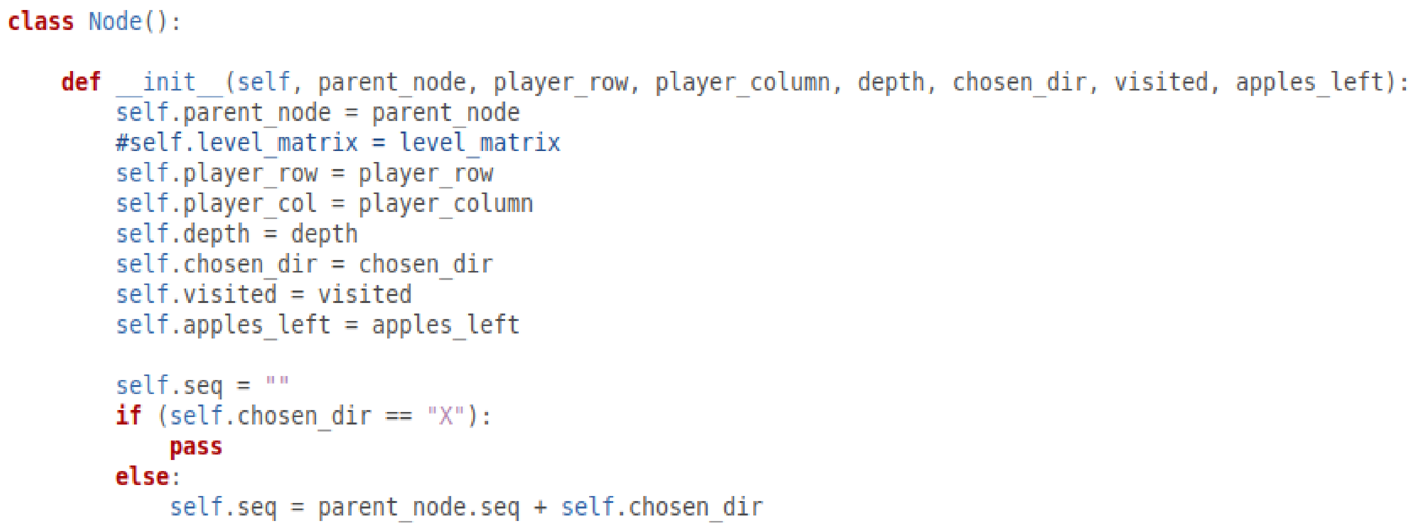
In all 3 solutions (BFS, DFS, A*) the same state and action representations are used except for the A* state where it is slightly different.

Actions:

- move up
- move down
- move left
- move right
- eat fruit

State representation for BFS and DFS:

Each state is represented as a node defined as the following:



```
class Node():

    def __init__(self, parent_node, player_row, player_column, depth, chosen_dir, visited, apples_left):
        self.parent_node = parent_node
        #self.level_matrix = level_matrix
        self.player_row = player_row
        self.player_col = player_column
        self.depth = depth
        self.chosen_dir = chosen_dir
        self.visited = visited
        self.apples_left = apples_left

        self.seq = ""
        if (self.chosen_dir == "X"):
            pass
        else:
            self.seq = parent_node.seq + self.chosen_dir
```

In each state there is a pointer to the parent node, the player’s coordinates, the depth of the node in the search tree, the direction the player had just moved in, the nodes that have been visited since last collecting an apple (or from the starting state), the coordinates of the apples that have not been collected and the sequence of directions that have been taken up until this state.

There is no level matrix in a node, instead the level matrix is accessed from a variable defined outside of the node. The reason for this is that the level matrix does not change between nodes and it would take unnecessary memory space to include the level matrix in each node.

State representation for A*:

Each state is represented as a node defined as the following:

```python
class Node():

    def __init__(self, parent_node, player_row, player_column, depth, chosen_dir, h_value, visited, apples_left):
        self.parent_node = parent_node
        self.player_row = player_row
        self.player_col = player_column
        self.depth = depth
        self.chosen_dir = chosen_dir
        self.h = h_value
        self.visited = visited
        self.apples_left = apples_left

        self.seq = ""
        if (self.chosen_dir == "X"):
            pass
        else:
            self.seq = parent_node.seq + self.chosen_dir

    def __lt__(self, other):
        return self.depth + self.h < other.depth + other.h
```

Here the only difference from the node in BFS and DFS is that there is a heuristic value (h_value) for the current state.

b)
BFS for level 1:
elapsed time step:17
number of generated nodes:66
number of expanded nodes:60
maximum number of nodes kept in memory:6
elapsed solve time:0.0013577938079833984

BFS for level 2:
elapsed time step:27
number of generated nodes:2954
number of expanded nodes:2252
maximum number of nodes kept in memory:475
elapsed solve time:0.05541038513183594

BFS for level 3:
elapsed time step:16
number of generated nodes:50455
number of expanded nodes:27359
maximum number of nodes kept in memory:23096
elapsed solve time:0.9214425086975098

BFS for level 4:
elapsed time step:96
number of generated nodes:98486
number of expanded nodes:88973
maximum number of nodes kept in memory:4282
elapsed solve time:2.440591335296631

```
DFS for level 1:
elapsed time step:22
number of generated nodes:26
number of expanded nodes:22
maximum number of nodes kept in memory:4
elapsed solve time:0.00033664703369140625

DFS for level 2:
elapsed time step:28
number of generated nodes:58
number of expanded nodes:41
maximum number of nodes kept in memory:13
elapsed solve time:0.0006134510040283203

DFS for level 3:
elapsed time step:36
number of generated nodes:73
number of expanded nodes:36
maximum number of nodes kept in memory:36
elapsed solve time:0.0007398128509521484

DFS for level 4:
elapsed time step:168
number of generated nodes:252
number of expanded nodes:215
maximum number of nodes kept in memory:28
elapsed solve time:0.002356767654418945З
```

From these results we can see that BFS finds the most efficient path but takes a longer time to calculate and a larger memory space and that DFS finds in inefficient path but takes a very short time to calculate and a very small memory space.

From a cursory inspection of the results we can see that the time and space complexity of BFS increases in an exponential manner and that the time and space complexity of DFS increases in a seemingly linear way as the search space increases.

We make an interesting observation in BFS level 3. Despite level 3 being much smaller then level 4. In BFS level 3 the amount of nodes stored in memory is 23096 which is over 5 times larger than the amount stored in BFS level 4. This is probably because that level 3 has no inner walls so that the branching factor of each step is very high. We can calculate the branching factor by dividing the number of generated node by the number of expanded nodes. For BFS level 3 we get a branching factor of about 1.844. For the branching factor of BFS level 4 we have 1.1. Therefore we can say that in BFS as the branching factor increases the number of nodes in stored in the memory increases very quickly.

c)
For the A* algorithm I used the following heuristic:
h(n) = real distance of farthest two apples + player’s distance to the closest apple in the calculated farthest pair + number of uncollected apples

This algorithm is admissible since the player will always have to travel at least the maximum distance between the two furthest fruits (real distance of the farthest two fruits) and will also have to travel to one of these fruits to collect it (players distance to the closest apple in the calculated farthest pair). The shortest possible path will be if all other apples are in the path between the player and the closest farthest-apple-pair and the path between the farthest apples. The number of uncollected apples is added to this heuristic so that when the player is traveling to an apple in the farthest-apple-pair it will collect nearby apples on the path further decreasing the number of nodes generated.

Checking the consistency is more difficult, however the results of the A* algorithm when using this heuristic finds the optimum path in every level, so it is probably consistent

To calculate the real distance I used BFS. This is not a problem when calculating the distance between the two farthest points since this value is only recalculated if one of the farthest-apple-pair is eaten. This mean this real distance is only calculated at most f-1 times where f is the number of fruits. This is a low amount of calculations so it does not affect the speed of the algorithm is a noticeable way. However when calculating the player’s distance to the closest apple in the calculated farthest pair using BFS to find the real distance is very costly. You need to apply BFS in every step. This problem can be solved by pre-calculating all distances for every point in the level. Then finding the real path between two points will take only O(1) time. For this homework I did not pre-calculate any points, since it would take extra effort to do this and that the run-time for the given levels is still at an acceptable amount. An other solution is to use the Manhattan distance to calculate the distance for the player’s distance to the closest apple in the calculated farthest pair. This takes much less time but has an increase in the amount of nodes generated. When using this heuristic with Manhattan distance for the player’s distance to the closest apple in the calculated farthest pair there is around a 30% increase in the number of generated nodes, expanded nodes and number of nodes kept in memory, but there is around a 50-100x decrease in run-time. For the submission I used the Manhattan distance, but this can be changed by setting real_dist = True in line 83 in astar_agent.py.

For A* level 4 using Manhattan distance in heuristic (<u>this is what is used by default in the submission</u>):
elapsed time step:96
number of generated nodes:1397
number of expanded nodes:1193
maximum number of nodes kept in memory:144
elapsed solve time:0.18465399742126465

For A* level 4 using real distance (calculated by BFS):
(To use this heuristic, set real_dist = True in line 83 in astar_agent.py.)

```
elapsed time step:96
number of generated nodes:1006
number of expanded nodes:893
maximum number of nodes kept in memory:98
elapsed solve time:1.6732394695281982
```