ASSIGNMENT #1

BLG 435E

Ece Naz Sefercioğlu
150130140

1)

Q 1 - b - Email sorting application based on preferences

Performance: Satisfy user, classify emails correctly

Environment: Web server; Mail provider; User;

Actuators: Matching algorithms; optimization algorithms

Sensors: Text parsing; Keyword search; user feedback

Environment description: Fully observable, episodic, static, continous, Single agent

Since the program will change its pattern to sort emails in accordance to user feedback it is a learning agent.

2)

Q2 - $k(n)$: cost of the cheapest path from n to the goal node.

prove by induction on the number of steps to the goal that $h(n) \leq k(n)$

Base case: If 0 steps between goal node and n n is goal node $\Rightarrow$

$$h(n) = 0 \leq k(n)$$

Induction Step: If n is i steps away from the goal, there must exist some successor n' of n generated by some action a s.t. n' is on the optimal path from n to the goal and n' is $i-1$ steps away from the goal,

Therefore $h(n) \leq c(n, a, n') + h(n')$

But by the induction hypothesis, $h(n') \leq k(n')$ Therefore,

$$h(n) \leq c(n, a, n') + k(n') = k(n)$$

n' is on the optimal path from n to the goal via action a

Consider a search problem where the states are nodes along a path $P = n_0, n_1, \dots n_m$. n_0 is start state, n_m is goal state and there is one action from each state n_i which gives n_{i+1} as a successor with cost 1. The cheapest cost to the goal from a state i is then $k(n_i) = m - i$. Defined heuristic function:

$$h(n_i) = m - 2\lceil i/2 \rceil$$

For all states n_i, $h(n_i) \leq k(i)$ and so h is admissible. However if i is odd, then $h(n_i) = h(n_{i+1}) > 1 + h(n_{i+1})$ Thus h not consistent

As the solution provided: http://reason.cs.uiuc.edu/eyal/classes/f06/cs440/hw/hw6/hw6_sol.pdf

3)

- Only code implementations done on search.py

About references and implementation

```python
def aStarSearch(problem, heuristic=nullHeuristic):
  "Search the node that has the lowest combined cost and heuristic first."
  "*** YOUR CODE HERE ***"
  print "Start:", problem.getStartState()
  print "Is the start a goal?", problem.isGoalState(problem.getStartState())
  print "Start's successors:", problem.getSuccessors(problem.getStartState())
  print problem

  frontier = util.PriorityQueue()
  visited = dict()

  state = problem.getStartState()
  node = {}
  node["parent"] = None
  node["action"] = None
  node["state"] = state
  node["cost"] = 0
  node["eval"] = heuristic(state, problem)
  # A* use f(n) = g(n) + h(n)
  frontier.push(node, node["cost"] + node["eval"])

  while not frontier.isEmpty():
    node = frontier.pop()
    state = node["state"]
    cost = node["cost"]
    v = node["eval"]
    #print state

    if visited.has_key(state):
      continue

    visited[state] = True
    if problem.isGoalState(state) == True:
      break

    for child in problem.getSuccessors(state):
      if not visited.has_key(child[0]):
        sub_node = {}
        sub_node["parent"] = node
        sub_node["state"] = child[0]
        sub_node["action"] = child[1]
        sub_node["cost"] = child[2] + cost
        sub_node["eval"] = heuristic(sub_node["state"], problem)
        frontier.push(sub_node, sub_node["cost"] + node["eval"])

  actions = []
  while node["action"] != None:
    actions.insert(0, node["action"])
    node = node["parent"]

  return actions
```

From the repository of user *weixsong* code on the below link is used for improvements on personally written code. https://github.com/weixsong/pacman/blob/master/search.py

```python
node = {}
node["parent"] = None
node["action"] = None
node["state"] = problem.getStartState()
node["cost"] = 1
```

To ease the node definition by using dictionary data type of python,

```python
visited=dict()
```

Use visited as dictionary.

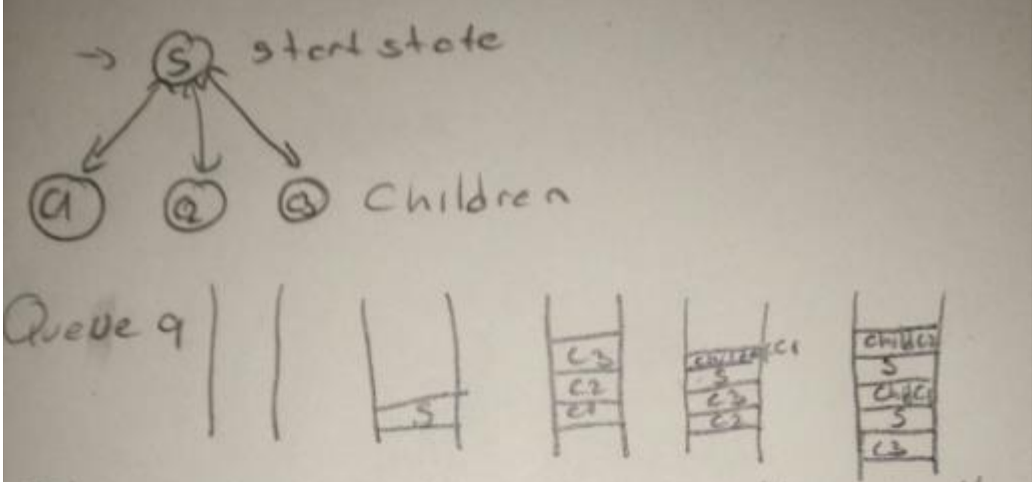
```python
actions = []
while node["action"] != None:
    actions.insert(0, node["action"])
    node = node["parent"]

return actions
```

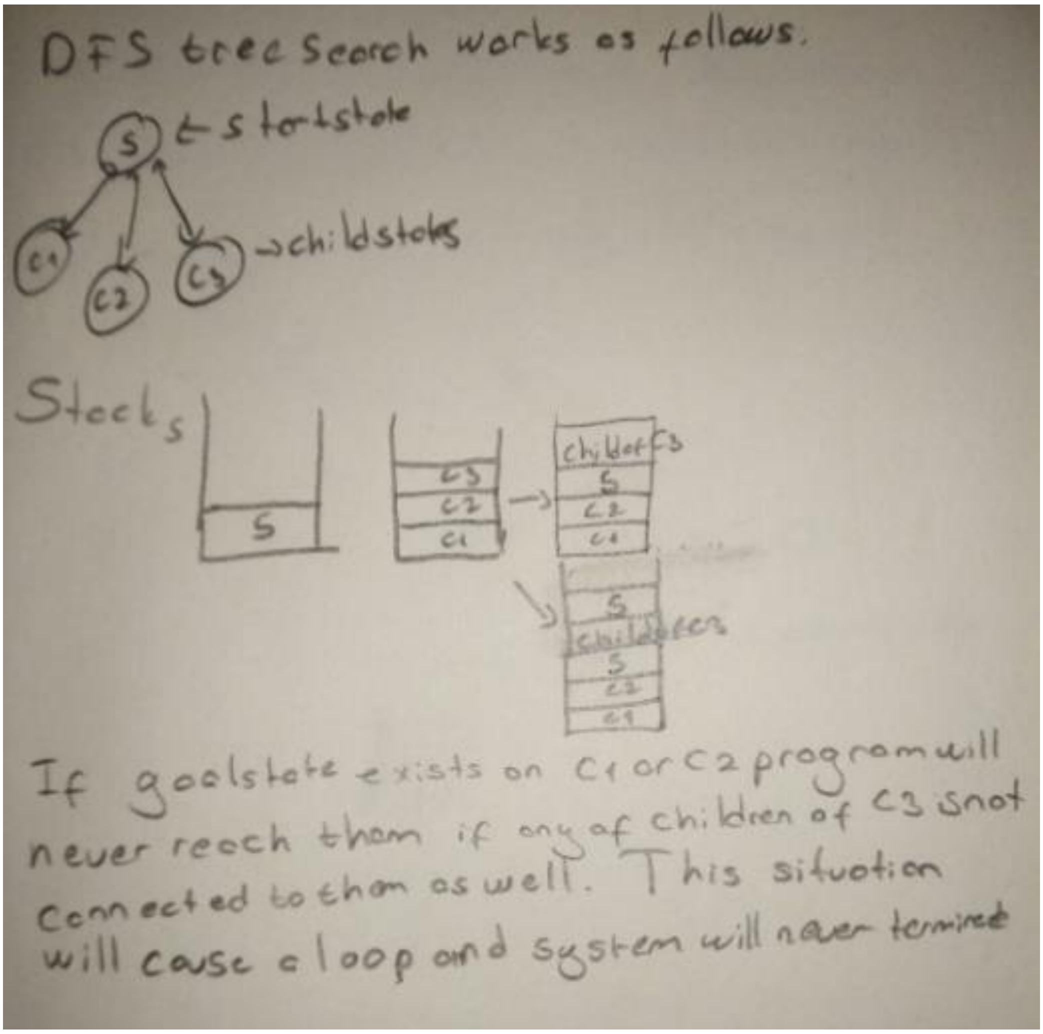
Iterate through parent processes from reached goal state up to start state and construct a path.

(a) Depth-First Search (DFS) and Breadth-First Search (BFS)

BFS tree search works as follows

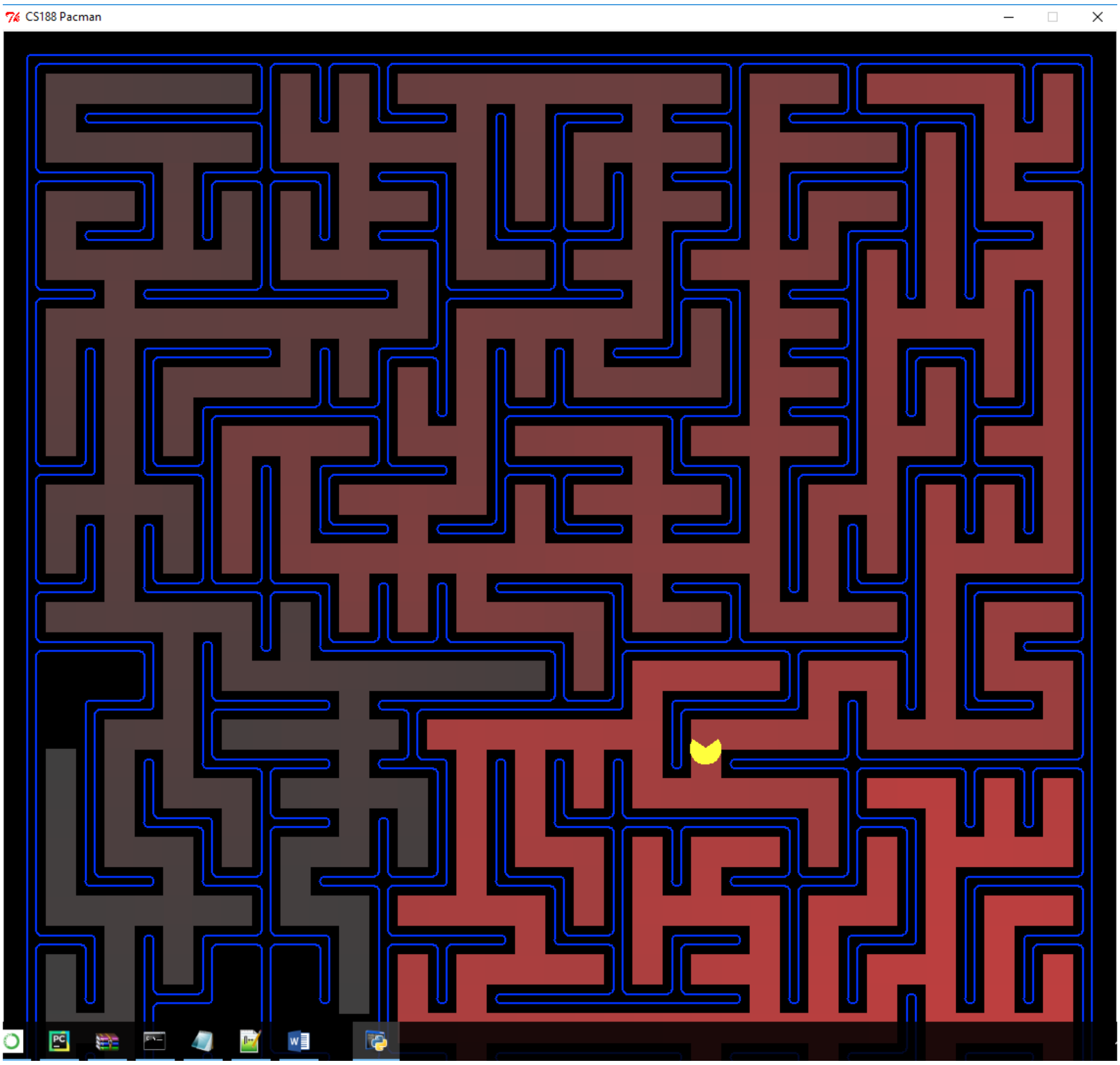


Theris a constant loop of going through all existing nodes. As also new nodes added to space I suggest that with enough node space program can reach goal state. As number of nodes is high that computer cannot operate, we cannot monitor the success to b

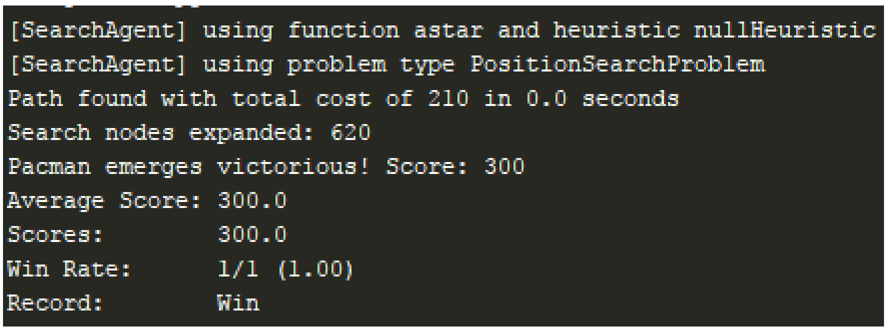


Through out the implementation both dfs and bfs did not stop searching for goal state, could not reach it because of loops.

(b) A* Search

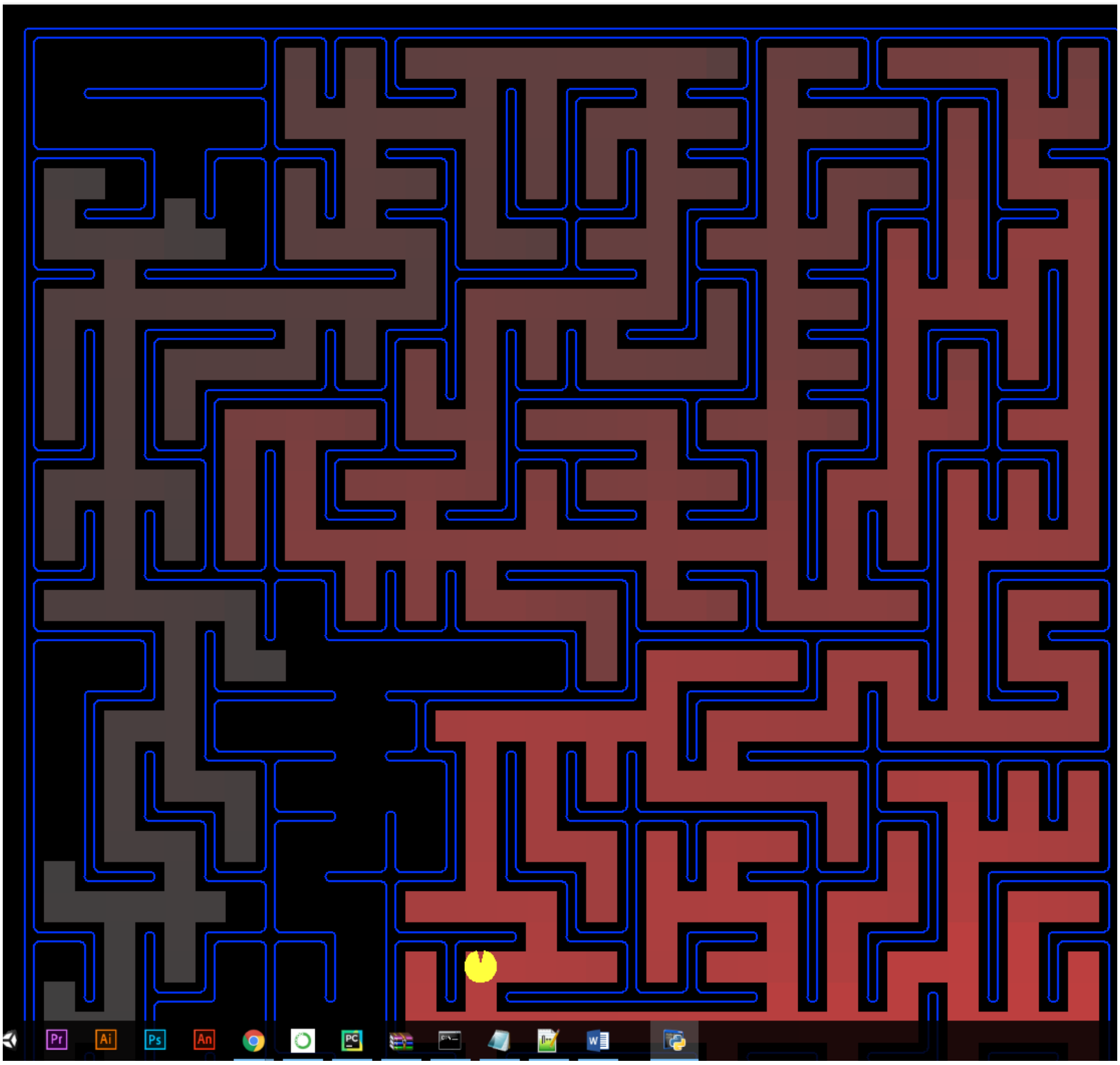


Gamemat discovered by h(n) = 1



```
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Output of run with h(n)=1



Gamemat discovered by h(n) = manhattanHeuristic

```
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 556
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Output of run with h(n) = manhattanHeuristic

```
#heuristicN=node["cost"]+searchAgents.manhattanHeuristic(node["state"],problem)
heuristicN=1
frontier.push(node,heuristicN)
```

By uncommenting

```
heuristicN=node["cost"]+searchAgents.manhattanHeuristic(node["state"],problem)
```

we use h(n) = manhattanHeuristic

By uncommenting

```
heuristicN=1
```

we use h(n) = 1

From above output of two A* implementations, it can be seen that there was not a big difference on time consumption. However, manhattanHeuristic showed better performance on node generation as it is more successful on deciding cost of the system.

Admissibblity of h(n) = manhattanHeuristic is proven by for the second function Manhattan distance computation is chosen as $k(n)$

best case $k(n) = \sqrt{(x_g - x_n)^2 + (y_g - y_n)^2}$

$k(n) = h(n)$

other cases

$k(n) \leq h(n)$

$h(g) = 0$

↓

$h(n) \leq k(n)$

$k(n) \geq 0 \quad k(g) = 0$

↓

Manhattan distance is admissible