CS5100 HOMEWORK 1

Sudharshan Subramaniam Janakiraman

06/29/2020

THEORY

1. 2.4

➔ *Playing Soccer:*

P : Number of Goals , Game Won

E : Soccer Ground, Clock

A : Legs, Head, Chest

S : Eyes, Ears, Mouth

Properties : Partially Observable, Multi Agent , Stochastic, Sequential, Dynamic, Continuous, Unknown.

➔ *Exploring the Subsurface Oceans of Titan:*

P : Area Explored, Minerals and Lifeforms discovered

E : Subsurface oceans of Titan

A : Robot Controller, Accelerator, Brake

S : Camera, Communication Sensors

Properties : Partially Observable, Single Agent , Stochastic, Sequential, Dynamic, Continuous, Unknown.

➔ *Playing a Tennis Match:*

P : Match won/lost , Number of Unforced & Forced errors

E : Tennis Court

A : Tennis Racquet , Hands, Legs/wheelChair

S : Eyes

Properties : Partially Observable, Multi Agent , Stochastic, Sequential, Dynamic, Continuous, Unknown.

➔ *Practicing tennis against walls:*

P : Match won/lost after practice

E : Wall with practice area

A : Tennis Racquet , Hands, Legs/wheelChair

S : Eyes

Properties : Fully Observable, Single Agent , Stochastic, Sequential, Dynamic, Continuous, Unknown.

[P : Performance Measure, E : Environment, A : Actuators, S : Sensors]

2. 2.5

➔ **Agent** : An entity that collects information of its environment through sensors and executes actions with actuators.

➔ ***Agent Function*** : Abstract description/ Relation of what actions needs to be executed based on perceptual sequence

➔ **Agent Program** : Real World implementation of Agent Function mapping percept sequence and actions

➔ **Rationality** : The action of maximizing performance measure

➔ **Autonomy** : Ability to self govern under dynamic/unknown environments

➔ **Reflex Agent** : Agent that takes actions based on current input and does not care about consequences

➔ **Model Based Agent** : Agent that executes action based on current percept and description of external world from current inputs

➔ **Goal Based Agent** : Agent that executes action based on current percepts and available goal information

➔ **Utility Based agent** : Agent that executes actions that will maximize performance measure based on current inputs, goal and best possible actions sequence

3. 2.7

PSEUDO CODE _ GOAL BASED AGENT

function GOAL-BASED-AGENT(percept) **returns** an action

persistent: state, the agent's current conception of the world state

model , a description of how the next state depends on current state and action

goals, a set of goals the agent wishes to achieve

actionlist, set of actions that will lead the agent to goal, initially none

action, the most recent action, initially none

constraints, helps agent to achieve goal

state ← UPDATE-STATE(state,action,percept,model)

actionlist ← GENERATE-ACTIONLIST-TO-ACHIEVE-GOAL(state, goals,constraints)

action ← actionlist.ACTION

return action

PSEUDO CODE _ UTILITY BASED AGENT

function UTILITY-BASED-AGENT(percept) **returns** an action

persistent: state, the agent's current conception of the world state

model , a description of how the next state depends on current state and action

goals, a set of goal the agent wishes to achieve

action, the most recent action, initially none

constraints, helps agent to achieve goal

state ← UPDATE-STATE(state,action,percept,model)

utility ← ACTIONLIST-TO-MAXIMIZE-PERFORMANCE-MEASURE(state, goals,constraints)

action ← utility.ACTION

return action

4. 3.6

a. Using only four colors, you have to color a planar map so that no two adjacent regions have the same color.

- ➔ **States**: The state is determined by whether all the locations in the map are colored or not.. A map environment with n locations has 2^n states.
- ➔ **Initial state**: Colorless/Uncolored Map.
- ➔ **Actions**: Assigning a color to an uncolored location of a map.
- ➔ **Transition model**: The actions have their expected effects, previously uncolored location will have an assigned color to it
- ➔ **Goal test**: Checks if all location are colored and no two adjacent locations share the same color
- ➔ **Path cost**: Each color assignment costs 1, so the path cost is the number of assignments in the map.

b. A 3-foot tall monkey is in a room where some bananas are suspended from the 8-foot ceiling. He would like to get the bananas. The room contains two stackable, movable, climbable 3-foot-high crates.

- ➔ **States**: Each state obviously includes a location of bananas, monkey, crates
- ➔ **Initial state**: Bananas suspended from ceiling, Monkey and Crates present on the floor.
- ➔ **Actions**: Monkey moves from one location to another, crates moved from one location to another, crates placed on one another, monkey standing on ground, monkey standing over a crate, Grabbing a banana if possible
- ➔ **Transition model**: The actions have their expected effects,
- ➔ **Goal test**: Checks if all the bananas are grabbed by monkey
- ➔ **Path cost**: Each action costs 1, so the path cost is the number of actions taken to grab all bananas.

5. 3.10

Define in your own words the following terms: state, state space, search tree, search node, goal, action, transition model, and branching factor.

- ➔ **State**: Representation of agents world at a particular instant of time
- ➔ **State Space**: An abstract representation of all the possible states of the agent in that world
- ➔ **Search Tree**: A tree structure containing the initial state as its root node and each node in the tree connects to neighbouring node by actions taken by the agent
- ➔ **Search Node**: Any node that is present in the search tree
- ➔ **Goal** : Agents desired state that it wishes to achieve
- ➔ **Action**: A reaction by the agent based on the perceptual inputs it receive
- ➔ **Transition Model:** description of the effect caused by actions taken by the agent
- ➔ **Branching Factor:** The number of child nodes we can visit from a parent node is called as branching factor

6. **Sudoku**:
   a. *state representation for Sudoku*

State:

- ➔ 9 X 9 Matrix
- ➔ {1,2,3,4,5,6,7,8,9} as elements of partially filled spaces of grid
- ➔ {0} for blank spaces in the grid
- ➔ No duplication of numbers in row, column, sub square

Initial State:

- ➔ state given in the problem (it can be any state if not specified)

Actions:

- ➔ Assigning valid values {0,1,2,3,4,5,6,7,8,9} to locations filled as 0

Transition Model:

- ➔ New state after action is executed

Goals:

- ➔ All elements of the matrix to be filled with valid numbers without duplicates in row , column, subsquare

Path Cost:

- ➔ Every assignment of values cost one. Path Cost is total number of assignments

b. pseudocode for the successor function

function SUDOKU-STATE-SUCCESSOR(current_state) **returns** a successor_list

persistent: temp: temporary matrix to store current state

successor_list : list of matrices that consists of valid successors of current state, initially none

temp ← current_state

for each row **in** temp

for each column **in** temp

if temp[row][column] is 0:

for each value **in** range(0,9)

temp[row][column] ← value

if not (duplicated(temp[row][]) **and** duplicated(temp[][column]) and duplicated(SUBSQUARE(temp) **):**

successor_list ← temp.APPEND

endif

endfor

endif

endfor

endfor

return successor_list

c. pseudocode for goal function that returns true if a state is a goal.

```
function IS-GOAL-STATE(current_state) returns True or False
    for each row in current_state
        for each column in current_state
            if current_state[row][column] is 0:
                return False
            endif
        endfor
    endfor
    return True
```

d.

Goal : 81 valid elements filled (9 * 9 = 81)

Given : 28 elements filled

Shortest path L : 81 - 28 = 53

7.

a. State representation and Traversal of tree in bfs and dfs

➔ **States**: Each node , connected/disconnected

➔ **Initial state**: Starting Node of the tree

➔ **Actions**: Exploring neighbors of a node

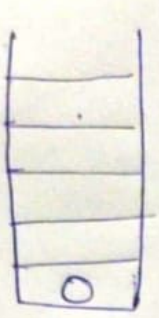
➔ **Transition model**: Spanning tree after every action

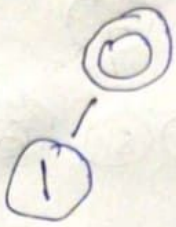
➔ **Goal test**: To explore every node in the search tree (to find a goal if one exists)

➔ **Path cost**: Each action costs 1, so the path cost is the number of actions taken to explore all nodes

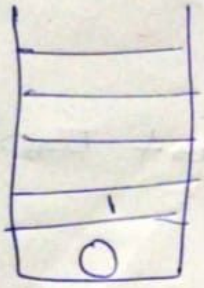
DEPTH FIRST SEARCH

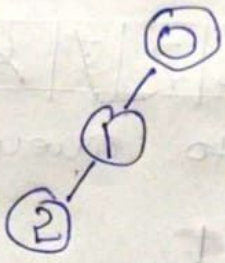
Step 1: Starting vertex : 0
Stack is used to store explored nodes



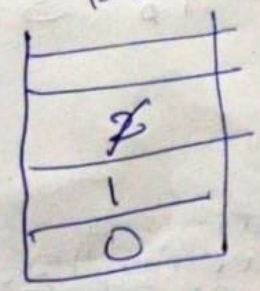


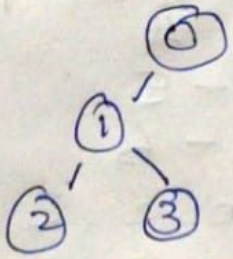
Step 2: Explore 1



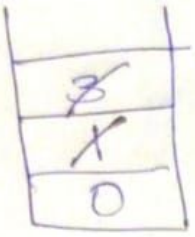


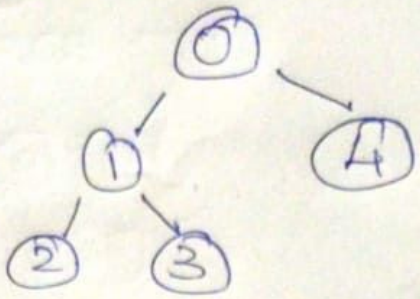
Step 3: Explore 2: Since there is no node beyond go back to previous node and explore again



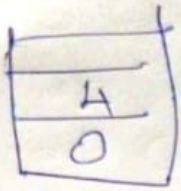


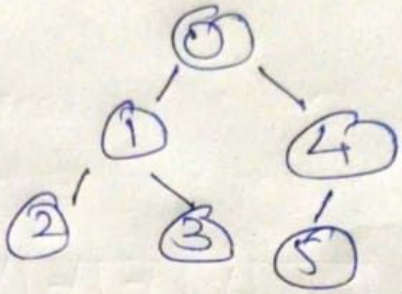
Step 4: Explore 3: Since there is no neighbour to 3 go to previous node, since 1 & 0 also explored go to previous node and start exploring 0





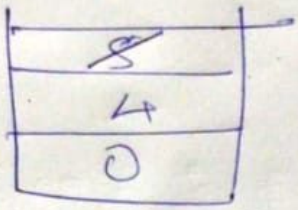
Step 5 : Start exploring 4

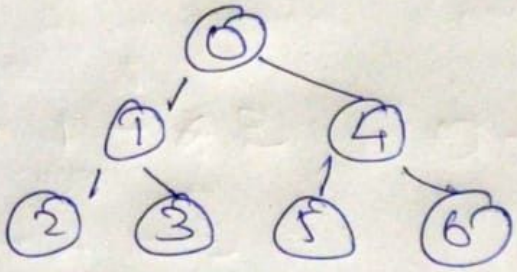




Step 6 : Start exploring 5

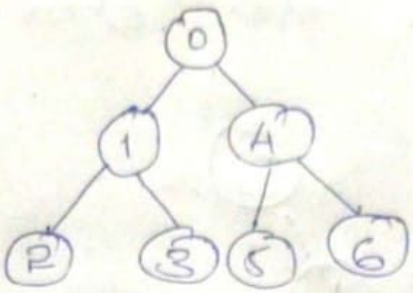
Since no future node exists for (5) go back to (4) and explore





Step 7 : go back to 4 since nothing to expand for (6)
go back to 0, since 4 is fully explored
pop 0 from stack as it is fully explored

SEARCH TREE



BFS :

Let Start Vertex : 0

[using Queue to explore Nodes

0	1	4	2	3	5	6

Step 1 : 0 to Queue, Start Explori...

0

Step 2 : 0 - 1

0 - 4

Step 3 : [randomly explore 1 or 4] Let's Explore 1

0 - 1 - 2 , 0 - 1 - 3

Step 4 : Explore 4

0 - 4 - 5

0 - 4 - 6

Step 5 : Explore 2 [Nothing further to explore]

0 - 1 - 2

Step 6 : Explore 3 "

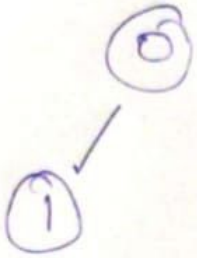
0 - 1 - 3

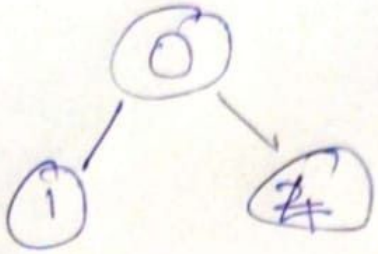
Step 7 : Explor 5 : 0 - 4 - 5

Step 8 : Explore 6 : 0 - 4 - 6

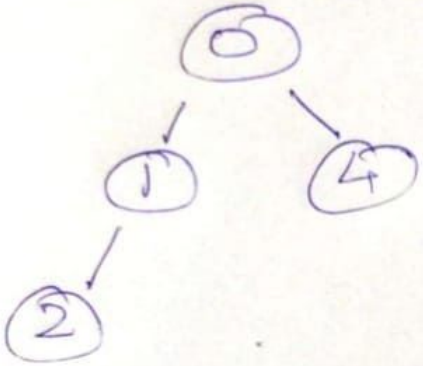
BFS : 0 1 4 2 3 5 6 :

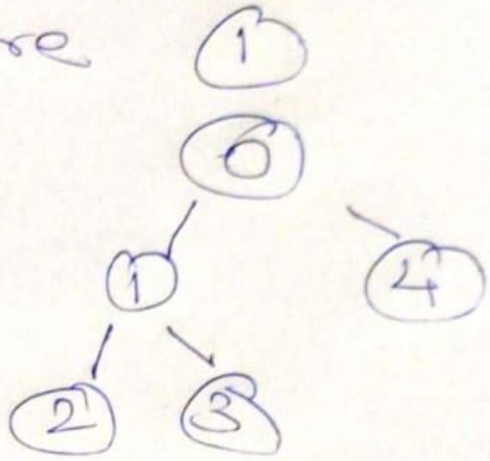
Step 1 : Explore 0



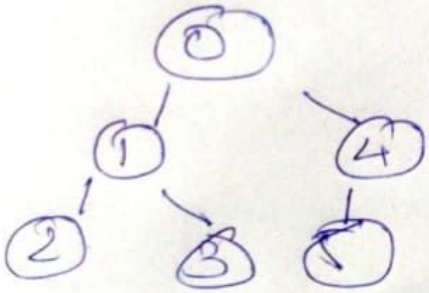


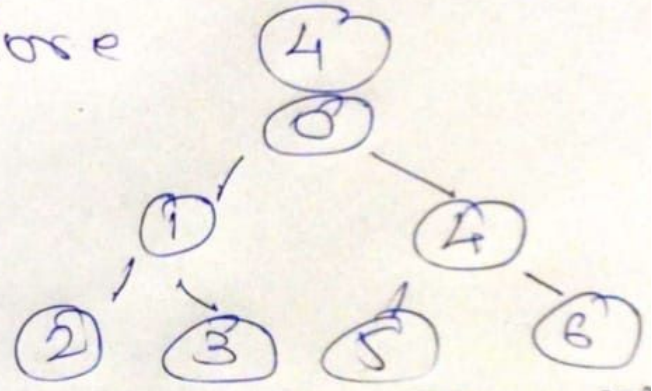
Step (2) : Explore 1





Step (3) Explore 4

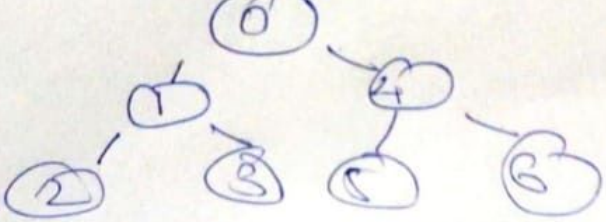




Sink not

Exploring Expand

2, 3, 5, 6 are leaf nodes



b. Python implementation of BFS and DFS

Program and O/P of breadth first search implementation of basic search tree

```
def bfs(graph, node):
    visited = []
    queue = []
    queue.append(node)
    print("Queue ",queue)
    while queue:
        s = queue.pop(0)
        print ("Explored",s, end = " ")
        if s not in visited:
            visited.append(s)
            print ("Visited ", visited, end = " ")
            for neighbour in graph[s]:
                queue.append(neighbour)
            print("Queue ",queue)

graph = {
  0 : [1,4],
  1 : [2, 3],
  4 : [5,6],
  2 : [],
  3 : [],
  5 : [],
  6 : []
}

bfs(graph, 0)
```

```
Queue  [0]
Explored 0 Visited  [0] Queue  [1, 4]
Explored 1 Visited  [0, 1] Queue  [4, 2, 3]
Explored 4 Visited  [0, 1, 4] Queue  [2, 3, 5, 6]
Explored 2 Visited  [0, 1, 4, 2] Queue  [3, 5, 6]
Explored 3 Visited  [0, 1, 4, 2, 3] Queue  [5, 6]
Explored 5 Visited  [0, 1, 4, 2, 3, 5] Queue  [6]
Explored 6 Visited  [0, 1, 4, 2, 3, 5, 6] Queue  []
```

Program and O/P of Depth first search implementation of basic search tree

```
def dfs(graph, node):
    visited = list()
    stack = set()
    stack.add(node)
    print("stack ",stack)
    while stack:
        s = stack.pop()
        print ("Explored ",s, end = " ")
        if s not in visited:
            visited.append(s)
            print ("Visited ", visited, end = " ")
            for neighbour in graph[s]:
                stack.add(neighbour)
            print("stack ",stack)

graph = {
  0 : [1,4],
  1 : [2, 3],
  4 : [5,6],
  2 : [],
  3 : [],
  5 : [],
  6 : []
}

dfs(graph, 0)
```

```
stack  {0}
Explored  0 Visited  [0] stack  {1, 4}
Explored  1 Visited  [0, 1] stack  {2, 3, 4}
Explored  2 Visited  [0, 1, 2] stack  {3, 4}
Explored  3 Visited  [0, 1, 2, 3] stack  {4}
Explored  4 Visited  [0, 1, 2, 3, 4] stack  {5, 6}
Explored  5 Visited  [0, 1, 2, 3, 4, 5] stack  {6}
Explored  6 Visited  [0, 1, 2, 3, 4, 5, 6] stack  set()
```

PROGRAMMING ASSIGNMENT
PACMAN OUTPUT

➔ 4.A* Search


```
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
(base) SUDHARSHANs-MacBook-Air:search sudharshan$
```

➔ 5.Finding All the Corners

- Starting state : pacman starting position and foodPellet list of length 4 consisting of all True values indicating the food pellets have not been eaten by the Pacman
- Goal State : All values in the food PelletList are False
- Successors : if the next position is a corner then change the foodPellet list value of corresponding corner to False

→ Tiny Corners ←

```
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 252
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:        512.0
Win Rate:      1/1 (1.00)
Record:        Win
```

→ Medium Corners ←

```
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.2 seconds
Search nodes expanded: 1966
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

➔ 6.Corners Problem: Heuristic

Heuristic = Maximum of Manhattan distance between current state of pacman and unvisted corners of the pac

```
Path found with total cost of 106 in 0.1 seconds
Search nodes expanded: 1136
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

➔ 7.Eating All The Dots

→ Test Search ←

```
Path found with total cost of 7 in 0.0 seconds
Search nodes expanded: 11
Pacman emerges victorious! Score: 513
Average Score: 513.0
Scores:        513.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Food Heuristic : returns the maximum maze distance the agent has to travel between the current position and all the available food pellets

→ Tricky Search ←

```
Path found with total cost of 60 in 85.9 seconds
Search nodes expanded: 8075
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
(base) SUDHARSHANs-MacBook-Air:search sudharshan$
```

➔ Suboptimal Search

→ Closest Dot Search ←

```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 350.
Pacman emerges victorious! Score: 2360
Average Score: 2360.0
Scores:        2360.0
Win Rate:      1/1 (1.00)
Record:        Win
```

→ Submission_autograder.py ←

```
(base) SUDHARSHANs-MacBook-Air:search sudharshan$ python submission_autograder.py
--------------------------------------------------------------------------------

CS 188 Local Submission Autograder
Version 1.4.0

--------------------------------------------------------------------------------

Setting up environment....................................................... DONE
Downloading autograder....................................................... DONE
Extracting autograder........................................................ DONE
Preparing student files:
  searchAgents.py.................. OK
  search.py........................ OK
Running tests (this may take a while):
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of imp
  import imp
  Question q1..................... 0/3
  Question q2..................... 3/3
  Question q3..................... 0/3
  Question q4..................... 3/3
  Question q5..................... 3/3
  Question q6..................... 3/3
  Question q7..................... 4/4
  Question q8..................... 3/3
Generating submission token.................................................. DONE

--------------------------------------------------------------------------------

Final score: 19/25
Token file: search.token
```

CONTINUUM VACUUM WORLD OUTPUT
AGENT 1

➔ Reflex agent:
- ◆ Agent Sucks the dirt if present in the current location
- ◆ Moves opposite to wall location (i.e., if it has a wall left to it , it moves in the right direction)
- ◆ Moves randomly inside the grid

N E (3,5)	S W (7, 1)
S 0.20 L 0.20 S 0.50 U 0.50 S 1.00 0.0 0.5 0.8 0.1 0.1 0.1 0.0 0.5 [0.0] 0.5 0.3 0.5 0.4 0.0 0.0 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0 U 1.00 S 1.10 D 1.10 U 1.10 D 1.10	R 0.00 U 0.00 R 0.00 L 0.00 D 0.00 0.0 0.5 0.8 0.1 0.1 0.1 0.0 0.5 0.5 0.5 0.3 0.5 0.4 0.3 0.2 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 0.5 0.1 0.0 [0.0] 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0 U 0.00 R 0.00 R 0.00 S 0.50 D 0.50

0.0 0.5 0.8 0.0 0.1 0.1 0.0 0.5 [0.0] 0.5 0.3 0.5 0.4 0.0 0.0 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0 U 1.10 D 1.10 U 1.10 D 1.10 R 1.10	0.0 0.5 0.8 0.1 0.1 0.1 0.0 0.5 0.5 0.5 0.3 0.5 0.4 0.3 0.2 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 [0.5] 0.1 0.0 0.0 0.0 0.2 0.0 S 1.00 L 1.00 R 1.00 L 1.00 L 1.00
0.0 0.5 0.8 0.0 0.1 0.1 0.0 0.5 0.0 [0.5] 0.3 0.5 0.4 0.0 0.0 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0 S 1.60 L 1.60 U 1.60 D 1.60 R 1.60	0.0 0.5 0.8 0.1 0.1 0.1 0.0 0.5 0.5 0.5 0.3 0.5 0.4 0.3 0.2 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 0.0 0.1 0.0 [0.0] 0.0 0.0 0.1 0.0 0.0 0.0 0.2 0.0 R 1.00 L 1.00 L 1.00 R 1.00 U 1.00

0.0 0.5 0.8 0.0 0.1 0.1 0.0 0.5 0.0 [0.0] 0.3 0.5 0.4 0.0 0.0 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0 L 1.60 R 1.60 L 1.60 L 1.60 S 2.10	0.0 0.5 0.8 0.1 0.1 0.1 0.0 0.5 0.5 0.5 0.3 0.5 0.4 0.3 0.2 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 [0.0] 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.2 0.0 U 1.00 D 1.00 R 1.00 U 1.00 S 1.20
0.0 0.5 0.8 0.0 0.1 0.1 0.0 [0.0] 0.0 0.0 0.3 0.5 0.4 0.0 0.0 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0 U 2.10 S 2.90 D 2.90 L 2.90 D 2.90	0.0 0.5 0.8 0.1 0.1 0.1 0.0 0.5 0.5 0.5 0.3 0.5 0.4 0.3 0.2 0.3 0.1 0.7 0.8 0.2 0.0 0.0 [0.0] 0.8 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.2 0.0 D 1.20 L 1.20 L 1.20 R 1.20 U 1.20

0.0 0.5 0.0 0.0 0.1 0.1 0.0 0.0 0.0 0.0 0.3 [0.5] 0.4 0.0 0.0 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0	0.0 0.5 0.8 0.1 0.1 0.1 0.0 0.5 0.5 0.5 0.3 0.5 0.4 0.3 0.2 0.3 0.1 0.7 0.8 0.2 0.0 [0.0] 0.0 0.8 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.2 0.0

AGENT 2

- ➔ Greedy Agent
  - ◆ Sucks dirt if present in the current state
  - ◆ Moves to neighbour with maximum dirt
  - ◆ Moves randomly if the more than one neighbours have same dirt value

S 0.20 U 0.20 S 0.70 L 0.70 S 1.20 0.0 0.5 0.8 0.1 0.1 0.1 0.0 0.5 [0.0] 0.0 0.3 0.5 0.4 0.3 0.0 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.5 0.1	U 0.00 R 0.00 R 0.00 R 0.00 S 0.50 0.0 0.5 0.8 0.1 0.1 0.1 0.0 0.5 0.5 0.5 0.3 0.5 0.4 0.3 0.2 0.3 0.1 0.7 0.8 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 [0.0] 0.1 0.0 0.0 0.0 0.5 0.1

0.0 0.0 0.0 0.2 0.0 L 1.20 S 1.70 U 1.70 S 2.50 L 2.50	0.0 0.0 0.0 0.2 0.0 U 0.50 S 1.30 U 1.30 S 2.10 L 2.10
0.0 [0.5] 0.0 0.1 0.1	0.0 0.5 0.8 0.1 0.1
0.1 0.0 0.0 0.0 0.0	0.1 0.0 0.5 0.5 0.5
0.3 0.5 0.4 0.3 0.0	0.3 0.5 0.4 0.3 0.2
0.3 0.1 0.7 0.8 0.2	0.3 0.1 [0.7] 0.0 0.2
0.0 0.0 0.2 0.8 0.3	0.0 0.0 0.2 0.0 0.3
0.0 0.0 0.0 0.5 0.1	0.0 0.0 0.0 0.0 0.1
0.0 0.0 0.0 0.5 0.1	0.0 0.0 0.0 0.5 0.1
0.0 0.0 0.0 0.2 0.0	0.0 0.0 0.0 0.2 0.0
S 3.00 D 3.00 D 3.00 S 3.50 R 3.50	S 2.80 U 2.80 S 3.20 U 3.20 S 3.70
0.0 0.0 0.0 0.1 0.1	0.0 0.5 0.8 0.1 0.1
0.1 0.0 0.0 0.0 0.0	0.1 0.0 [0.0] 0.5 0.5
0.3 0.0 [0.4] 0.3 0.0	0.3 0.5 0.0 0.3 0.2
0.3 0.1 0.7 0.8 0.2	0.3 0.1 0.0 0.0 0.2
0.0 0.0 0.2 0.8 0.3	0.0 0.0 0.2 0.0 0.3
0.0 0.0 0.0 0.5 0.1	0.0 0.0 0.0 0.0 0.1
0.0 0.0 0.0 0.5 0.1	0.0 0.0 0.0 0.5 0.1
0.0 0.0 0.0 0.2 0.0	0.0 0.0 0.0 0.2 0.0

S 3.90 D 3.90 S 4.60 R 4.60 S 5.40	U 3.70 S 4.50 L 4.50 S 5.00 R 5.00
0.0 0.0 0.0 0.1 0.1 0.1 0.0 0.0 0.0 0.0 0.3 0.0 0.0 0.3 0.0 0.3 0.1 0.0 [0.0] 0.2 0.0 0.0 0.2 0.8 0.3 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0	0.0 0.0 [0.0] 0.1 0.1 0.1 0.0 0.0 0.5 0.5 0.3 0.5 0.0 0.3 0.2 0.3 0.1 0.0 0.0 0.2 0.0 0.0 0.2 0.0 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0
D 5.40 S 6.20 D 6.20 S 6.70 D 6.70	R 5.00 S 5.10 D 5.10 S 5.60 R 5.60
0.0 0.0 0.0 0.1 0.1 0.1 0.0 0.0 0.0 0.0 0.3 0.0 0.0 0.3 0.0 0.3 0.1 0.0 0.0 0.2 0.0 0.0 0.2 0.0 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 [0.5] 0.1 0.0 0.0 0.0 0.2 0.0	0.0 0.0 0.0 0.0 0.1 0.1 0.0 0.0 0.0 [0.5] 0.3 0.5 0.0 0.3 0.2 0.3 0.1 0.0 0.0 0.2 0.0 0.0 0.2 0.0 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0

S 7.20 D 7.20 S 7.40 R 7.40 U 7.40 0.0 0.0 0.0 0.1 0.1 0.1 0.0 0.0 0.0 0.0 0.3 0.0 0.0 0.3 0.0 0.3 0.1 0.0 0.0 0.2 0.0 0.0 0.2 0.0 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0 [0.1] 0.0 0.0 0.0 0.0 0.0	S 6.10 D 6.10 S 6.30 L 6.30 S 6.60 0.0 0.0 0.0 0.0 0.1 0.1 0.0 0.0 0.0 0.0 0.3 0.5 0.0 [0.0] 0.0 0.3 0.1 0.0 0.0 0.2 0.0 0.0 0.2 0.0 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0

AGENT 3

➔ Optimised Agent

- ◆ Sucks dirt if present in the current state
- ◆ Moves to neighbour with maximum dirt
- ◆ Keeps track of visited nodes to avoid visiting again
- ◆ Moves randomly if and only all the neighbours of the current state are already visited

S 0.20 U 0.20 S 0.70 L 0.70 S 1.20	D 0.00 R 0.00 R 0.00 R 0.00 S 0.20

0.0 0.5 0.8 0.1 0.1

0.1 0.0 0.5 [0.0] 0.0

0.3 0.5 0.4 0.3 0.0

0.3 0.1 0.7 0.8 0.2

0.0 0.0 0.2 0.8 0.3

0.0 0.0 0.0 0.5 0.1

0.0 0.0 0.0 0.5 0.1

0.0 0.0 0.0 0.2 0.0

L 1.20
S 1.70
U 1.70
S 2.50
L 2.50

0.0 [0.5] 0.0 0.1 0.1

0.1 0.0 0.0 0.0 0.0

0.3 0.5 0.4 0.3 0.0

0.3 0.1 0.7 0.8 0.2

0.0 0.0 0.2 0.8 0.3

0.0 0.0 0.0 0.5 0.1

0.0 0.0 0.0 0.5 0.1

0.0 0.0 0.0 0.2 0.0

S 3.00
L 3.00
D 3.00
S 3.10
D 3.10

0.0 0.5 0.8 0.1 0.1

0.1 0.0 0.5 0.5 0.5

0.3 0.5 0.4 0.3 0.2

0.3 0.1 0.7 0.8 0.2

0.0 0.0 0.2 0.8 0.3

0.0 0.0 0.0 0.5 0.1

0.0 0.0 0.0 0.5 0.1

0.0 0.0 0.0 [0.0] 0.0

U 0.20
S 0.70
U 0.70
S 1.20
U 1.20

0.0 0.5 0.8 0.1 0.1

0.1 0.0 0.5 0.5 0.5

0.3 0.5 0.4 0.3 0.2

0.3 0.1 0.7 0.8 0.2

0.0 0.0 0.2 [0.8] 0.3

0.0 0.0 0.0 0.0 0.1

0.0 0.0 0.0 0.0 0.1

0.0 0.0 0.0 0.0 0.0

S 2.00
U 2.00
S 2.80
L 2.80
S 3.50

0.0 0.0 0.0 0.1 0.1

0.0 0.0 0.0 0.0 0.0

[0.3] 0.5 0.4 0.3 0.0

0.3 0.1 0.7 0.8 0.2

0.0 0.0 0.2 0.8 0.3

0.0 0.0 0.0 0.5 0.1

0.0 0.0 0.0 0.5 0.1

0.0 0.0 0.0 0.2 0.0

S 3.40
R 3.40
S 3.90
R 3.90
S 4.30

0.0 0.0 0.0 0.1 0.1

0.0 0.0 0.0 0.0 0.0

0.0 0.0 [0.0] 0.3 0.0

0.3 0.1 0.7 0.8 0.2

0.0 0.0 0.2 0.8 0.3

0.0 0.0 0.0 0.5 0.1

0.0 0.0 0.0 0.5 0.1

0.0 0.0 0.0 0.2 0.0

D 4.30
S 5.00
R 5.00
S 5.80
D 5.80

0.0 0.5 0.8 0.1 0.1

0.1 0.0 0.5 0.5 0.5

0.3 0.5 0.4 0.3 0.2

0.3 0.1 [0.0] 0.0 0.2

0.0 0.0 0.2 0.0 0.3

0.0 0.0 0.0 0.0 0.1

0.0 0.0 0.0 0.0 0.1

0.0 0.0 0.0 0.0 0.0

U 3.50
S 3.90
U 3.90
S 4.40
U 4.40

0.0 0.5 [0.8] 0.1 0.1

0.1 0.0 0.0 0.5 0.5

0.3 0.5 0.0 0.3 0.2

0.3 0.1 0.0 0.0 0.2

0.0 0.0 0.2 0.0 0.3

0.0 0.0 0.0 0.0 0.1

0.0 0.0 0.0 0.0 0.1

0.0 0.0 0.0 0.0 0.0

S 5.20
L 5.20
S 5.70
D 5.70
D 5.70

0.0 0.0 0.0 0.1 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.3 0.0 0.3 0.1 0.0 0.0 0.2 0.0 0.0 0.2 [0.8] 0.3 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.5 0.1 0.0 0.0 0.0 0.2 0.0 S 6.60 D 6.60 S 7.10 D 7.10 S 7.60	0.0 0.0 0.0 0.1 0.1 0.1 0.0 0.0 0.5 0.5 0.3 [0.5] 0.0 0.3 0.2 0.3 0.1 0.0 0.0 0.2 0.0 0.0 0.2 0.0 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.0 S 6.20 L 6.20 S 6.50 D 6.50 S 6.80
0.0 0.0 0.0 0.1 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.3 0.0 0.3 0.1 0.0 0.0 0.2 0.0 0.0 0.2 0.0 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 [0.0] 0.1 0.0 0.0 0.0 0.2 0.0	0.0 0.0 0.0 0.1 0.1 0.1 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.3 0.2 [0.0] 0.1 0.0 0.0 0.2 0.0 0.0 0.2 0.0 0.3 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0 0.0

Starting Point	Part A	Part B	Part C
N E (3, 5)	**2.90 Poor**	**7.40 better to best**	**7.60 better to best**
S W(7, 1)	**1.20 Poor**	**6.60 better to best**	**6.80better to best**