Ateneo De Manila University
School of Science and Engineering

Technical Documentation

Meljohn Ugaddan
CSCI 211
August 8, 2021

A. Project Specification

- ➢ Programming Language : Python 3.8.8
- ➢ Jupyter Notebook Information:

```
jupyter core     : 4.7.1
jupyter-notebook : 6.3.0
qtconsole        : 5.0.3
ipython          : 7.22.0
ipykernel        : 5.3.4
jupyter client   : 6.1.12
jupyter lab      : 3.0.14
nbconvert        : 6.0.7
ipywidgets       : 7.6.3
nbformat         : 5.1.3
traitlets        : 5.0.5
```

B. Implementation of Depth-first search

The first mode is the Depth-First Search. Depth-first search is implemented with the use of stack. Adjacent items are stacked in this manner, left,down,right then upward. Let's take example the point (0,0) , the next item will be row (0,1) because that is the item that will be access by the stack

1 →	2 →	3 →	4 →	5 ↓
22	21	14	13	6 ↓
23	20	15	12	7
24	19	16	11	8
25	18	17	10	9

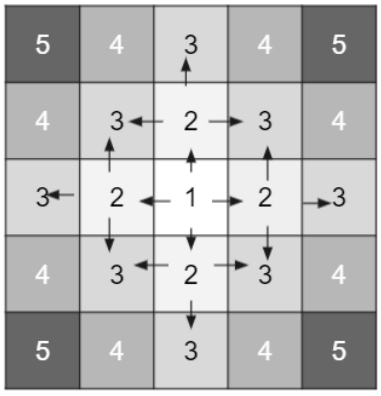
21	20	3 →	4 →	5 ↓
22	19	2 ↑	13	6 ↓
23	18	1 ↑	12	7 ↓
24	17	14	11	8
25	16	15	10	9

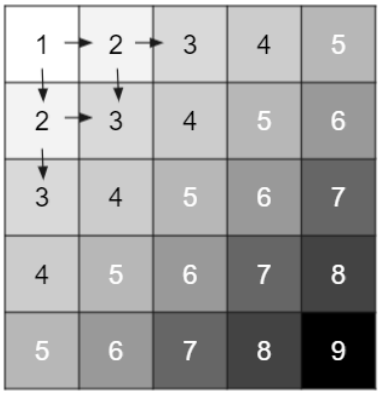
```
def grid_dfs(row, col , grid):
    stack = []
    stack.append([row, col])
    while (len(stack) > 0):
        current_point = stack[len(stack) - 1]
        stack.remove(stack[len(stack) - 1])
        row = current_point[0]
        col = current_point[1]

        if ( is_visited(row, col) == False ):
            continue

        knowledge[row][col][VISITED] = True
        for i in range(4):
                index_i = row + adjacent_i[i]
                index_j = col + adjacent_j[i]
                if(isValidRange((index_i,index_j))):
                    stack.append([index_i, index_j])
```

```
adjacent_i = [0, 1, 0, -1]
adjacent_j = [-1, 0, 1, 0]

grid = np.array([
        [1, 2, 3, 4, 5],
        [6, 7, 8, 9, 10],
        [11,12,13,14,15],
        [16,17,18,19,20],
        [21,22,23,24,25],
        ])
```

C. Implementation of Breadth-First Search

The next mode mode is breadth-first. Breadth-first is implemented with the use of queues. Adjacent items from the block (i,j) are looped in this manner, left,down,right then upward. So that each item in the adjacent block will be traverse first in contrast to Depth-First Search





```
def grid_bfs(row, col , grid):
    queue = []
    queue.append([row, col])
    while (len(queue) > 0):
        current_point = queue[0]
        queue.remove(queue[0])
        row = current_point[0]
        col = current_point[1]

        if ( is_visited(row, col) == False ):
            continue

        knowledge[row][col][VISITED] = True
        for i in range(4):
            index_i = row + adjacent_i[i]
            index_j = col + adjacent_j[i]
            if(isValidRange((index_i,index_j))):
                queue.append([index_i, index_j])
```

```
adjacent_i = [0, 1, 0, -1]
adjacent_j = [-1, 0, 1, 0]

grid = np.array([
        [1, 2, 3, 4, 5],
        [6, 7, 8, 9, 10],
        [11,12,13,14,15],
        [16,17,18,19,20],
        [21,22,23,24,25],
        ])
```

D. Implementation for the Inference of Knowledge-Based Agent

Each knowledge of the state is implemented in a 5x5x6 dimensional array. The first two numbers represent the row and grid of the world. Then the last number represents each boolean of the Smelly, Wetsoil, Grass, Quicksand, Visited/Safe and Wolf. Each of these booleans will be used for the propositional rules as the cow agent navigates the world safely with the use of the propositional rule.

Propositional Rule:

1. $Smelly_{i,j} \rightarrow Wolf_{i-1,j}$ OR $Wolf_{i,j-1}$ OR $Wolf_{i+1,j}$ OR $Wolf_{i,j+1}$
2. $Wolf_{i,j} \rightarrow Visited_{i-1,j-1}$ AND $Smelly_{i-1,j}$ AND $Smelly_{i,j-1}$

$(i-1, j-1)$	$(i-1, j)$	$(i-1, j+1)$
$(i, j-1)$	(i, j)	$(i,j+1)$
$(i+1,j-1)$	$(i+1, j)$	$(i+1,j+1)$