

ACI Assignment – 1

Group: 76

Group Members: Kolli Hrudhay (2019AH04040),
Narendrababu S (2019AH04022),
Akshat Gupta (2019AH04001)

Answers to Theoretical Questions:

1. Considering Bloxorz problem as a search problem, below is the formulation of the problem:
 - States Representation: position of the block in the terrain (x1, y1, x2, y2).
 - Initial State: is (2, 2, 2, 2).
 - Actions: block can move right, left, up and down to go to a new state. Only legal moves, which are those moves that generate a new state within the dimensions of the terrain can be applied.
 - Transition Model: Return the new legal position of the block after applying any legal move. The initial state along with the actions and the transition model constitute the state space of the game problem.
 - Goal State: Position (5, 8, 5, 8).
 - Path Cost function $g(n)$: Step cost is 1 for every move. The path cost of any state/node n is the summation of all steps cost from the initial state to that state/node n .
2. Bloxorz problem can be solved using both uninformed search and informed search strategies.

Hence, BFS and DFS being uninformed search strategies can be used to solve this problem.

A. Breadth-First Search

In this search strategy, the “Expand” data structure is a simple FIFO (first in first out) queue. When a current node gets expanded, its children get inserted at the end of the queue and when polling out, always poll out the node at the first. This strategy searches in a wide development direction.

Pseudo code of the BFS algorithm is as follows:

- 1) Predefine the initial node (start node) and the goal node (end node).
- 2) Initialize the initial node of the respective data structure (Queue in case of BFS) to empty.
- 3) Add the initial node into the Expand queue and the Visited list.
- 4) Check if the queue is empty → there is no solution and exit, otherwise continue.
- 5) Poll out the first node of the Expand queue (called parent node).
- 6) Check if it is equal to the goal node → solution is found and return the MoveSequence string of that node, otherwise continue.

7) Expand the node and generate its child nodes which represent the new positions of the block after moving in all 4 directions from the current position defined in the parent node.

8) For each of the generated nodes, if it is valid (block position is valid in the terrain), its MoveSequence equals to its parent's plus the move it generated from, add the node to the Expand queue and the Visited list.

Finally go back to step 4.

B. Depth-First Search

In this search strategy, the "Expand" is a LIFO (last in first out) data structure which is simply a stack. When a current node gets expanded, its children get inserted at the top of the stack and when popping out, always pop out the node at the top. This strategy searches in a deep development direction.

Pseudo code of the DFS algorithm is as follows:

- 1) Predefine the initial node (start node) and the goal node (end node).
- 2) Initialize the initial node of the respective data structure (Stack in case of DFS) to empty.
- 3) Add the initial node into the Expand stack and the Visited list.
- 4) Check if the stack is empty → there is no solution and exit, otherwise continue.
- 5) Pop out the top node of the Expand stack (called parent node).
- 6) Check if it is equal to the goal node → solution is found and return the MoveSequence string of that node, otherwise continue.
- 7) Expand the node and generate its child nodes which represent the new positions of the block after moving in all 4 directions from the current position defined in the parent node.
- 8) For each of the generated nodes, if it is valid (block position is valid in the terrain), its MoveSequence equals to its parent's plus the move it generated from, add the node to the Expand stack and the Visited list.

Finally go back to step 4.

3. A* Search can be used to solve this problem.

In this search strategy, the "Expand" data structure is a priority queue. Its priority is based on an Evaluation function $f(n)$ and the highest priority is for node with the minimum $f(n)$ value. When polling out a node, always poll out the node with the highest priority and the children of an expanded node can be inserted anywhere. This strategy expands only the nodes that's more likely will reach the goal. Finally, in this search algorithm we'll follow Tree Search strategy which doesn't exclude the visited nodes from expanding again since the heuristic evaluation function $h(n)$ used is not consistent.

The main idea of any heuristic search algorithm is to use an estimated evaluation function to guide the search, assess each node and find out the node that more likely leads to the goal. The evaluation function is defined in (1):

$$f(n) = g(n) + h(n) \quad (1)$$

Where $g(n)$ is the actual path cost from the initial state S to the state defined in the current node. The path cost is represented by the number of moves required to reach that state. $h(n)$ is used to estimate path cost from the state defined in the current node to reach the goal state G . Hence, $f(n)$ represents the overall path cost from the initial state S passing by state in node n to the goal state G .

Since the heuristic value is an estimated cost calculated from specific node to the goal node, many calculations could be used as a heuristic function, choosing which one is appropriate depends on the problem itself. In the case of

Bloxorz, Chebyshev distance gives a good estimated distance between current and goal state, plus the cheaper calculation it has when comparing with Euclidean distance as an example. The formula of the function is in (2):

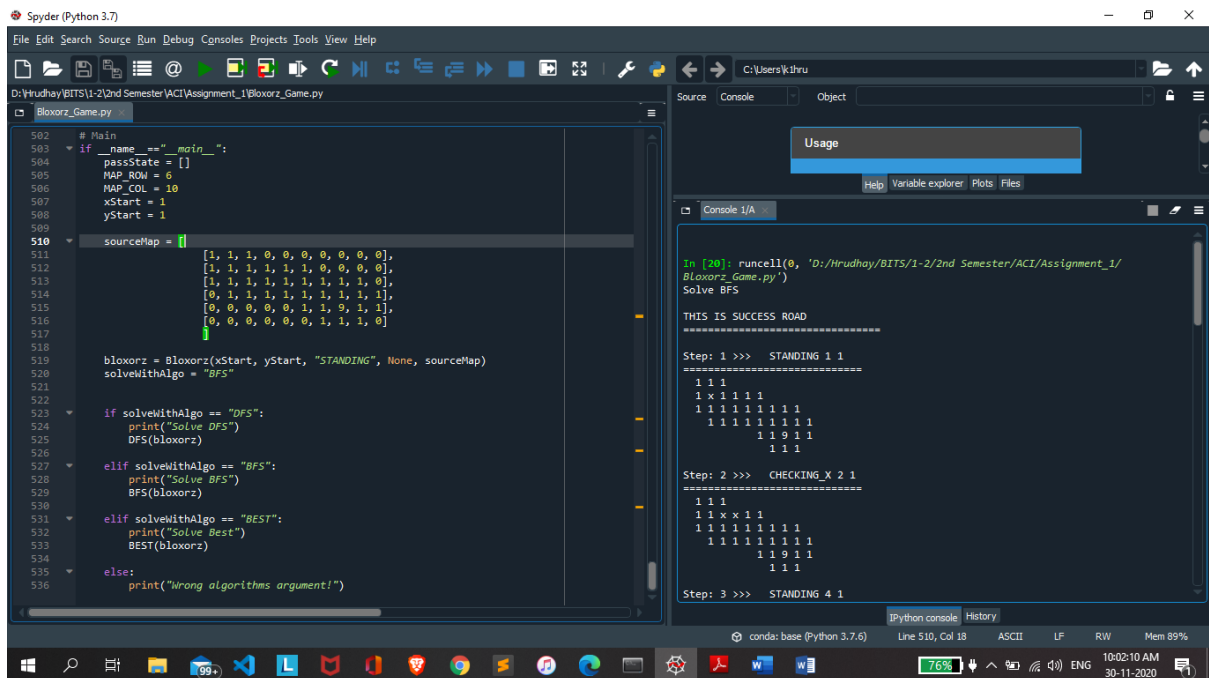
$$h(n) = \max(|x - x_g|, |y - y_g|) \quad (2)$$

Where x is the row number of current positions of block, x_g is the row number of goal position, y and y_g are the same but in terms of columns instead of rows. Since the block has 2 bricks and represented by two x, y coordinates, the heuristic value for the both bricks $h(n)_1$ and $h(n)_2$ are calculated, then the maximum one is considered as the overall heuristic.

Pseudo code of the A* algorithm is as follows:

- 1) Predefine the initial node (start node) and the goal node (end node).
- 2) Initialize the initial node of the respective data structure (Priority Queue in case of A*) to empty.
- 3) Add the initial node into the Expand queue.
- 4) Check if the queue is empty \rightarrow there is no solution and exit, otherwise continue.
- 5) Poll out the highest priority node of the Expand queue (called parent node).
- 6) Check if it is equal to the goal node \rightarrow solution is found and return the MoveSequence string of that node, otherwise continue.
- 7) Expand the node and generate its child nodes which represent the new positions of the block after moving in all 4 directions from the current position defined in the parent node.
- 8) For each of the generated nodes, if it is valid (block position is valid in the terrain), its MoveSequence equals to its parent's plus the move it generated from, compute its $h(n)$ value and $g(n)$ which is equal to its parent's plus 1, then set its $f(n) = h(n) + g(n)$, add the node to the Expand queue. Finally go back to step 4.

4. Output Screenshots:

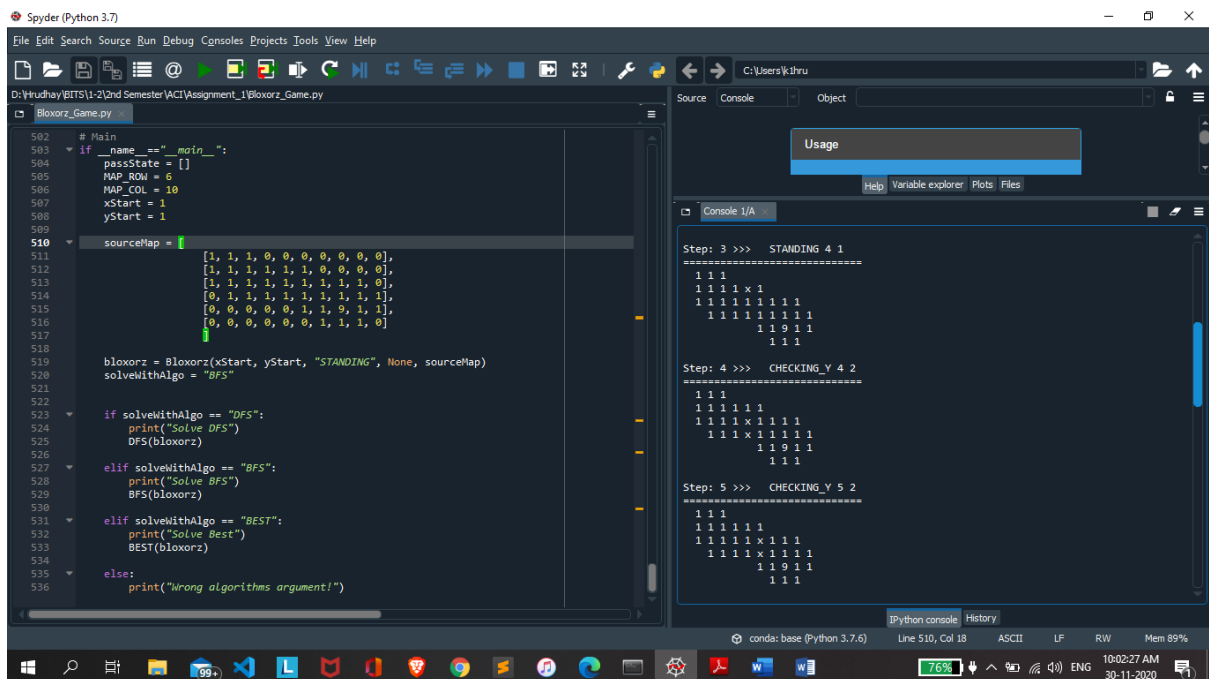


```
502 # Main
503 if __name__ == "__main__":
504     passState = []
505     MAP_ROW = 6
506     MAP_COL = 10
507     xStart = 1
508     yStart = 1
509
510     sourceMap = [
511         [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
512         [1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
513         [1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
514         [0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
515         [0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
516         [0, 0, 0, 0, 0, 0, 1, 1, 1, 0]
517     ]
518
519     bloxorz = Bloxorz(xStart, yStart, "STANDING", None, sourceMap)
520     solveWithAlgo = "BFS"
521
522     if solveWithAlgo == "DFS":
523         print("Solve DFS")
524         DFS(bloxorz)
525     elif solveWithAlgo == "BFS":
526         print("Solve BFS")
527         BFS(bloxorz)
528     elif solveWithAlgo == "BEST":
529         print("Solve Best")
530         BEST(bloxorz)
531     else:
532         print("Wrong algorithms argument!")
```

Console I/A

```
In [20]: runcell(0, 'D:/Hrudhay/BITS/1-2/2nd Semester/ACI/Assignment_1/
Bloxorz_Game.py')
Solve BFS

THIS IS SUCCESS ROAD
=====
Step: 1 >>> STANDING 1 1
=====
1 1 1
1 x 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 9 1 1
1 1 1
Step: 2 >>> CHECKING_X 2 1
=====
1 1 1
1 1 x x 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 9 1 1
1 1 1
Step: 3 >>> STANDING 4 1
```



```
502 # Main
503 if __name__ == "__main__":
504     passState = []
505     MAP_ROW = 6
506     MAP_COL = 10
507     xStart = 1
508     yStart = 1
509
510     sourceMap = [
511         [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
512         [1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
513         [1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
514         [0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
515         [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
516         [0, 0, 0, 0, 0, 1, 1, 1, 1, 0]
517     ]
518
519     bloxorz = Bloxorz(xStart, yStart, "STANDING", None, sourceMap)
520     solveWithAlgo = "BFS"
521
522     if solveWithAlgo == "DFS":
523         print("Solve DFS")
524         DFS(bloxorz)
525     elif solveWithAlgo == "BFS":
526         print("Solve BFS")
527         BFS(bloxorz)
528     elif solveWithAlgo == "BEST":
529         print("Solve Best")
530         BEST(bloxorz)
531     else:
532         print("Wrong algorithms argument!")
```

Console I/A

```
Step: 3 >>> STANDING 4 1
=====
1 1 1
1 1 1 1 x 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 9 1 1
1 1 1
Step: 4 >>> CHECKING_Y 4 2
=====
1 1 1
1 1 1 1 1
1 1 1 1 x 1 1 1
1 1 1 x 1 1 1 1
1 1 9 1 1
1 1 1
Step: 5 >>> CHECKING_Y 5 2
=====
1 1 1
1 1 1 1 1
1 1 1 1 x 1 1 1
1 1 1 1 x 1 1 1
1 1 9 1 1
1 1 1
```

