**AI project**

**name: Suliman hassan aljarbua**

**id: 435102530**

**Introduction:**

new problem is a hard one in implementation but theoretically it's really easy the project says solve the maze problem by the 3 algorithm BFS, HIL-CLIMBING, A\*

and there is other problem which is the battery the robot can only move "n+m" moves before he die so there is a charge station he can charge his self if he go to the charge station and do RECHARGE action, that's it I think.

**algorithms:**

* **I used simple heuristics the treasure location minus robot location that’s it.**

I implemented three different algorithms ,A\*,HC , BFS

in **A\*** there is a fringe and closed nodes which is the nodes that already explored and the fringe uses priority queue in which the expanded nodes goes in by there (cost + Manhattan distance) and continue by doing this until he finds a goal or fails.

in **BFS** it the same as A\* but rather than adding the node in the fringe by their priority it adds it in the end of the fringe.

in **HILL-CLIMBING** there is no fringe so it’s much faster It takes the nodes by there heuristic and delete every node in the same depth and so on until it finds the goal or falls in local maxima or global maxima, etc.

**Modeling:**

**you may find a differ between this modeling and the code implementation so please if it's gonna be a problem call me to tell you my excuse.**

**States:**

**char maze[][]**

**int robX**

**int robY**

**boolean onHole**

**boolean onT**

**boolean onT\_H**

**Boolean onT\_H\_C**

**Boolean onC**

**Int battery**

**initial state:**

**maze[6][6]**

**battery = 12**

**robX = 0**

**robY = 0**

**onC = false**

**maze[robX][robY] = 'R'**

**onHole = false**

**onT = false**

**onT\_H = false**

**onT\_H\_C = false**

**maze[1][0] = 'B'**

**\*Rest of the blocks are the same but the position.**

**maze[2][0] = 'H'**

**\*Rest of the Holes are the same but the position.**

**maze[5][5] = 'T'**

**maze[0][5] = 'C'**

**maze[5][2] = 'C'**

**the rest = ' '**

**actions:**

**moveU, moveD, moveR, moveL, recharge**

**transition model:**

**moveU:**

**pre=condition(battery != 0)**

**result(**

**pre-condition(robY > 0 && maze[robX][robY-1] != 'B' && onHole == false && onT\_H == false && onT\_H\_C = false)**

**result(if(onT&& maze[robX][robY-1] = ' '){**

**maze[robX][robY--]= 'T', onT = false,**

**maze[robX][robY] = 'R'}}**

**else if(onT&& maze[robX][robY-1] = 'H'){**

**maze[robX][robY--]= 'T', onT = false,**

**onHole = true, maze[robX][robY] = 'X'}}**

**else if(onT&& maze[robX][robY-1] = 'Y'){**

**maze[robX][robY--]= 'T', onT\_H = true,**

**onHole = true, maze[robX][robY] = 'Z'}}**

**else if(maze[robX][robY-1] = ' '){**

**maze[robX][robY--]= ' ',**

**maze[robX][robY] = 'R'}}**

**else if(maze[robX][robY-1] = 'H'){**

**maze[robX][robY--]= ' ', onHole = true,**

**maze[robX][robY] = 'X'}}**

**else if(maze[robX][robY-1] = 'Y'){**

**maze[robX][robY--]= ' ', onT\_H = true,**

**maze[robX][robY] = 'Z']}**

**else if(onT && maze[robX][robY-1] == ‘C’){**

**maze[robX][robY--] == ‘F’ && onT\_C = true)**

**else if (maze[robX][robY-1] == ‘C’ && maze[robX][robY] == ‘ ‘){**

**maze [robx][robY--] = ‘D’ && onC = true, maze [robx][robY] = ‘ ’})**

**battery = battery -1))**

**moveD:**

**pre=condition(battery != 0)**

**result(**

**pre-condition(robY < 5 && maze[robX][robY+1] != 'B' && onHole == false && onT\_H == false)**

**result(if(onT&& maze[robX][robY+1] = ' '){**

**maze[robX][robY++]= 'T', onT = false,**

**maze[robX][robY] = 'R'}}**

**else if(onT&& maze[robX][robY+1] = 'H'){**

**maze[robX][robY++]= 'T', onT = false,**

**onHole = true, maze[robX][robY] = 'X'}}**

**elseif(onT&& maze[robX][robY+1] = 'Y'){**

**maze[robX][robY++]= 'T', onT\_H = true,**

**onHole = true, maze[robX][robY] = 'Z'}}**

**else if(maze[robX][robY+1] = ' '){**

**maze[robX][robY++]= ' ',**

**maze[robX][robY] = 'R'}}**

**else if(maze[robX][robY+1] = 'H'){**

**maze[robX][robY++]= ' ', onHole = true,**

**maze[robX][robY] = 'X'}}**

**else if(maze[robX][robY+1] = 'Y'){**

**maze[robX][robY++]= ' ', onT\_H = true,**

**maze[robX][robY] = 'Z'} else if(onT && maze[robX][robY+1] == ‘C’){**

**maze[robX][robY++] == ‘F’ && onT\_C = true)**

**else if (maze[robX][robY+1] == ‘C’ && maze[robX][robY] == ‘ ‘){**

**maze [robx][robY++] = ‘D’ && onC = true, maze [robx][robY] = ‘ ’})**

**battery = battery -1)**

**moveW:**

**pre=condition(battery != 0)**

**result(**

**pre-condition(robX > 0 && maze[robX-1][robY] != 'B' && onHole == false && onT\_H == false)**

**result(if(onT&& maze[robX-1][robY] = ' '){**

**maze[robX-1][robY]= 'T', onT = false,**

**maze[robX][robY] = 'R'}}**

**else if(onT&& maze[robX-1][robY] = 'H'){**

**maze[robX--][robY]= 'T', onT = false,**

**onHole = true, maze[robX][robY] = 'X'}}**

**elseif(onT&& maze[robX-1][robY] = 'Y'){**

**maze[robX--][robY]= 'T', onT\_H = true,**

**onHole = true, maze[robX][robY] = 'Z'}}**

**else if(maze[robX-1][robY] = ' '){**

**maze[robX--][robY]= ' ',**

**maze[robX][robY] = 'R'}}**

**else if(maze[robX-1][robY] = 'H'){**

**maze[robX--][robY]= ' ', onHole = true,**

**maze[robX][robY] = 'X'}}**

**else if(maze[robX-1][robY] = 'Y'){**

**maze[robX--][robY]= ' ', onT\_H = true,**

**maze[robX][robY] = 'Z'}**

**else if(onT && maze[robX-1][robY] == ‘C’){**

**maze[robX--][robY] == ‘F’ && onT\_C = true)**

**else if (maze[robX-1][robY] == ‘C’ && maze[robX][robY] == ‘ ‘){**

**maze [robx--][robY] = ‘D’ && onC = true, maze [robx][robY] = ‘ ’})**

**battery = battery -1)**

**moveE:**

**pre=condition(battery != 0)**

**result(**

**pre-condition(robX < 5 && maze[robX+1][robY] != 'B' && onHole == false && onT\_H == false)**

**result(if(onT&& maze[robX+1][robY] = ' '){**

**maze[robX++][robY]= 'T', onT = false,**

**maze[robX][robY] = 'R'}}**

**else if(onT&& maze[robX+1][robY] = 'H'){**

**maze[robX++][robY]= 'T', onT = false,**

**onHole = true, maze[robX][robY] = 'X'}}**

**elseif(onT&& maze[robX+1][robY] = 'Y'){**

**maze[robX++][robY]= 'T', onT\_H = true,**

**onHole = true, maze[robX][robY] = 'Z'}}**

**else if(maze[robX+1][robY] = ' '){**

**maze[robX++][robY]= ' ',**

**maze[robX][robY] = 'R'}}**

**else if(maze[robX+1][robY] = 'H'){**

**maze[robX++][robY]= ' ', onHole = true,**

**maze[robX][robY] = 'X'}}**

**else if(maze[robX+1][robY] = 'Y'){**

**maze[robX++][robY]= ' ', onT\_H = true,**

**maze[robX][robY] = 'Z']}**

**else if(onT && maze[robX+1][robY] == ‘C’){**

**maze[robX++][robY] == ‘F’ && onT\_C = true)**

**else if (maze[robX+1][robY] == ‘C’ && maze[robX][robY] == ‘ ‘){**

**maze [robx++][robY] = ‘D’ && onC = true, maze [robx][robY] = ‘ ’})**

**battery = battery -1)**

**Recharge:**

**Pre-condition(onC == true || onT\_C == true || onT\_C\_H == true)**

**Result(battery = 6+6)**

**path cost:**

**= 1**

**goal test:**

**onT == true or onT\_H == true or onT\_C == true or onT\_C\_H == true**

**COMPARSION:**

For 25\*25 map

|  |  |  |  |
| --- | --- | --- | --- |
| **TITLE** | **BFS** | **A\*** | **HILL-CLIMBING** |
| **Run time** | 1400 ms | 473 ms | 14 ms |
| **#nodes explored** | 5140 | 2875 | 23 |
| **Solution cost** | 67 | 67 | No solution |

For 6\*6 map

|  |  |  |  |
| --- | --- | --- | --- |
| **TITLE** | **BFS** | **A\*** | **HILL-CLIMBING** |
| **Run time** | 15 ms | 11 ms | 4 ms |
| **#nodes explored** | 175 | 88 | 8 |
| **Solution cost** | 18 | 18 | No solution |

**Summary:**

As I mentioned before I used three different algorithm in BFS it’s kind of good but the problem is it’s not that speed and it will took a lot of memory in A\* it will explore less nodes and will find the same solution with the same cost, so it’s very good, in hill-climbing none of the time it found the solution but as someone mentioned in the real world they use HC as part of the solution, so I guess it’s not that necessary to find a solution with it