



MARMARA UNIVERSITY

CSE 4082 – PROJECT 1

Furkan Kuse 150117041

Zeynep Alici 150119517

CONTENTS

Section 1: The Output of the Example Maze.....2

Section 2: The Output of the Maze We Designed.....3

Section 3: Description of the Project.....4

Section 1: The Output of the Example Maze

	1	2	3	4	5	6	7	8
1								
2				T				
3		S				T	G	
4								
5								
6							G	
7						T		T
8					G			G

a. Depth First Search

Cost of the founded solution: 29

The number of expanded nodes: 32

The maximum size of the frontier: 5

The maximum size of the explored set: 32

Solution path: (2,3)-(1,3)-(1,2)-(1,1)-(2,1)-(2,2)-(3,2)-(4,2)-(4,3)-(5,3)-(6,3)-(7,3)

b. Breadth First Search

Cost of the founded solution: 29

The number of expanded nodes: 37

The maximum size of the frontier: 7

The maximum size of the explored set: 37

Solution path: (2,3)-(1,3)-(1,2)-(1,1)-(2,1)-(2,2)-(3,2)-(4,2)-(4,3)-(5,3)-(6,3)-(7,3)

c. Iterative Deepening

Cost of the founded solution: 29

The number of expanded nodes: 528

The maximum size of the frontier: 5

The maximum size of the explored set: 32

Solution path: (2,3)-(1,3)-(1,2)-(1,1)-(2,1)-(2,2)-(3,2)-(4,2)-(4,3)-(5,3)-(6,3)-(7,3)

d. Uniform Cost Search

Cost of the founded solution: 18

The number of expanded nodes: 46

The maximum size of the frontier: 7

The maximum size of the explored set: 46

Solution path: (2,3)-(1,3)-(1,2)-(1,1)-(2,1)-(2,2)-(3,2)-(3,1)-(4,1)-(5,1)-(6,1)-(7,1)-(8,1)-(8,2)-(8,3)-(8,4)-(8,5)-(8,6)-(7,6)

e. Greedy Best First Search

Cost of the founded solution: 29

The number of expanded nodes: 32

The maximum size of the frontier: 6

The maximum size of the explored set: 32

Solution path: (2,3)-(1,3)-(1,2)-(1,1)-(2,1)-(2,2)-(3,2)-(4,2)-(4,3)-(5,3)-(6,3)-(7,3)

f. A* Heuristic Search

Cost of the founded solution: 18

The number of expanded nodes: 45

The maximum size of the frontier: 6

The maximum size of the explored set: 45

Solution path: (2,3)-(1,3)-(1,2)-(1,1)-(2,1)-(2,2)-(3,2)-(3,1)-(4,1)-(5,1)-(6,1)-(7,1)-(8,1)-(8,2)-(8,3)-(8,4)-(8,5)-(8,6)-(7,6)

Section 2: The Output of the Maze We Designed

	1	2	3	4	5
1	G				
2			T		
3			S		
4					
5			T	T	G
6		G			

a. Depth First Search

Cost of the founded solution: 4
 The number of expanded nodes: 4
 The maximum size of the frontier: 7
 The maximum size of the explored set: 4
 Solution path: (3,3)-(4,3)-(4,4)-(5,4)-(5,5)

b. Breadth First Search

Cost of the founded solution: 4
 The number of expanded nodes: 18
 The maximum size of the frontier: 9
 The maximum size of the explored set: 18
 Solution path: (3,3)-(4,3)-(4,4)-(5,4)-(5,5)

c. Iterative Deepening

Cost of the founded solution: 4
 The number of expanded nodes: 10
 The maximum size of the frontier: 7
 The maximum size of the explored set: 4
 Solution path: (3,3)-(4,3)-(4,4)-(5,4)-(5,5)

d. Uniform Cost Search

Cost of the founded solution: 4
 The number of expanded nodes: 16
 The maximum size of the frontier: 11
 The maximum size of the explored set: 16
 Solution path: (3,3)-(2,3)-(2,2)-(2,1)-(1,1)

e. Greedy Best First Search

Cost of the founded solution: 4
 The number of expanded nodes: 4
 The maximum size of the frontier: 7
 The maximum size of the explored set: 4
 Solution path: (3,3)-(4,3)-(4,4)-(5,4)-(5,5)

f. A* Heuristic Search

Cost of the founded solution: 4
 The number of expanded nodes: 9
 The maximum size of the frontier: 9
 The maximum size of the explored set: 9
 Solution path: (3,3)-(4,3)-(4,4)-(5,4)-(5,5)

Section 3: Description of the Project

1. **GraphSearch:** The graph search algorithm is implemented in this class.

a. **Features:**

strategy: It is an instance of one of the classes of search algorithms defined below.

grid: Two dimensional array for input maze

cost: Integer value for final cost of the solution

exploredSet: Array to stored nodes of the explored set

lastNode: An instance of Node class to store last visited node

goalNodes: : Array to store goal nodes

maxDepth: Integer value to define maximum depth for iterative deepening search

currentDepth: Integer value to check current depth for iterative deepening search

IDS_exploredSet: Explored set of iterative deepening search

maxLenOfExploredSet: Maximum length of the explored set

b. **Functions**

expandNode(node): It returns accessible nodes from the input node.

checkInNotFrontierOrExploredSet(self, nextNode): It returns true if input node not in explored set or frontier; otherwise returns false.

search(): Graph search algorithm is implemented on this function.

printPath(): It prints the solution path.

printExploredSet(): It prints explored set.

printIterativeDeepeningExploredSet(): It prints explored set for iterative deepening.

2. **Node:**

a. **Features:**

status: It can be N for normal, G for goal, T for trap nodes.

eastWall, westWall, northWall, southWall: Boolean values indicates whether there is a wall east, west, north and south of the node or not

verticalIndex: Vertical index of node

horizontalIndex: Horizontal index of node

cost: Cost of the node from initial state

successor: Last node on the solution path before coming current node

heuristicCost: Heuristic function value from the node to goal state

3. **DFS:** It is a strategy class for depth-first search.

a. **Features**

frontier: It is LIFO queue for graph search.

maxLenFrontier: It is an integer value to store maximum length of the frontier.

b. **Functions**

operate(): It pops a node from the frontier as regarding the type of the frontier

append(): It push a node to frontier

getLengthOfTheFrontier(): It returns the length of the frontier.

getAllFrontier(): It returns the frontier as a list.

4. **BFS:** It is a strategy class for best-first search.

a. **Features**

frontier: It is FIFO queue for graph search.

maxLenFrontier: It is an integer value to store maximum length of the frontier.

b. **Functions**

operate(): It pops a node from the frontier as regarding the type of the frontier

append(): It push a node to frontier

getLengthOfTheFrontier(): It returns the length of the frontier.

getAllFrontier(): It returns the frontier as a list.

5. **IterativeDeepeningSearch:** It is a strategy class for iterative deepening search.

a. **Features**

frontier: It is LIFO queue for graph search.

maxLenFrontier: It is an integer value to store maximum length of the frontier.

b. **Functions**

operate(): It pops a node from the frontier as regarding the type of the frontier

append(): It push a node to frontier

getLengthOfTheFrontier(): It returns the length of the frontier.

getAllFrontier(): It returns the frontier as a list.

6. **UniformCostSearch:** It is a strategy class for uniform cost search.

a. **Features**

frontier: It is priority queue for graph search.

maxLenFrontier: It is an integer value to store maximum length of the frontier.

b. Functions

operate(): It pops a node from the frontier as regarding the type of the frontier

append(): It push a node to frontier

getLengthOfTheFrontier(): It returns the length of the frontier.

getAllFrontier(): It returns the frontier as a list.

7. GreedyBestFirstSearch: It is a strategy class for greedy best search.

a. Features

frontier: It is priority queue for graph search.

maxLenFrontier: It is an integer value to store maximum length of the frontier.

b. Functions

calculateHeuristicValues(grid, goalSquares): It uses Manhattan distance as heuristic function and calculates and updates heuristic costs of the nodes.

operate(): It pops a node from the frontier as regarding the type of the frontier

append(): It push a node to frontier

getLengthOfTheFrontier(): It returns the length of the frontier.

getAllFrontier(): It returns the frontier as a list.

8. A_StarSearch: It is a strategy class for A* search.

a. Features

frontier: It is priority queue for graph search.

maxLenFrontier: It is an integer value to store maximum length of the frontier.

b. Functions

calculateHeuristicValues(grid, goalSquares): It uses Manhattan distance as heuristic function and calculates and updates heuristic costs of the nodes.

operate(): It pops a node from the frontier as regarding the type of the frontier

append(): It push a node to frontier

getLengthOfTheFrontier(): It returns the length of the frontier.

getAllFrontier(): It returns the frontier as a list.