Pacman Game

Step 1 - Depth-First Search

Intelligent Systems

Master in Software Engineering

Manuel Cura (76546) | Carolina Albuquerque (80038)

deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

Context

The aim of the first challenge of the Intelligent Systems course is the implementation of depth-first search algorithm to allow the pacman agent to find paths across the game board to achieve the goals.

In this work, the strategies used in the implementation of the algorithm will be addressed, as well as the results obtained for the different boards of the pacman game. At the end, the costs of the obtained paths will be analyzed to verify if the path returned by the algorithm is the one with the lowest cost.

Depth-First Search

Depth-First search is an algorithm for searching tree or graph structures where it starts at the root node and explores as far as possible along each branch before backtracking.

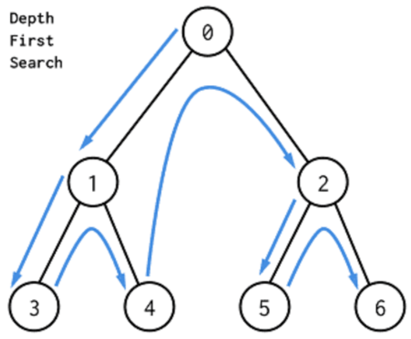


Figure 1 - *Demonstration of depth-first algorithm*

Implementation

The algorithm was implemented using the graph search approach in order to avoid expanding already explored nodes, optimizing the agent. All explored nodes are stored in a set and for each successor node of the current node it is checked if this was already explored. With this approach, contrary to the tree search approach, only non explored nodes will be considered and pushed to data structure avoiding the rewriting of actions for an already explored node.

It expected a list of actions as a result of algorithm search. During the search, if the goal is achieved, the correspondent list of actions to achieve it must be returned. Otherwise, an empty list of actions is returned to the agent.

Data Structures

As data structure to store all the agent knowledge was used a stack due to depth-first search implementation. As this algorithm acts in the base of the last element visited is the first to be analyzed, and as stack implements the last-in first-out (LIFO) approach, this data structure is the most suitable to the problem.

On this structure are stored all the discovered nodes that could be selected on the next move if the solution was not found yet. Also, it is important to save the corresponding actions that the agent needs to take to achieve the end goal, and these actions will be returned after the algorithm is completed.

For explored nodes structure, a set is used in order to not store duplicate nodes to this structure and optimize the computational effort.

Code

```
def depthFirstSearch(problem):
    startNode = problem.getStartState()
    explored = set() # set of explored nodes

    structure = util.Stack() # stack because use LIFO implementation

    # (Node, list of actions until to achieve the node)
    structure.push((startNode, list()))

    while not structure.isEmpty():
        currentNode, actions = structure.pop() # returns the last element of list

        if problem.isGoalState(currentNode): # returns actions if is the goal
            return actions

        if currentNode not in explored:
            explored.add(currentNode)

            for successorNode, action, cost in problem.getSuccessors(currentNode):
                if successorNode not in explored:
                    # copy actions list and append the new action to achieve the goal
                    nextAction = actions.copy()
                    nextAction.append(action)
                    # add the list of actions to achieve the current node
                    structure.push((successorNode, nextAction))
    return list()
```

The entire code can be checked on GitHub repository.

Results Analysis

Testing the approach implemented for mediumMaze board, the results obtained are considered positives and the agent performs the algorithm correctly according to the visual output shown in the game window.

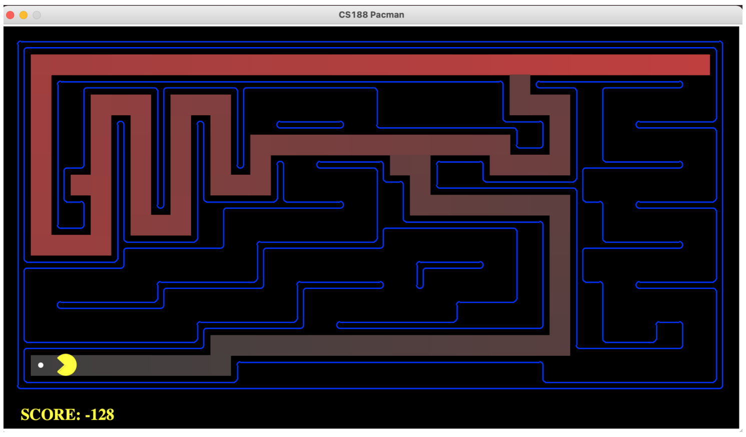
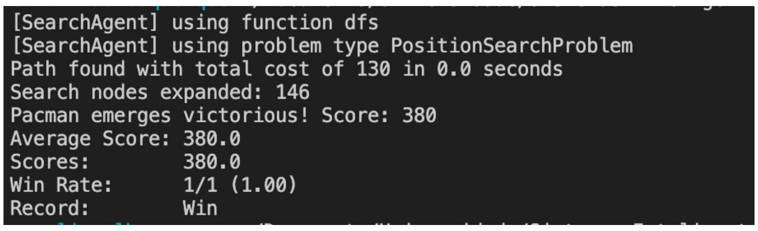


Figure 2 - *Expanded nodes of depth-first search algorithm for mediumMaze*



```
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure 3 - *Results of depth-first search algorithm for mediumMaze*

The implementation was tested also using the bigMaze board in order to analyse the agent's behaviour varying the map.

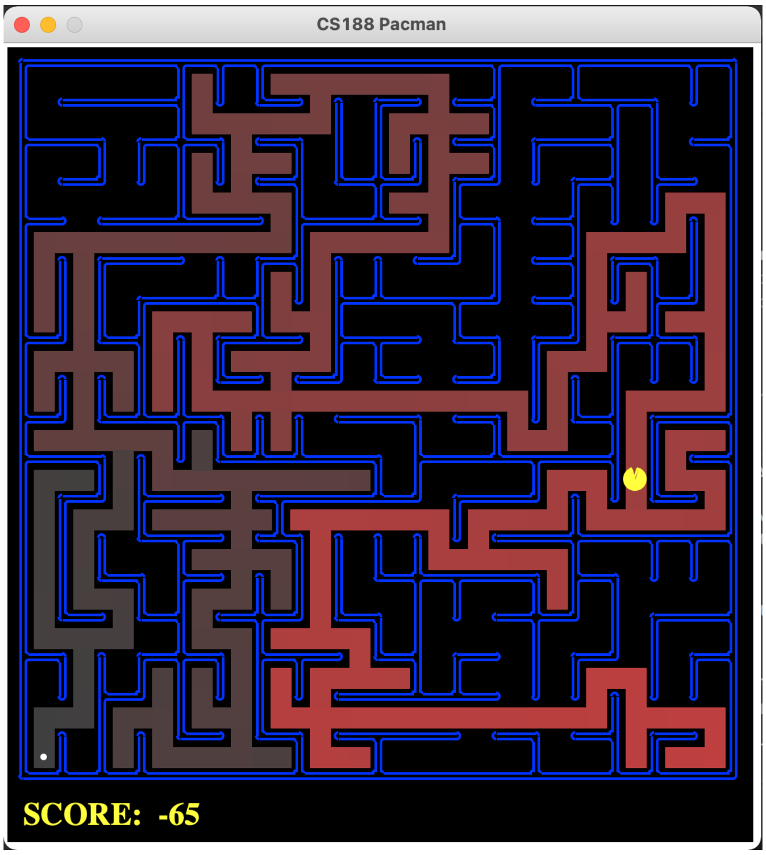
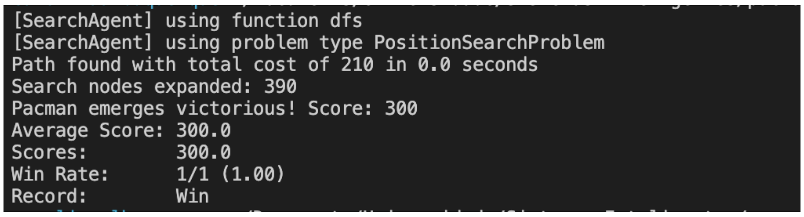


Figure 4 - *Expanded nodes of depth-first search algorithm for bigMaze*



```
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 390
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure 5 - *Results of depth-first search algorithm for bigMaze*

The output results shows pacman agent managed the path successfully using the implemented algorithm.

In order to compare the costs of the solutions and the nodes explored for each map, the following table was constructed.

Map	Cost	Nodes Expanded	Time
testMaze	7	7	0.0 s
tinyMaze	10	15	0.0 s
smallMaze	49	59	0.0 s
mediumMaze	130	146	0.0 s
bigMaze	210	390	0.0 s
openMaze	298	576	0.0 s
contoursMaze	85	85	0.0 s

Table 1 - *Comparison of cost and nodes expanded to find the path to achieve the pacman agent's goal*

This algorithm does not always present solutions that are optimal, since it only expands nodes, and it selects the successor nodes as possible nodes to visit. It might find the best solution, but it does not take in consideration any costs of the decisions or previous learning. If the algorithm runs ten times, the same result will always be obtained. Finding the best path with this algorithm is a matter of luck.

Least Cost Solution

Map	Cost	Least Cost Solution
testMaze	7	Yes
tinyMaze	10	No
smallMaze	49	No
mediumMaze	130	No
bigMaze	210	Yes
openMaze	298	No
contoursMaze	85	No

Table 2 - *Comparison of cost returned by search algorithm and least cost solution*

Analysing Table 2, it appears that in most maps the algorithm does not return the least cost solution. The algorithm acts on the basis of exploring nodes in depth with a principle of last-in first-out analysis. Sometimes, the successor nodes of the last node added to the knowledge base do not contain the final solution or contain a more expensive path to achieve it. For example, considering Figure 1, if the final goal is node 2 or any of its successors, the algorithm when using in-depth search is expanding nodes that may be considered irrelevant to the final solution.

Conclusions

First, it is concluded that graph search approach is better than tree search approach because tree search does not consider previously explored nodes and does not optimize the search.

Secondly, it is not correct to say that the Depth-First search algorithm presents worse results compared to other search algorithms not yet implemented. Whether an algorithm returns a path other than the lowest cost depends on the type of game map the agent is deciphering. There may be maps with characteristics in which in-depth research is an asset in terms of time or the search for less costly solutions, as there may be maps in which this does not happen. However, with the tests implemented, it was noted that the algorithm did not always return the lowest cost solution.