Pacman Game

Step 4 and 5 - A* Search and Finding All Corners Problem

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 A* search to allow the pacman agent to find paths across the game board to achieve the goals. In addition, an implementation was required to solve the finding all corners problem where the agent needs to find all four maze corners and visit it whether or not they have food.

A* Search

A* is a popular algorithm for searching graphs and finding paths. This algorithm is considered one of the best because it is fairly flexible and can be used in a wide range of contexts. A* uses an evaluation function to estimate the total cost of the path to achieve the goal. This function combines the cost so far to reach a successor goal, *g(n)*, with a heuristic function to estimated cost from the current goal to a successor goal, *h(n)*.

$$f(n) = g(n) + h(n)$$

A* achieves better performance by using heuristics to estimate the cost from the current goal to a successor (or the final) goal.

Implementation

A* Search Algorithm

Due to the implementation similarities between the A* search algorithm and the uniform-cost search, it was implemented as a generic function reused for both algorithms. Both algorithms use as data structure a PriorityQueue, only differing in the priority calculation method.

This function receives different heuristics by parameters in accordance with the search algorithm used. As uniform-cost search only considers the cost of actions returned for a successor node, the heuristic used for this algorithm is the nullHeuristic returning 0 as cost. On the other hand, the A* search algorithm apart from the cost of actions also considers a heuristic to calculate the path cost to a successor node. The heuristic used for these calculations should be passed in the start game command as an argument, however if a heuristic is not passed, A* uses the nullHeuristic by default to guarantee the game does not crash.

```
def genericPrioriryGraphSearch(problem, heuristic=nullHeuristic):
    startNode = problem.getStartState()
    explored = set() # set of explored nodes
    structure = util.PriorityQueue() # queue because use FIFO implementation

    # ((Node, list of actions until to achieve the node), cost)
    structure.push((startNode, list()), 0)

    if problem.isGoalState(startNode): # returns actions if start node is the goal
        return list()

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

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

            for successorNode, action, cost in problem.getSuccessors(currentNode):
                nextAction = actions.copy()
                nextAction.append(action)

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

                # get the cost of the actions' path
                pathCost = problem.getCostOfActions(nextAction)+heuristic(successorNode,
problem)
                # add the list of actions to achieve the current node
                structure.update((successorNode, nextAction), pathCost)

    return list()
```

```
def aStarSearch(problem, heuristic=nullHeuristic):
    return genericPrioriryGraphSearch(problem, heuristic)
```

```
def uniformCostSearch(problem):
    return genericPrioriryGraphSearch(problem, nullHeuristic)
```

Finding All Corners Problem

The corners problem aims to find paths through all four corners of the map. The objective is even if there is food or not in a specific corner, the game should find a solution to get there.

To achieve this, it was necessary to complete and implement three functions:

- **getStartState -** Returns the start state of the Pacman, and all the corners that it needs to visit;
- **isGoalState -** Returns success if the path for all the corners has been found;
- **getSuccessors -** Returns the successors, this includes their states, actions they require, and a cost of 1;

At the beginning, the getStartState gives the initial starting position of the Pacman, and also all the four corners, because none of them are already mapped.

On the getSuccessors function, it will remove from the corners to visit structure one of the corners, if that is the next successor, retrieving the information for the corners that are not visited yet.

Once all the corners have been mapped, it means that the corners to visit structure is empty, and the goal of the problem is solved, at that point the problem ends calling the isGoalState function.

```
def getStartState(self):
    state = (self.startingPosition, self.corners)
    return state

def isGoalState(self, state):
    if len(state[1]) == 0:
        return True
    return False

def getSuccessors(self, state):
    successors = []
    for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST,
Directions.WEST]:
        x, y = state[0]
        dx, dy = Actions.directionToVector(action)
        nextX, nextY = int(x + dx), int(y + dy)

        if not self.walls[nextX][nextY]:
            cornersToVisit = state[1]
            nextLocation = (nextX, nextY)
            if nextLocation in cornersToVisit:
                cornersToVisit = tuple(x for x in state[1] if x != nextLocation)

            nextState = (nextLocation, cornersToVisit)
            successors.append((nextState, action, 1))

    self._expanded += 1 # DO NOT CHANGE
    return successors
```

The entire code can be checked on GitHub repository.

Results Analysis

A* Search

A* algorithm was tested using the different heuristics already implemented in the codebase: nullHeuristic, manhattanHeuristic and euclideanHeuristic.

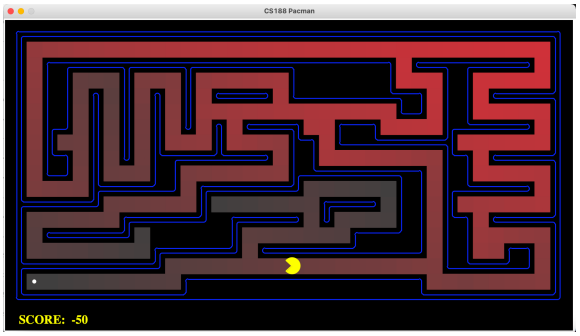

Figure 1 *- A* search algorithm using nullHeuristic for mediumMaze*

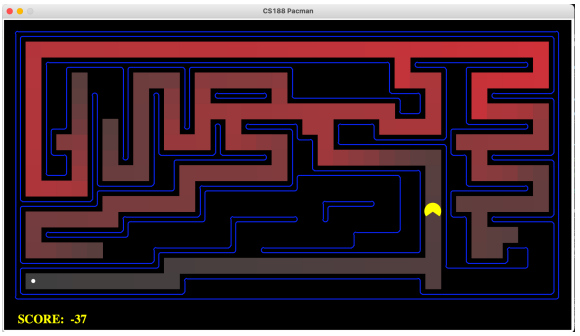

Figure 2 *- A* search algorithm using manhattanHeuristic for mediumMaze*

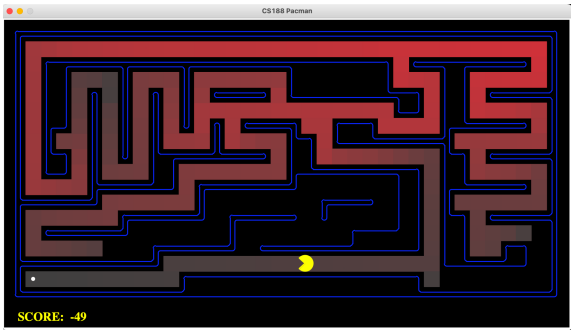

Figure 3 *- A* search algorithm using euclideanHeuristic for mediumMaze*

```
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 267
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure 4 *- Results of A* search using nullHeuristc for mediumMaze*

```
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 221
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure 5 *- Results of A* search using manhattanHeuristc for mediumMaze*

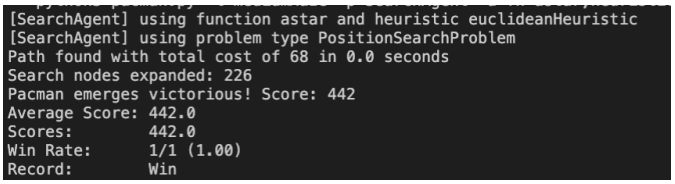

Figure 6 - *Results of A* search using euclideanHeuristc for mediumMaze*

Map	nullHeuristic			manhattanHeuristic			euclideanHeuristic		
	Cost	Expanded Nodes	Time	Cost	Expanded Nodes	Time	Cost	Expanded Nodes	Time
tinyMaze	8	15	0.0s	8	14	0.0s	8	12	0.0s
smallMaze	19	90	0.0s	19	53	0.0s	19	56	0.0s
mediumMaze	68	267	0.0s	68	221	0.0s	68	226	0.0s
bigMaze	210	617	0.2s	210	549	0.1s	210	557	0.2s
openMaze	54	679	0.1s	54	534	0.2s	54	549	0.1s
contoursMaze	13	165	0.0s	13	48	0.0s	13	59	0.0s

Table 1 - *Comparison of A* algorithm's heuristics used to achieve the pacman agent's goal*

Of all the heuristics analysed above, the Manhattan Heuristic is the best compared with others in terms of time and expanded nodes. Therefore, in order to establish a comparison between all search already implemented algorithms below, it was chosen the Manhattan Heuristic for this purpose.

Depth-First vs Breadth-First vs Uniform-Cost vs A* Searches

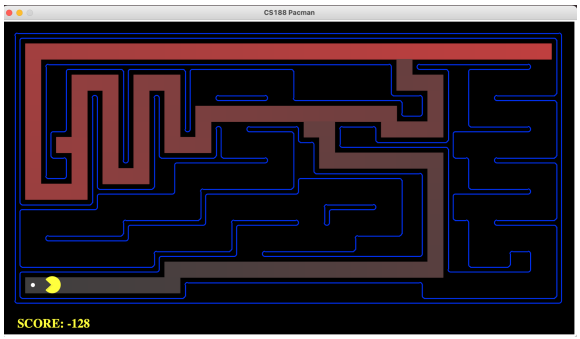

Figure 7 - *Depth-first algorithm for mediumMaze*

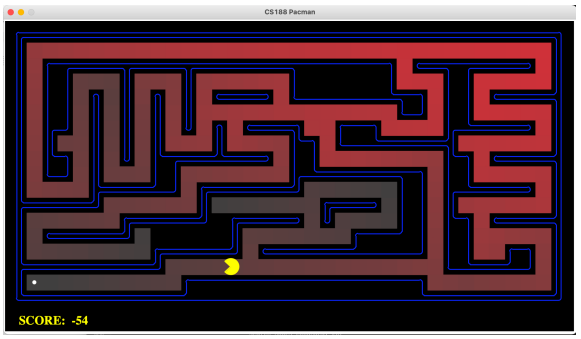

Figure 8 - *Breadth-first algorithm for mediumMaze*

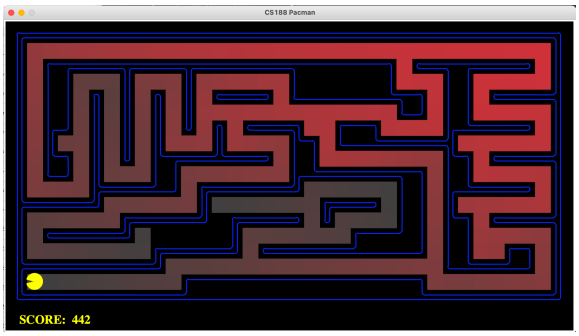

Figure 9 - *Uniform-cost algorithm for mediumMaze*

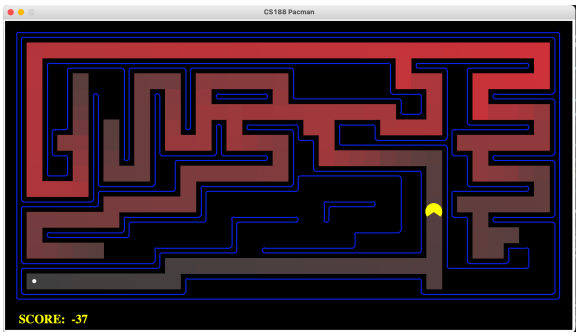

Figure 10 - *A* search algorithm using manhattanHeuristic for mediumMaze*

As seen in the previous Figures, all algorithms have different behaviours. As required in the problem's statement, it also compared all search algorithms for openMaze.

Figure 11 - *Depth-first algorithm in openMaze*

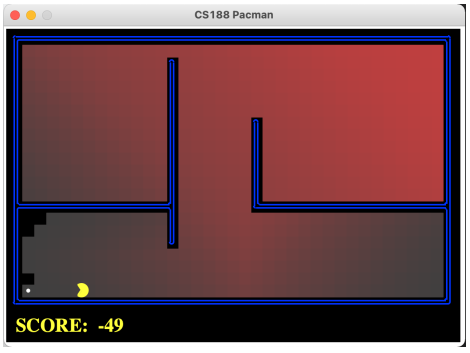

Figure 12 - *Breadth-first algorithm in openMaze*

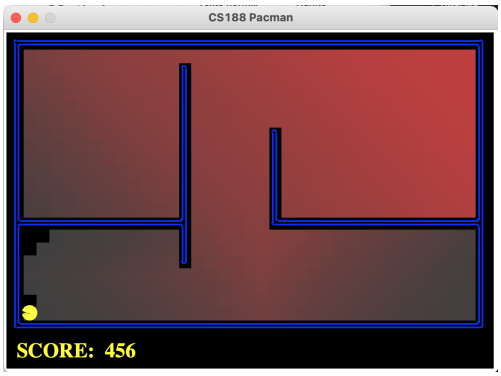

Figure 13 - *Uniform-cost algorithm in openMaze*

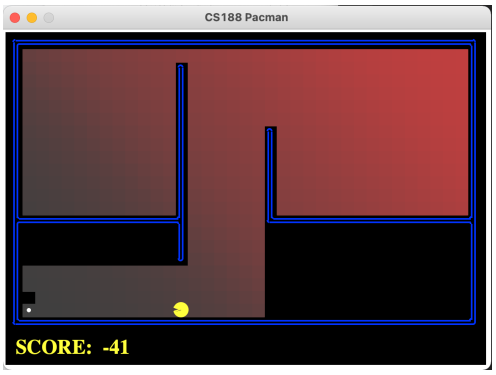

Figure 14 - *A* algorithm using manhattanHeuristic in openMaze*

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

Map	Depth-First	Breadth-First	Uniform-Cost	A* (manhattanHeuristic)
tinyMaze	10	8	8	8
smallMaze	49	19	19	19
mediumMaze	130	68	68	68
bigMaze	210	210	210	210
openMaze	298	54	54	54
contoursMaze	85	13	13	13

Table 2 - *Comparison of cost of the found path to achieve the pacman agent's goal*

Map	Depth-First	Breadth-First	Uniform-Cost	A* (manhattanHeuristic)
tinyMaze	15	15	15	14
smallMaze	59	90	90	53
mediumMaze	146	267	267	221
bigMaze	390	617	617	549
openMaze	576	679	679	534
contoursMaze	85	165	165	48

Table 3 - *Comparison of the number of nodes expanded to find the path to achieve the pacman agent's goal*

Map	Depth-First	Breadth-First	Uniform-Cost	A* (manhattanHeuristic)
tinyMaze	0.0s	0.0s	0.0s	0.0s
smallMaze	0.0s	0.0s	0.0s	0.0s
mediumMaze	0.0s	0.0s	0.0s	0.0s
bigMaze	0.0s	0.0s	0.1s	0.1s
openMaze	0.0s	0.0s	0.1s	0.2s
contoursMaze	0.0s	0.0s	0.0s	0.0s

Table 4 - *Comparison of the time spent to find the path to achieve the pacman agent's goal*

Analysing the previous presented results, it is clear that A* search algorithm behaves better than the others implemented. Although the returned total cost is equal to breadth-first and uniform-cost searches, A* using the Manhattan Heuristic expands fewer nodes until reaching the optimal solution. In terms of time spent, uniform-cost and A* algorithms require some research time for more complex maps.

Least Cost Solution

Map	Least Cost	Depth-First	Breadth-First	Uniform-Cost	A* (manhattanHeuristic)
tinyMaze	8	✗	✓	✓	✓
smallMaze	19	✗	✓	✓	✓
mediumMaze	68	✗	✓	✓	✓
bigMaze	210	✓	✓	✓	✓
openMaze	54	✗	✓	✓	✓
contoursMaze	13	✗	✓	✓	✓

Table 5 - *Algorithms that returns the least cost solution for each maze*

Analysing Table 5, depth-first search algorithm only returns one least cost solution (bigMaze) while breadth-first, uniform-cost and A* search often returns the least cost solution for each maze. However, and as seen before, A* algorithm presents a better performance compared to the others.

Finding All Corners Problem

The corners problem was tested using the breadth-first search algorithm for the three different corners maps: tinyCorners, mediumCorners, bigCorners and openMaze that contains food in one corner.

```
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 243
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:        512.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure 15 - *Results of Breadth-First search using the corners problem for tinyCorners*

```
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 1921
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure 16 - *Results of Breadth-First search using the corners problem for mediumCorners*

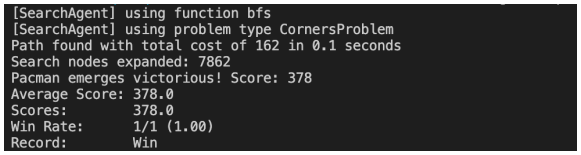

Figure 17 - *Results of Breadth-First search using the corners problem for bigCorners*

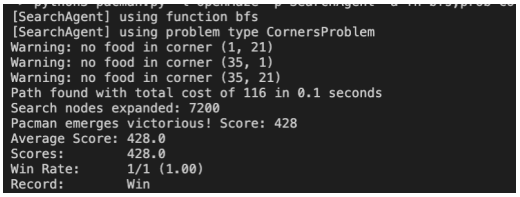

Figure 18 - *Results of Breadth-First search using the corners problem for openMaze*

Map	Cost	Expanded Nodes	Time
tinyCorners	28	243	0.0 s
mediumCorners	106	1921	0.0 s
bigCorners	162	7862	0.1 s
openMaze	116	7200	0.1 s

Table 6 - *Results of solving the Corners Problem using breadth-first search algorithm*

OpenMaze layout only has food in one of all four corners as seen in the results. The agent starts to visit the corners, however, when he eats the food in the third corner, he wins the game and never visits the last corner.

The breadth-first search algorithm presents in this problem always the same cost solution, being optimal since the graph presents the same cost in all the nodes.

Conclusions

After solving the proposed steps (step 4 and 5), it can be concluded that different search algorithms can perform better depending on the problem that they are solving. However, the A* search algorithm presents better results in every situation tested. This algorithm grants the least cost solution, like the uniform-cost algorithm, but it does that by expanding less nodes to solve the problem.

To compare the A* algorithm with the other algorithms previously implemented, we used by default the manhattan heuristic, since this one presented the best results on our tests.

It was also interesting to see that the breadth-first search algorithm can be used to solve different types of problems, for example the Corners Problem, proving that the implementation is generic and well optimized. Also, with the implementation of the Corners Problem, it was better understood the way that nodes work and how to get them.