Inteligência Artificial
T1 - Relatório

Professor: Eduardo Bezerra
Aluno: Lucas Lima da Cruz

- # Q1 - Encontrando comida usando DFS

Implementando o algoritmo de busca DFS no arquivo *search.py*:

```
def depthFirstSearch(problem):
    "*** YOUR CODE HERE ***"
    node = getStartNode(problem)
    frontier = util.Stack()
    frontier.push(node)
    explored = set()

    while not frontier.isEmpty():
        node = frontier.pop()

        if problem.isGoalState(node['STATE']):
          return getActionSequence(node)

        if node['STATE'] in explored:
          continue

        explored.add(node['STATE'])

        if problem.isGoalState(node['STATE']):
            return getActionSequence(node)

        for successor in problem.expand(node['STATE']):
            child_node = getChildNode(successor,node)
            frontier.push(child_node)

    return []
```

Executando o programa para os labirintos *tinyMaze*, *mediumMaze* e *bigMaze*, obtemos os seguintes resultados, que representam uma solução não ótima:

- ## tinyMaze

Input:

```
python3 pacman.py -l tinyMaze -p SearchAgent -a fn=dfs
```

Output:

```
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:        500.0
Win Rate:      1/1 (1.00)
Record:        Win
```

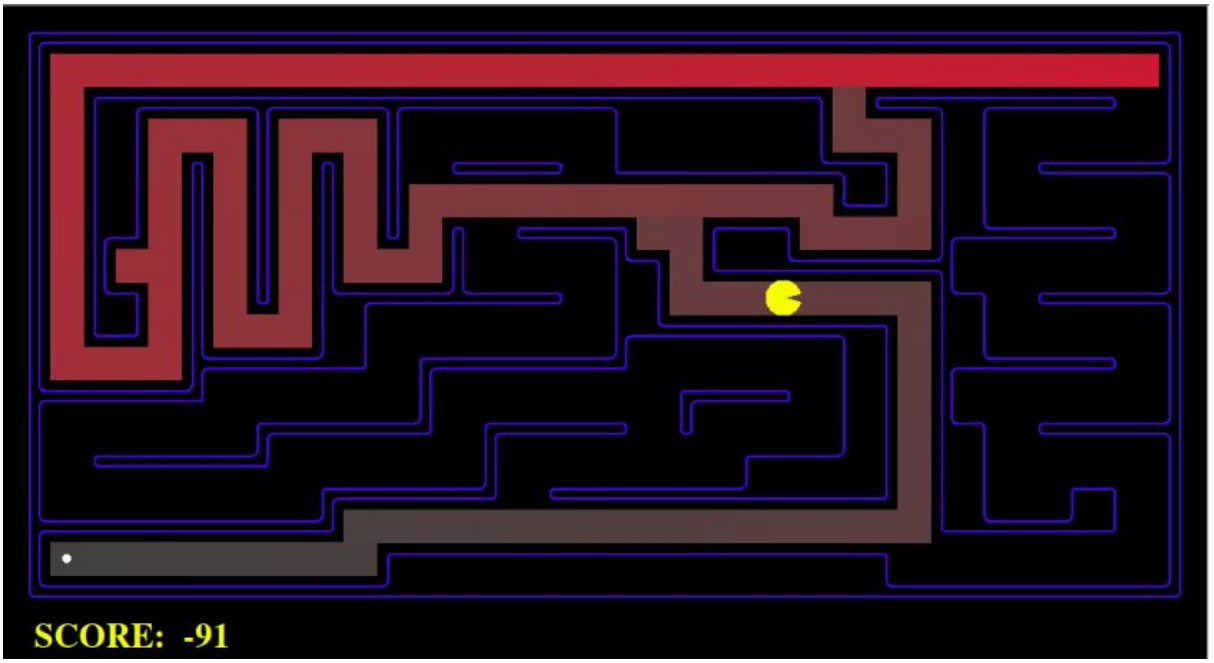
- **mediumMaze**

Input:

```
python3 pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
```

Output:

```
[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
```



- ## bigMaze

Input:

```
python3 pacman.py -l bigMaze -p SearchAgent -a fn=dfs
```

Output:

```
[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
```

● Q2 - BFS

A implementação do algoritmo de busca BFS difere do DFS apenas na estrutura de dados responsável pela retirada de folhas da árvore, neste caso:

```
frontier = util.Queue()
```

Implementando o algoritmo de busca BFS no arquivo *search.py* e executando o programa para os labirintos *mediumMaze* e *bigMaze*, obtemos os seguintes resultados, que representam uma solução ótima:

- ## mediumMaze

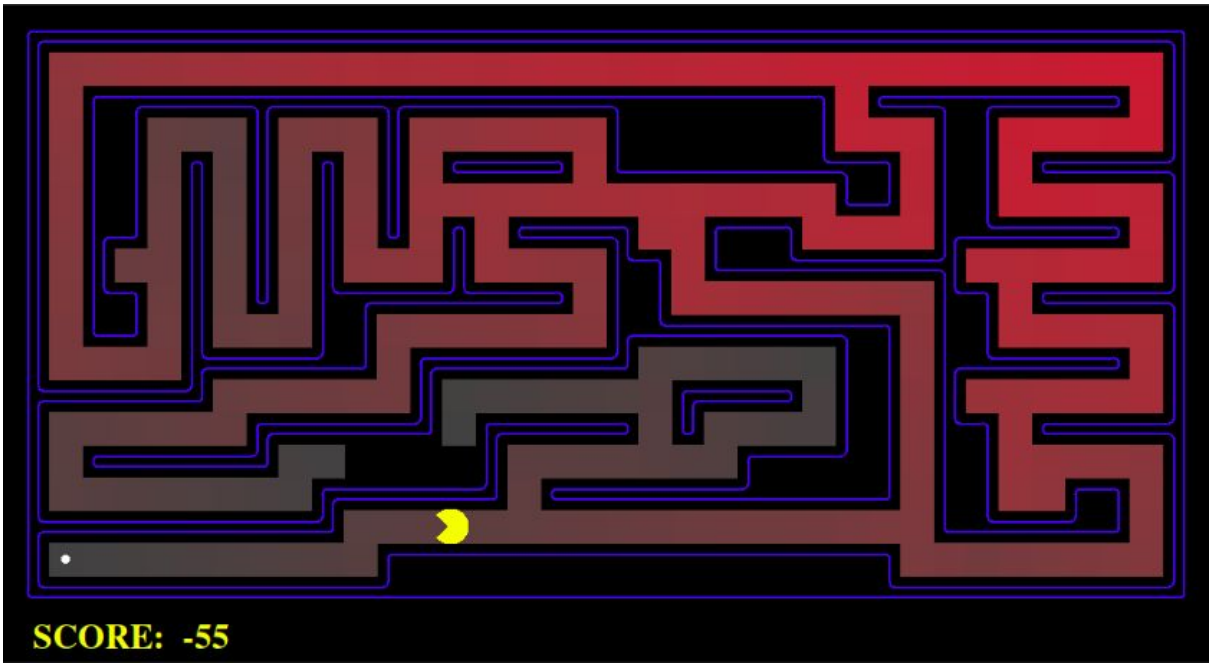
Input:

```
python3 pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
```

Output:

```
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
```

```
Win Rate:      1/1 (1.00)
Record:        Win
```



- **bigMaze**

  Input:

  ```
  python3 pacman.py -l bigMaze -p SearchAgent -a fn=bfs
  ```

  Output:

  ```
  [SearchAgent] using function bfs
  [SearchAgent] using problem type PositionSearchProblem
  Path found with total cost of 210 in 0.0 seconds
  Search nodes expanded: 620
  Pacman emerges victorious! Score: 300
  Average Score: 300.0
  Scores:        300.0
  Win Rate:      1/1 (1.00)
  Record:        Win
  ```

- **eightpuzzle.py**

  Para este problema, a busca por BFS levou 5 movimentos para chegar à solução.

  Input:

  ```
  python3 pacman.py -l bigMaze -p SearchAgent -a fn=bfs
  ```

Output:

Output do último estágio da solução

```
After 5 moves: left
-------------
|   | 1 | 2 |
-------------
| 3 | 4 | 5 |
-------------
| 6 | 7 | 8 |
-------------
```

● Q3 - A* Search

A implementação de A* Search difere das demais também no uso da estrutura de dados, neste caso uma fila com prioridade:

```
fn_total_cost_for_node = lambda a_node: a_node['PATH-COST'] +
heuristic(a_node['STATE'], problem=problem)

frontier=util.PriorityQueueWithFunction(fn_total_cost_for_node)
```

Executando o programa para os labirintos *tinyMaze*, *mediumMaze* e *bigMaze*, obtemos os seguintes resultados:

● bigMaze

Input:

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a
fn=astar,heuristic=manhattanHeuristic
```

Output:

```
SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
```

```
Record:       Win
```

- **openMaze**

Input:

```
python pacman.py -l openMaze -z .5 -p SearchAgent -a
fn=astar,heuristic=manhattanHeuristic
```

Output:

```
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 535
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:       456.0
Win Rate:      1/1 (1.00)
Record:       Win
```

● Q4 - Encontrando todos os cantos

Implementando o algoritmo do problema *CornersProblem* nas funções do arquivo searchAgent.py e executando para *tinyCorners* e *mediumCorners*, obtemos:

- **tinyCorners**

Input:

```
python3 pacman.py -l tinyCorners -p SearchAgent -a
fn=bfs,prob=CornersProblem
```

Output:

```
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:       512.0
Win Rate:      1/1 (1.00)
```

```
Record:       Win
```

- **mediumCorners**

Input:

```
python3 pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
```

Output:

```
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.1 seconds
Search nodes expanded: 1988
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:       434.0
Win Rate:      1/1 (1.00)
Record:       Win
```

Q5 - Heurística para o Problema dos Cantos

Implementando uma heurística do problema *CornersProblem* em *CornersHeuristic*:

```
def cornersHeuristic(state, problem):
    "*** YOUR CODE HERE ***"
    """return 0 # Default to trivial solution"""
    corners_with_pills = list(state[1])
    pacmanPos = state[0]
    pathLength = 0
    while corners_with_pills != []:
        for index, corner in enumerate(corners_with_pills):
            d = util.manhattanDistance(pacmanPos, corner)
        pathLength += d
        corners_with_pills.pop(index)
    return pathLength
```

A heurística é implementada considerando um labirinto sem obstáculos, o que faz com que o custo seja sempre menor que o custo efetivo de solução do problema. E é consistente pois o valor da função heurística diminui conforme os valores objetivos do Pacman são atingidos no labirinto.

Input:

```
python3 pacman.py -l mediumCorners -p SearchAgent -a
fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic
-z 0.5
```

Output:

```
[SearchAgent] using function aStarSearch and heuristic
cornersHeuristic
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 506
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores: 434.0
Win Rate: 1/1 (1.00)
Record: Win
```

- # Q7 - Busca subótima

Preenchendo a função *IsGoalState*() da classe *AnyFoodSearchProblem* de forma que *IsGoalState()* retorne *true* ou *false* dependendo da existência de comida na posição atual do pacman:

```
def isGoalState(self, state):
        """
        The state is Pacman's position. Fill this in with a goal
test that will complete the problem definition.
        """
        x,y = state

        "*** YOUR CODE HERE ***"
        return self.food[x][y]
```

Preenchendo a função *findPathToClosestDot*() da classe *ClosestSearchDotAgent* para utilização do algoritmo de busca BFS:

```
def findPathToClosestDot(self, gameState):
        """
        Returns a path (a list of actions) to the closest dot,
starting from
        gameState.
        """
        # Here are some useful elements of the startState
        startPosition = gameState.getPacmanPosition()
```

```
        food = gameState.getFood()
        walls = gameState.getWalls()
        problem = AnyFoodSearchProblem(gameState)

        "*** YOUR CODE HERE ***"
        return search.bfs(problem)
```

Input:

```
python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
```

Output:

```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 350.
Pacman emerges victorious! Score: 2360
Average Score: 2360.0
Scores:        2360.0
Win Rate:      1/1 (1.00)
Record:        Win
```