U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

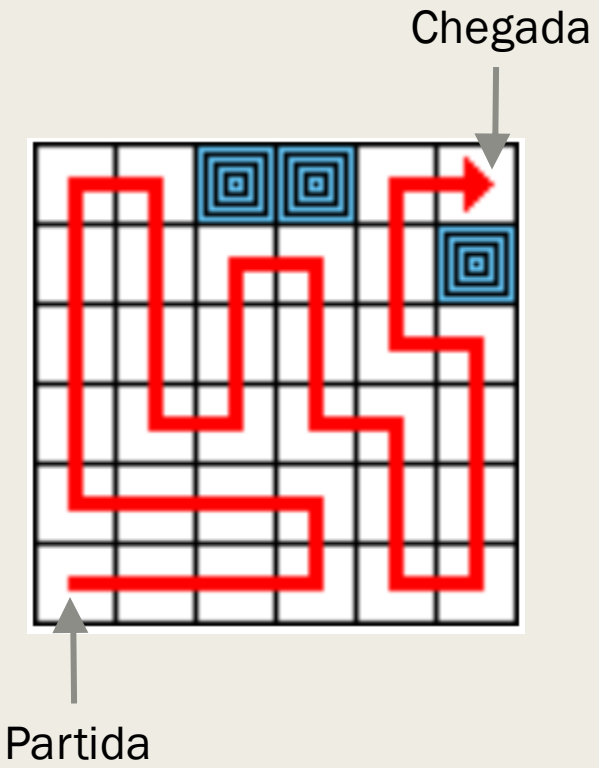
UNEQUAL LENGTH MAZES

Inteligência Artificial, 3LEIC01

Grupo 05_1A:
Henrique Ribeiro Nunes, up201906852
Margarida Assis Ferreira, up201905046
Patrícia do Carmo Nunes Oliveira, up201905427

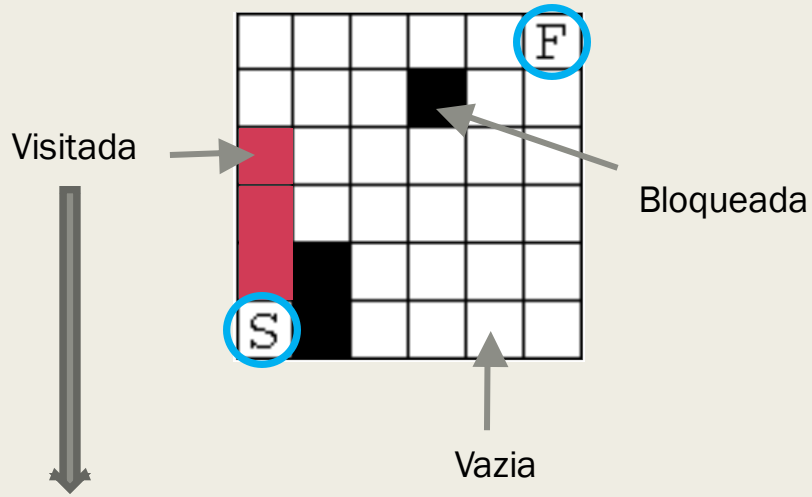
Descrição do Jogo

- O jogo ***Unequal Length Mazes*** consiste num tabuleiro de dimensões arbitrárias com algumas das suas células bloqueadas (obstáculos).
- O objetivo do jogo é encontrar um caminho desde a célula de partida até à célula de chegada que passe por todas as células disponíveis (não bloqueadas) apenas uma vez.
- O caminho deve alternar entre segmentos horizontais e verticais e dois segmentos consecutivos não podem ter o mesmo tamanho.



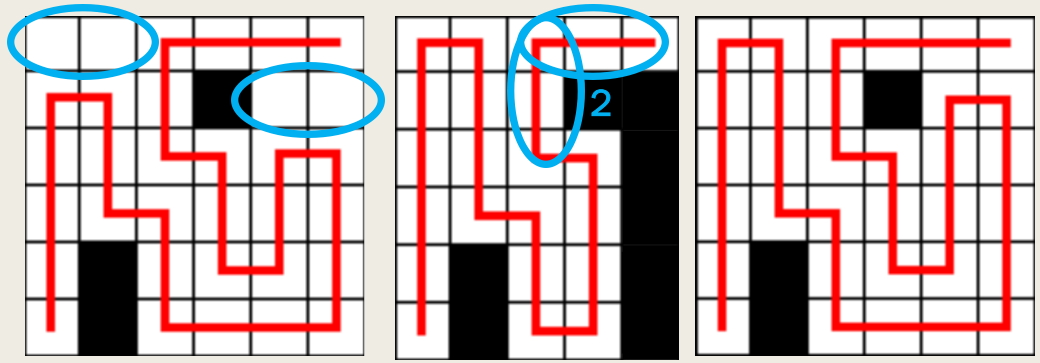
Formulação como Jogo Individual

- **Tipo de tabuleiro:** tabuleiro de dimensões arbitrárias, célula de partida no canto inferior esquerdo e célula de chegada no canto superior direito;
- **Tipo de peças:** células vazias, células bloqueadas e células visitadas;
- **Regras de movimento:** o jogador tem de alternar entre segmentos horizontais e verticais, sendo que dois segmentos consecutivos não podem ter o mesmo tamanho;



Próximo Movimento?
Continuar nesta direção ou posso trocar de direção

- **Condições para terminar jogo derrotado:** ser impossível de resolver, no sentido de não existir nenhum caminho possível que esteja de acordo com as regras do jogo;
- **Condições para terminar jogo com vitória:** chegar à célula do campo superior direito do tabuleiro, sendo que todas as células do tabuleiro que não eram bloqueadas foram visitadas e que com esse movimento final os segmentos consecutivos não ficaram com igual tamanho;
- **Pontuação:** o tempo gasto na resolução do puzzle penalizado pelo número de dicas utilizado.



Exemplos de não vitória

Vitória

Formulação como Problema de Pesquisa

- **Representação do Estado:**
  - Matriz que representa o tabuleiro - b[*HEIGHT*][*WIDTH*]:
    - Célula Vazia (*EC*): ***0***
    - Célula Visitada (*VC*): ***1***
    - Célula com Obstáculo (*BC*): ***9***
  - Célula atual: ***Row, Col, Dir*** ∈{*Up, Down, Left, Right*}, ***Length***
  - Comprimento do último segmento: ***LastLenght***

- **Estado Inicial:**
  - Tabuleiro vazio com obstáculos e célula inicial visitada.
  - Célula atual é a célula inicial (sem direção).
  - Não existe informação sobre o comprimento do segmento anterior.
  - Comprimento do segmento atual é zero.

- **Teste Objetivo:**
  - Tabuleiro com obstáculos e as restantes células todas visitadas.
  - Célula atual é a célula final.
  - Segmento atual e segmento anterior têm tamanhos diferentes.

- **Heurísticas:**
  - Inverso da distância de Manhattan à célula final.
  - De forma a garantir que todas as células são visitadas antes de nos aproximarmos da célula final.

- **Operadores:**

Nomes	Pré-condições	Efeitos	Custos
Up	Row > 0 Dir = Up b[Row-1, Col] = 0	Row -= 1 b[Row, Col] = 1 Lenght++	1
Down	Row < HEIGHT Dir = Down b[Row+1, Col] = 0	Row += 1 b[Row, Col] = 1 Lenght++	1
Left	Col > 0 Dir = Left b[Row, Col-1] = 0	Col -= 1 b[Row, Col] = 1 Lenght++	1
Right	Col < WIDTH Dir = Right b[Row, Col+1] = 0	Col += 1 b[Row, Col] = 1 Lenght++	1
SwapToUp	Length != 0 Length != LastLenght Dir=Left v Dir = Right	Dir = Up LastLenght = Lenght Lenght = 0	0
SwapToDown	Length != 0 Length != LastLenght Dir=Left v Dir = Right	Dir = Down LastLenght = Length Lenght = 0	0
SwapToLeft	Length != 0 Length != LastLenght Dir=Up v Dir = Down	Dir = Left LastLenght = Lenght Lenght = 0	0
SwapToRight	Length != 0 Length != LastLenght Dir=Up v Dir = Down	Dir = Right LastLenght = Lenght Lenght = 0	0

Implementação (1)

- Estruturas de Dados:

```python
class Node:
    def __init__(self, state, depth, cost, heuristic, parent):
        self.state = state
        self.depth = depth
        self.cost = cost
        self.heuristic = heuristic
        self.parent = parent
```

Árvore de Pesquisa com Nós (contêm informação sobre o estado, profundidade, custo, heurística e antecessor)

```python
class SearchProblem:
    def __init__(self, initState, isFinalState):
        self.initState = initState
        self.queue = [Tree.Node(initState, 0, 0, 0, -1)]
        self.isFinalState = isFinalState
        self.visited = []
```

Fila de Prioridade (o critério de ordenação dos nós varia em função do algoritmo)

- Estado Inicial, Teste Objetivo e Função de Heurística:

```python
initBoard = [[BC, BC, EC], [EC, EC, EC], [VC, BC, BC]]
H = len(initBoard)
W = len(initBoard[0])
currentCell = (H-1, 0, None, 0)
lastSegment = None

initState = (initBoard, currentCell, lastSegment)
```

Representação do Estado Inicial

```python
def isFinalState(state):
    (board, (row,col,dir,length), lastSegment) = state
    for line in board:
        for cell in line:
            if cell == EC:
                return False
    return row == 0 and col == W-1 and length != lastSegment
```

Função de Verificação de Estado Final

```python
def heuristics(state, type):
    '''Function with the different heuristics that can be used. '''
    def manhattan(x1, y1, x2, y2):
        return abs(x2-x1) + abs(y2-y1)

    (_, (row, col, _, _), _) = state
    if type == 1:
        dist = manhattan(row, col, 0, W-1)
        if dist == 0: return 0
        else: return 1/dist
    return 0
```

Função de Heurística

Implementação (2)

- **Algoritmos Implementados:**
  - Pesquisa em Largura *(BFS);*
  - Pesquisa em Profundidade *(DFS);*
  - Pesquisa em Profundidade Limitada *(Limited DFS);*
  - Aprofundamento Progressivo *(Iterative-Deepening);*
  - Pesquisa de Custo Uniforme *(Uniform-Cost);*
  - Pesquisa Gananciosa *(Greedy Algorithm);*
  - Algoritmo de Pesquisa A*.

```
def sortQueue(self, algorithm):
  if algorithm == algorithmTypes["breadth"]:
    self.queue.sort(key=lambda node: node.depth)
  elif algorithm == algorithmTypes["depth"]:
    self.queue.sort(key=lambda node: -node.depth)
  elif algorithm == algorithmTypes["depth_cut"]:
    self.queue.sort(key=lambda node: -node.depth)
  elif algorithm == algorithmTypes["iterative_deepening"]:
    self.queue.sort(key=lambda node: -node.depth)
  elif algorithm == algorithmTypes["uniform"]:
    self.queue.sort(key=lambda node: node.cost)
  elif algorithm == algorithmTypes["greedy"]:
    self.queue.sort(key=lambda node: node.heuristic)
  elif algorithm == algorithmTypes["A*"]:
    self.queue.sort(key=lambda node: node.cost + node.heuristic)
```

Ordenação da fila de prioridade segundo o critério apropriado para cada algoritmo

Referências Bibliográficas

Trabalhos relacionados e referências:

- Pathfinding e resolução de labirintos com algoritmo de pesquisa A*
- Algoritmo de pesquisa A* para resolução de labirintos
- "Rato no Labirinto" (jogo semelhante)