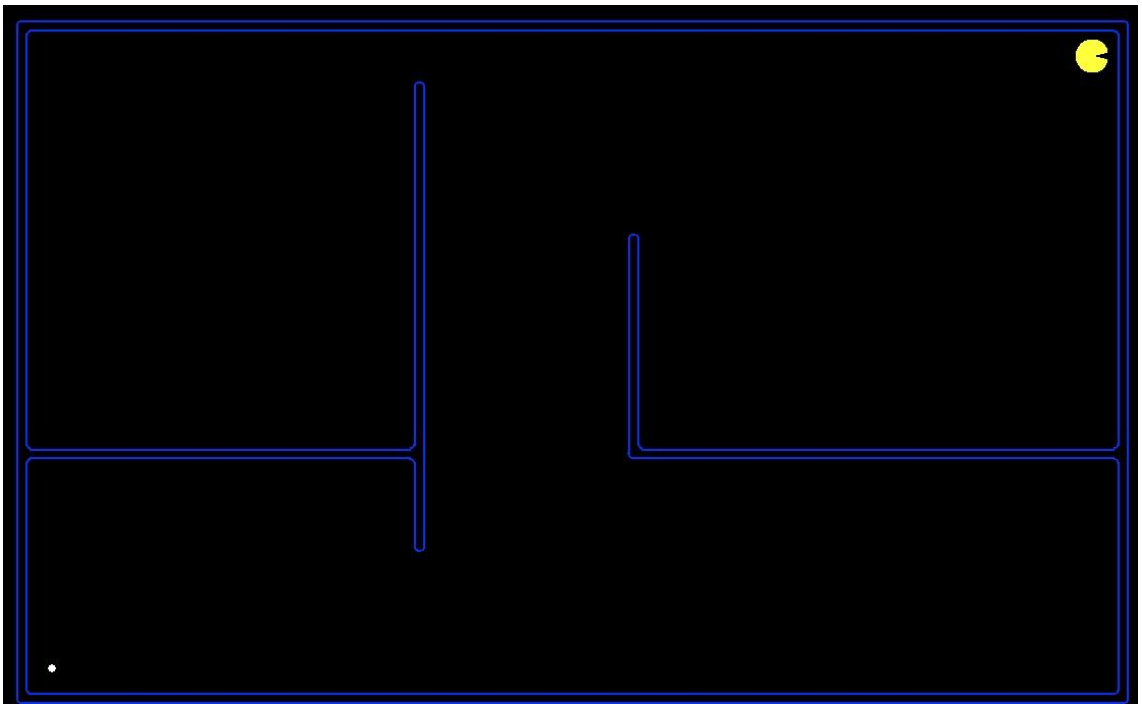
Relatório de Inteligência Artificial
Estratégias de busca no Open Maze do Pac-Man

Néli José da Fonseca Júnior

LP 2

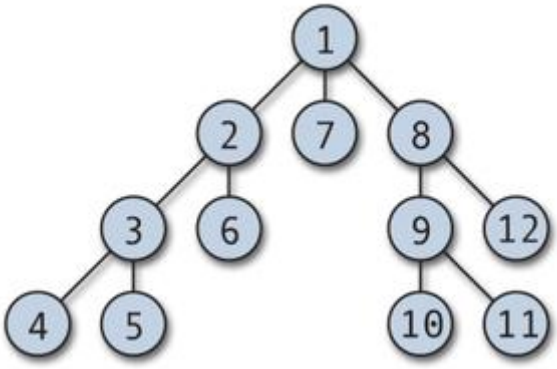
1. Labirinto de Testes (Open Maze)

O labirinto openMaze (abaixo) foi utilizado como testes para elaboração desse relatório. O objetivo é fazer com que o Pac Man vá até a bolinha utilizando cada algoritmo de busca.



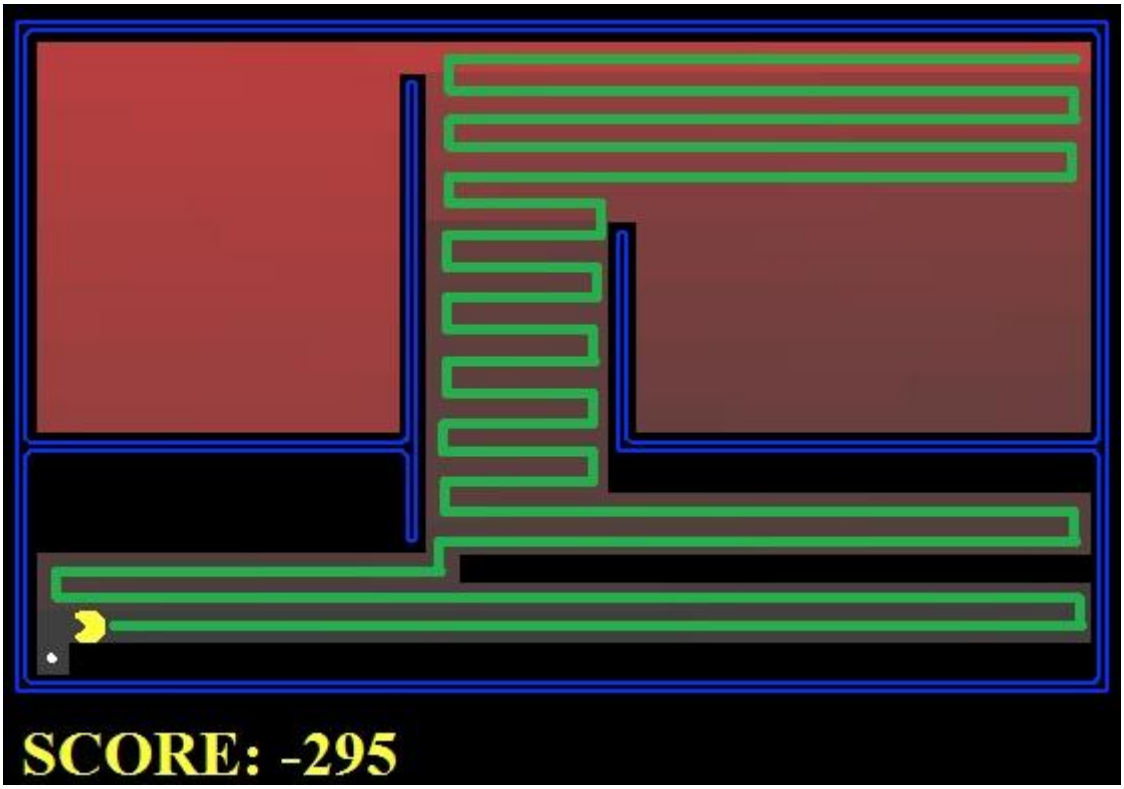
2. Busca em Profundidade (DFS)

Na busca em profundidade, o algoritmo parte da raiz e explora tanto quanto possível cada um de seus filhos, antes de retroceder.



Dessa forma, no caso do Open Maze (visto que é um labirinto aberto) a busca em profundidade não se comportará muito bem. A busca irá aprofundando na árvore até

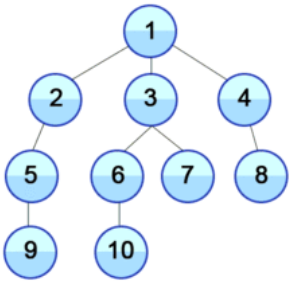
o final, ao chegar ao final irá retroceder uma posição e voltar a aprofundar, repetindo esses passos até chegar ao estado final. Passando para o ambiente do Pac-Man, ele irá percorrer em zig zag. Aprofundando até o último nó e voltando.



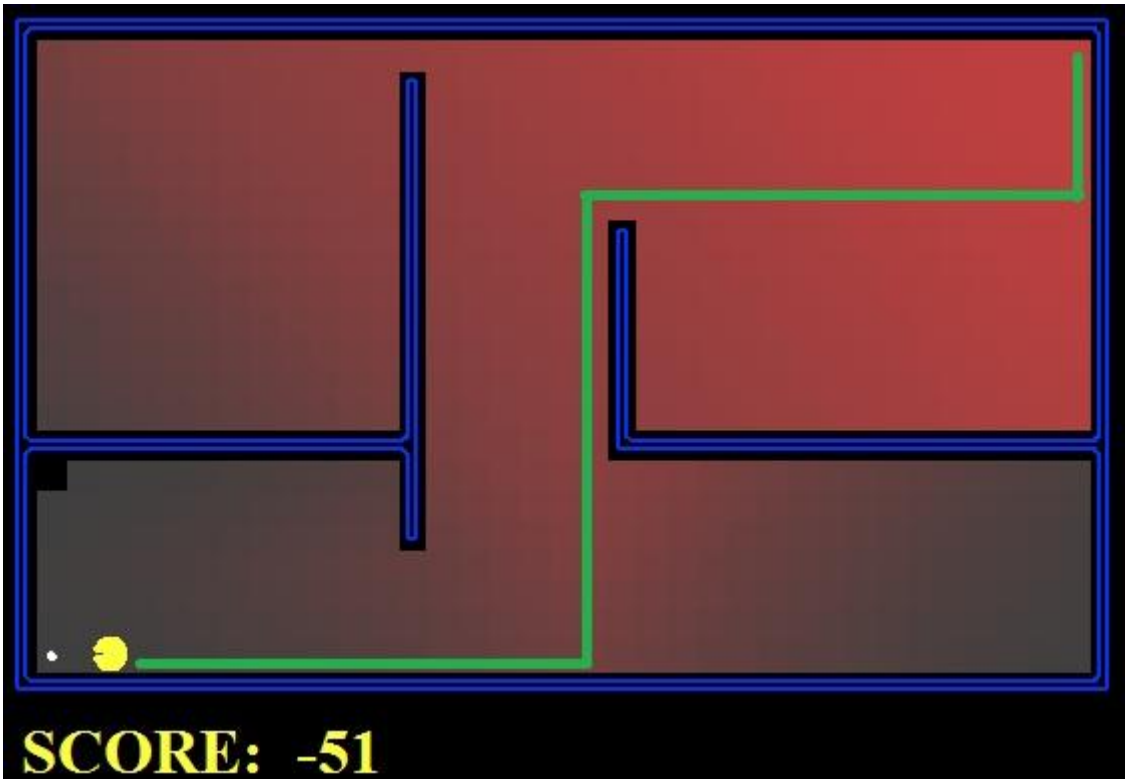
Custo do caminho: **158**
Nós expandidos: **315**
Tempo para encontrar: **0.1s**
Pontuação: **352**

3. Busca em Largura (BFS)

Na busca em largura, parte-se do vértice raiz e explora todos os vértices vizinhos. Para cada um desses vértices, é explorado seus respectivos vizinhos inexplorados e assim por diante, até que encontre o estado final. Desse modo, é possível construir uma árvore de distância mínima (menor número de arestas) entre a raiz e o final.



No ambiente do Pac-Man, é possível perceber que a busca em largura percorre praticamente todos os estados do labirinto (apenas 1 não é visitado), em contra partida ela também gera o melhor caminho, em relação a número de estados.

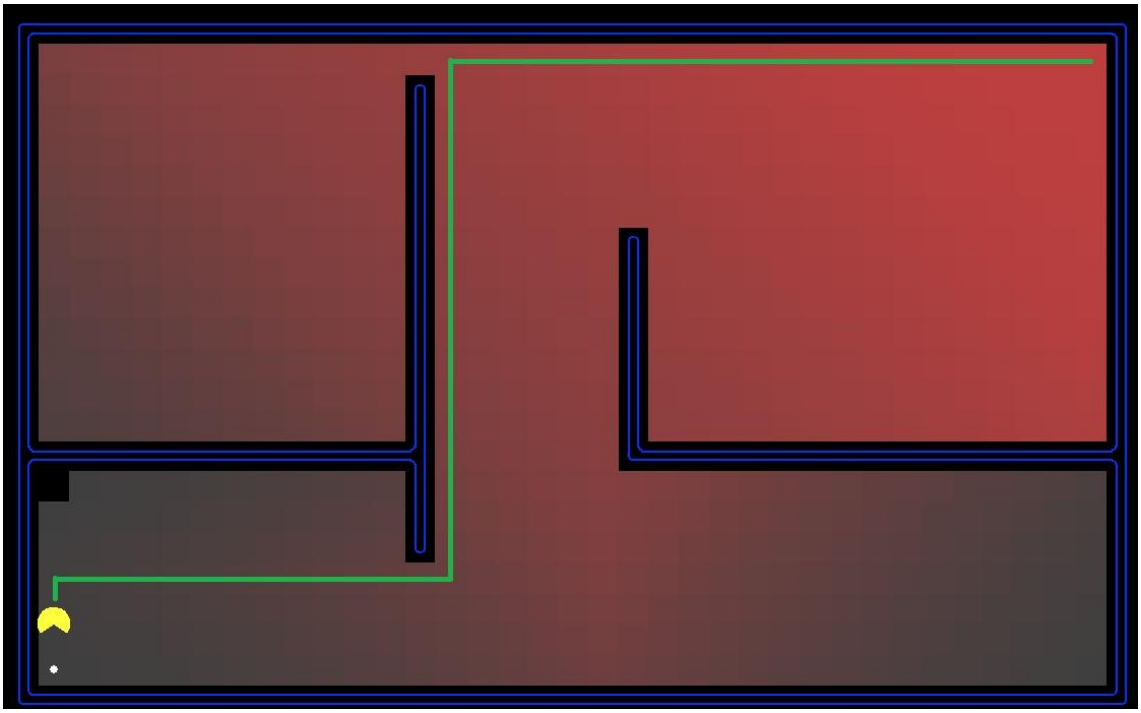


Custo do caminho: **54**
Nós expandidos: **680**
Tempo para encontrar: **0.1s**
Pontuação: **456**

4. **Busca de Custo Uniforme (UCS)**

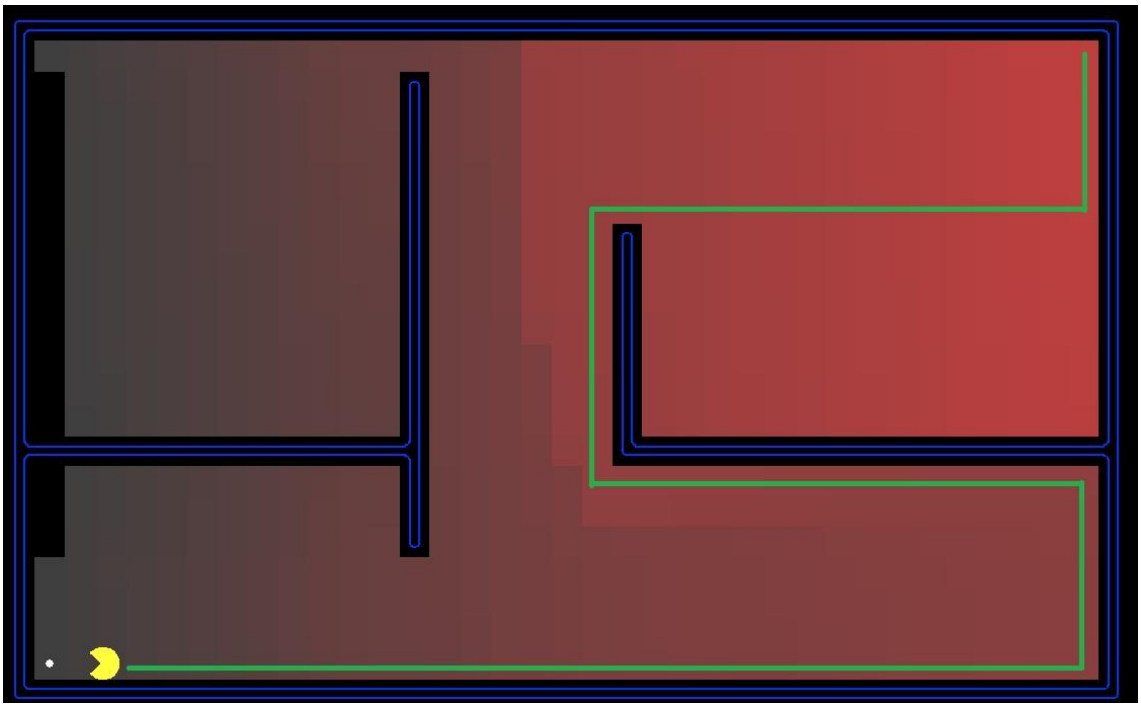
Essa é uma busca parecida com a busca em largura, porém é usada uma fila de prioridades para armazenar os vértices que forem sendo expandidos. Assim ela vai percorrendo expandindo as folhas com menores custos. Os testes com a UCS foram feitos de três formas: normal, usando um agente que penaliza os custos de posições do lado oeste do tabuleiro, e por último usando um agente que penaliza os custos de posições do lado leste do tabuleiro.

A busca padrão teve resultados bem parecidos com o da busca em largura, apesar do caminho final ter sofrido uma pequena alteração:



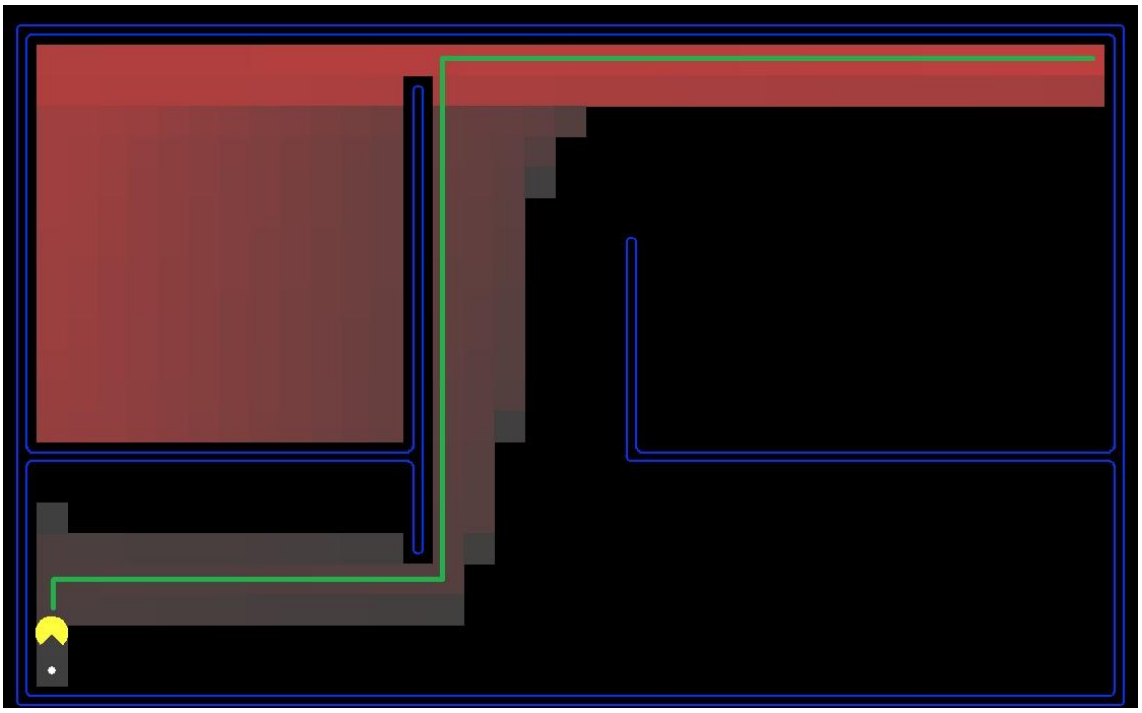
Custo do caminho: **54**
Nós expandidos: **680**
Tempo para encontrar: **0.2s**
Pontuação: **456**

Usando o agente que penaliza os custos do lado esquerdo do tabuleiro, a busca até atingiu a maioria dos vértices do lado esquerdo, mas o caminho final se manteve sempre o mais a direita possível:



Custo do caminho: **+-80 (por estimativa) *Algum bug no código do Pac-Man informa erroneamente que o custo desse caminho é 1.**
Nós expandidos: **650**
Tempo para encontrar: **0.2s**
Pontuação: **424**

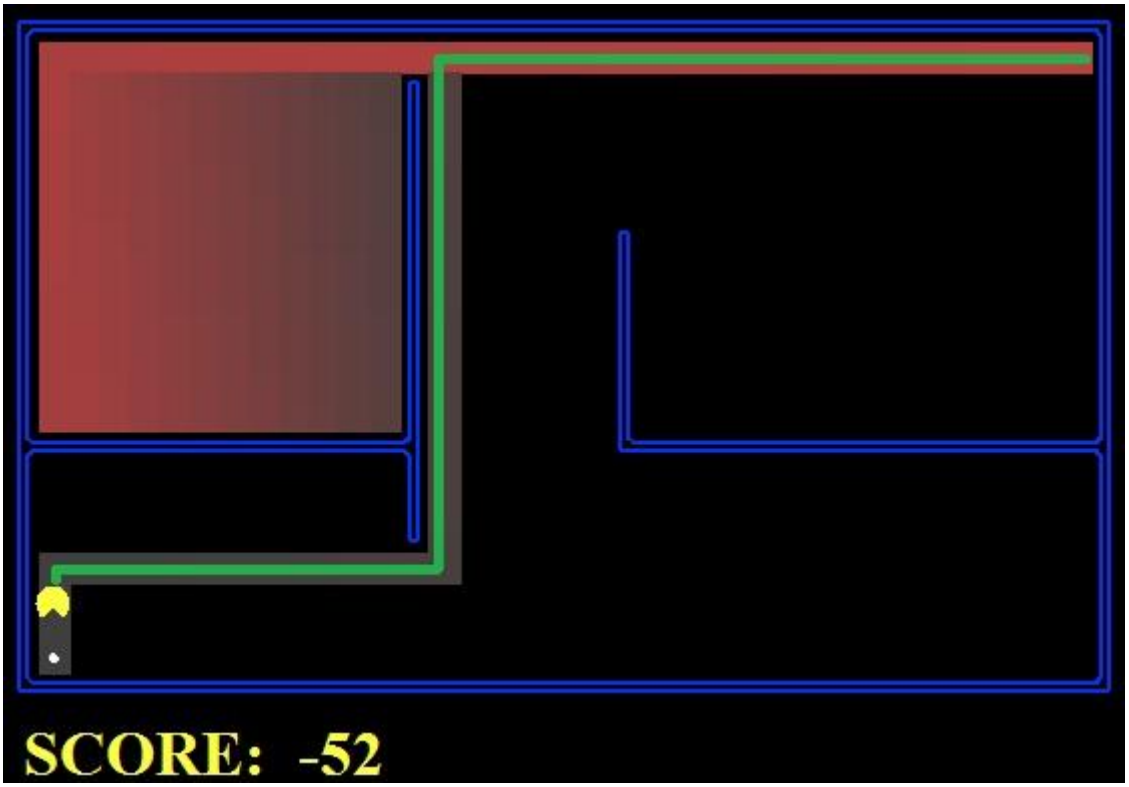
Já com o agente que penaliza os custos do lado direito do tabuleiro, o resultado foi mais satisfatório para este caso. Pelo fato do estado final (bolinha) estar no lado esquerdo do tabuleiro, o caminho foi encontrado rapidamente e nem foi preciso expandir os nós do lado direito do tabuleiro:



Custo do caminho: **54**
Nós expandidos: **287**
Tempo para encontrar: **0.1s**
Pontuação: **456**

5. **Busca A***

O funcionamento da busca A* é muito parecido com o do custo uniforme, a grande diferença é que ao invés de seus custos serem baseados nas distância de nós, ele também leva em consideração uma heurística. No teste foi utilizado a heurística da distância de Manhattan.



Custo do caminho: **54**
Nós expandidos: **145**
Tempo para encontrar: **0.0s**
Pontuação: **456**

6. **Resultados**

MELHORES		
Custo	BFS, UCS(Normal), UCS(Penalidade a direita) e A*	54
Nós Expandidos	A*	145
Tempo	A*	0s
Pontuação	BFS, UCS(Normal), UCS(Penalidade a direita) e A*	456
PIORES		
Custo	DFS	158
Nós Expandidos	UCS(Penalidade a esquerda)	680
Tempo	UCS(Normal) e UCS(Penalidade a esquerda)	0.2s
Pontuação	DFS	352

LP 3

Estado: Tupla formada por (posição, [cantos visitados])

Goal State: Apenas verifica se a posição atual é um canto, se for adiciona na lista de cantos já visitados. Então é verificado se possui 4 elementos na lista, caso possua então chegou ao estado final.

Proximos Estados: Para cada possível ação (Norte, Sul, Leste e Oeste), então verifica se cada ação é possível (não possui parede) e se é um canto não visitado. Se a ação for

possível, adiciona na lista de sucessores e se for um canto não visitado, adiciona na lista do estado.

Heuristica Para o cornersProblem:

Para cada canto, calcula-se: |Xatual – Xcanto| + |Yatual – Ycanto|

Retorna o menor valor.

Custo do caminho: **106**
Nós expandidos: **868**
Tempo para encontrar: **0.3s**
Pontuação: **434**

Heuristica Para foodProblem:

Para cada comida, calcula-se: |Xatual – Xcomida| + |Yatual – Ycomida|

Retorna o maior valor.

Custo do caminho: **60**
Nós expandidos: **9309**
Tempo para encontrar: **18.8s**
Pontuação: **570**

Busca Sub-Ótima:

Aplicou-se o algoritmo de UCS, com uma única alteração. O estado final passa a ser a primeira comida avistada.

Custo do caminho: **323**
Pontuação: **2387**

LP 4

Agente Reflexo

A heurística divide a distância do fantasma mais próximo pela distância da comida mais próxima e então acrescenta a pontuação

H = (Fmaisproximo / Cmaisproximo) + pontuação obtida no próximo estado

Labirinto testClassic executado 10 vezes	
Vitórias	9/10
Média de Pontuação	462,2
Labirinto mediumClassic com 1 fantasma executado 10 vezes	
Vitórias	10/10
Média de Pontuação	1142,7
Labirinto mediumClassic com 2 fantasmas executado 10 vezes	
Vitórias	5/10
Média de Pontuação	592,1

Labirinto openClassic executado 10 vezes	
Vitórias	10/10
Média de Pontuação	1084,3

Agente Minimax

O algoritmo minimax verifica qual dos agentes está sendo executado no momento (Pac Man ou um dos fantasmas. No caso do agente Pac Man, o algoritmo pegará todas as possíveis ações (removendo a de permanecer imóvel) e maximizar a pontuação atual com a pontuação obtida recursivamente do próximo estado. A implementação do agente fantasma é similar, o que difere é que o algoritmo procura pelo pior caso (minimização) para que não tenha riscos para o Pac Man. A função é executada

Labirinto minimaxClassic executado 10 vezes com profundidade 4	
Vitórias	7/10
Média de Pontuação	213
Labirinto trappedClassic executado 10 vezes com profundidade 3	
Vitórias	0/10
Média de Pontuação	-501

Agente Alpha Beta

O valor da heurística no algoritmo com corte alfa/beta é representado por uma lista com o numero e a ação que deve ser executada para chegar ao próximo estado. O funcionamento é similar ao do minimax, mas é introduzido um corte que é representado pelas variáveis alfa e beta. No caso do agente Max, se a pontuação (obtida de forma recursiva) for menor do que a representada em beta, alfa é salvo com a pontuação, que então é retornada. Já o agente Min faz o inverso, se alfa for maior, a pontuação é salva em beta.

Labirinto smallClassic executado 10 vezes	
Vitórias	1/10
Média de Pontuação	-47.4

Agente Expectimax

O Agente Expeticmax testa cada possível movimento do Pac Man e simula sua execução de forma recursiva. Se o agente for 0 (vez do Pac Man), o algoritmo retorna a ação que resulta na melhor pontuação obtida. Caso seja maior que 0 (quantidade de fantasmas), será retornado a média de todas as possíveis pontuações.

Labirinto trappedClassic executado 10 vezes	
Vitórias	6/10
Média de Pontuação	118.4

Função de Avaliação

A nova função de avaliação é calculada pela menor distância de Manhattan do novo possível estado do Pac Man até o fantasma mais próximo, dividido pela distância do possíveis próximos

estados até a comida mais próxima. Tudo isso acrescentado da pontuação que será obtida ao mover-se para os possíveis próximos estados.

Labirinto smallClassic executado 10 vezes	
Vitórias	9/10
Média de Pontuação	948.6
Labirinto mediumClassic executado 10 vezes	
Vitórias	10/10
Média de Pontuação	1611

LP 5

Observe

Para cada posição possível de se mover é calculado a distância de Manhattan do pacman até ela. A posição relativo a cada posição no mapa de probabilidades recebe a probabilidade relativa a crença atual para aquela posição multiplicada pela probabilidade através do barulho.

Elapse Time

Para cada crença não nula, ela é atualizada levando em conta cada possível próxima posição.

Choose Action

Para cada fantasma vivo, é escolhida a posição com maior probabilidade de ele estar. Então a ação escolhida será a mais próxima entre as posições separadas acima.