Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Biológicas
Departamento de Computação
BCC325 – Inteligência Artificial

Trabalho Prático I

Aplicação de técnicas de IA no domínio do Pacman

Saulo Rocha de Assis	13.2.4639
Ana Luiza Fernandes Moraes	15.1.4244
Barbara Oliveira Neves	15.2.4046

Professor:

Prof. Alvaro Guarda

Ouro Preto, abril de 2018

RESUMO

O trabalho proposto na disciplina BCC325 é baseado no “Pacman Project” da Universidade de Berkeley, o qual visa aplicar conceitos de IA no domínio do Pacman. A mecânica do jogo Pacman é simples: o jogador é uma cabeça redonda com uma boca que se abre e fecha, posicionado em um labirinto simples repleto de pastilhas e 4 fantasmas que o perseguem. O objetivo era comer todas as pastilhas sem ser alcançado pelos fantasmas, em ritmo progressivo de dificuldade. Neste trabalho, o Pacman deve encontrar um caminho para atingir certas posições no seu mundo, composto inicialmente por um labirinto. O código foi desenvolvido na linguagem Python e é constituído de vários arquivos Python. Deve-se implementar os algoritmos de busca genéricos que foram apresentados em sala de aula para melhorar o desempenho do Pacman, estes são: busca em profundidade com retrocesso – DFS, busca em extensão (largura) – BFS, busca de custo uniforme – UCS e busca A*.

Sumário

Introdução

Uma das técnicas mais utilizadas em Inteligência Artificial é a busca no espaço de estados. Baseia-se em supor a existência de um agente capaz de executar ações que modificam o estado corrente de seu mundo. Desta forma, sendo dados um estado inicial representando a configuração corrente do mundo do agente, um conjunto de ações que o agente é capaz de executar e uma descrição do estado meta que se deseja atingir, a solução do problema consiste numa sequência de ações que, quando executada pelo agente, transforma o estado inicial num estado meta. Neste trabalho foi utilizado o conceito de espaço de estados para programar algoritmos que permitam que o Pacman encontre um caminho para atingir certas posições no seu mundo, composto inicialmente por um labirinto. Em seguida, compara-se os resultados obtidos pelas estratégias de busca cega (em largura e em profundidade) e as estratégias de busca heurística (custo uniforme e A*) como formas de garantir a qualidade das soluções e melhorar a eficiência da busca determinística.

Objetivos

- Implementar algoritmos de busca genéricos para melhorar o desempenho do Pacman.
- Entender e completar um código, desenvolvido na linguagem Python.
- Verificar a aplicabilidade das buscas em espaço de estado estudadas em sala, sendo essas: Busca em Profundidade com retrocesso (DFS), Busca em Extensão (Largura) (BFS), Busca de Custo Uniforme (UCS) e Busca A*.
- Analisar e explicar os resultados desses.

Busca em profundidade com retrocesso – DFS

A Busca em Profundidade explora todos os vértices de um grafo, usando como critério o vértice visitado mais recentemente e não marcado, tem como característica principal a utilização de uma pilha (stack) explícita ou recursividade para guiar a busca. Nesse método, expande-se sempre o estado mais à esquerda, no nível mais profundo da árvore de busca, até que se encontre uma solução ou atinja um estado que não pode ser expandido, seja porque ele já foi visitado anteriormente, ou porque não há nenhum operador aplicável para a sua expansão. Caso a expansão não seja possível, como dito no último caso, é feito um retrocesso (desempilhamento) e a busca é reiniciada no próximo estado ainda não expandido, posicionado mais à esquerda, no nível mais profundo da árvore. Deve-se observar que esse algoritmo apesar de gastar pouca memória, não é completo nem ótimo, sua complexidade de tempo é dada em O(m+n).

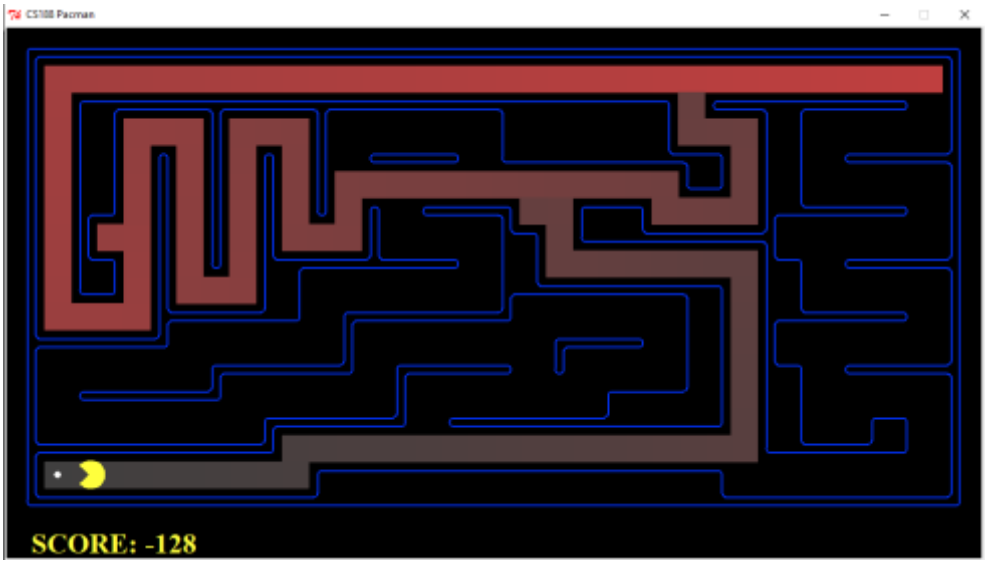

Figura 1 – representação gráfica do caminho do pac-man usado DFS – mediumMaze.

```
C:\Windows\system32\cmd.exe

C:\Users\saulo\OneDrive\UFOP\18.1\IA\TP 1>pacman.py -l mediumMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[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
```

Figura 2 - Resultado da DFS – mediumMaze.

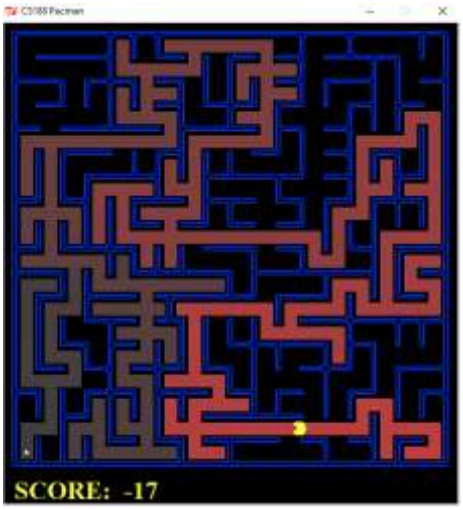

Figura 3 - representação gráfica do caminho do pac-man usado DFS – bigMaze.

```
C:\Windows\system32\cmd.exe

C:\Users\saulo\OneDrive\UFOP\18.1\IA\TP 1>pacman.py -l bigMaze -z .5 -p SearchAgent
[SearchAgent] using function depthFirstSearch
[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
```

Figura 4 - Resultado da DFS – bigMaze.

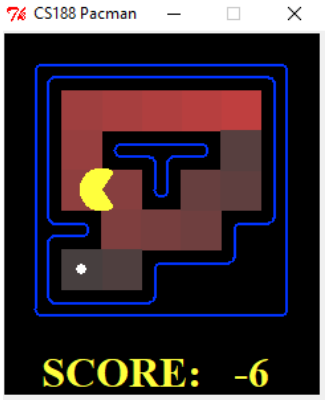

Figura 5 - representação gráfica do caminho do pac-man usado DFS – tinyMaze.

```
C:\Windows\system32\cmd.exe

C:\Users\saulo\OneDrive\UFOP\18.1\IA\TP 1>pacman.py -l tinyMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[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
```

Figura 6 - Resultado da DFS – tinyMaze.

Busca em extensão (Largura) – BFS

A Busca em Largura explora todos os vértices de um grafo, usando como critério o vértice visitado menos recentemente e não marcado, tem como característica principal a utilização uma fila para guiar a busca. A única diferença entre os algoritmos BFS e DFS é a estrutura de dados que estes usam para suas implementações, como dito anteriormente, a DFS é guiada por uma pilha enquanto a BFS é guiada por uma fila. Na busca em largura, primeiro é feita a expansão do estado inicial (nível 0), sendo seus sucessores posicionados no nível 1 da árvore de busca. Em seguida, cada um dos estados do nível 1 são expandidos, sendo seus sucessores posicionados no nível 2, e assim por diante, ou seja, nesse caso é feita a seleção do nó mais superficial que ainda não foi expandido. Este algoritmo, por sua vez, é completo e ótimo quando o custo de cada passo é igual e tem complexidade de tem O(n+m).

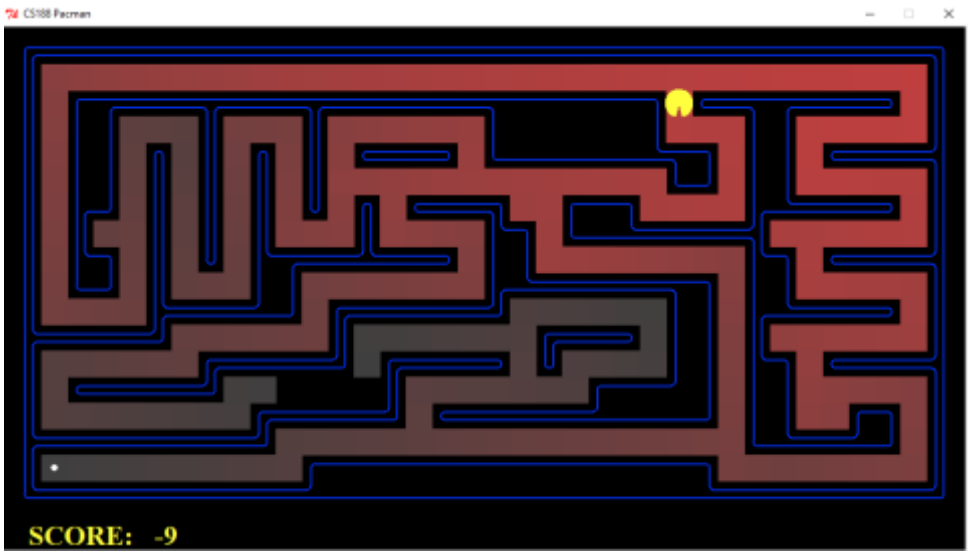

Figura 7 - representação gráfica do caminho do pac-man usado BFS – mediumMaze.

```
C:\Users\saulo\OneDrive\UFOP\18.1\IA\TP 1>pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
[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
```

Figura 8 - Resultado da BFS – mediumMaze.

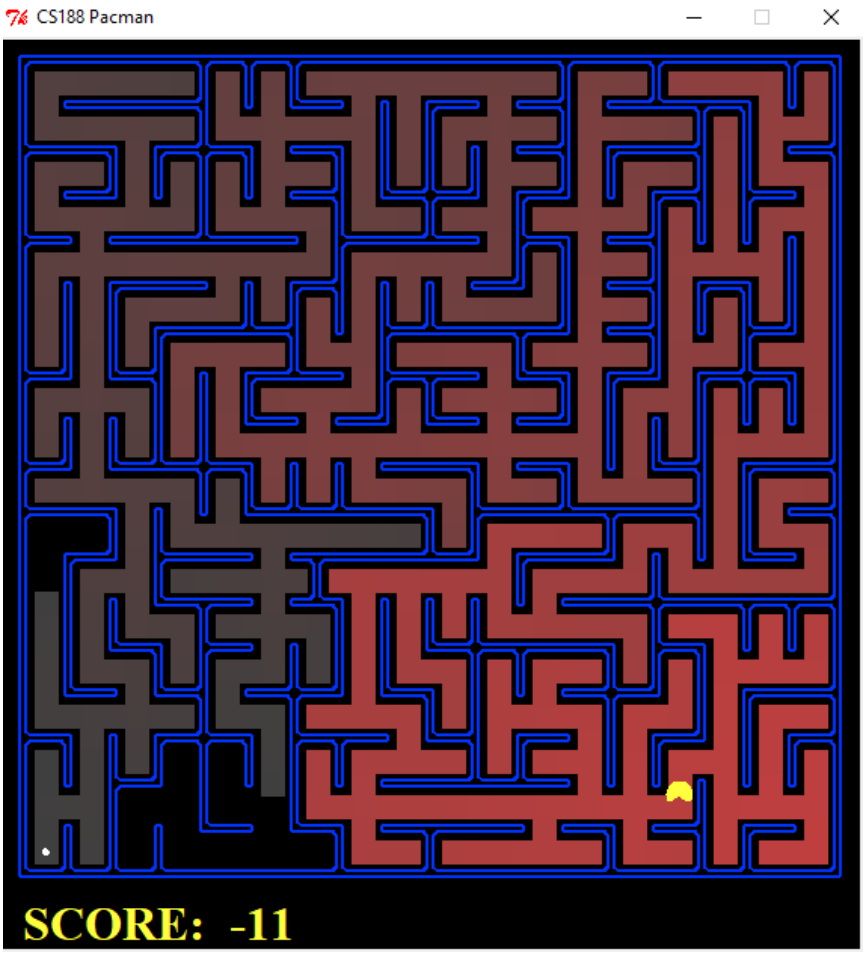

Figura 9 - representação gráfica do caminho do pac-man usado BFS – bigMaze.

```
C:\Users\saulo\OneDrive\UFOP\18.1\IA\TP 1>pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
[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
```

Figura 10 - Resultado da BFS – bigMaze.

Custo uniforme – UCS

O método de Custo uniforme, diferentemente dos métodos anteriores, considera também o custo das operações, expandindo o nó de custo mínimo primeiro e obedece a sistematização da ordem de inserção e remoção de uma fila de prioridade. A diferença deste algoritmo para os anteriores é que a estrutura de dados adotada é uma fila de prioridade. Essa busca, apesar de ser completa e ótima, explora opções em todas as direções e não leva em conta a localização do alvo.

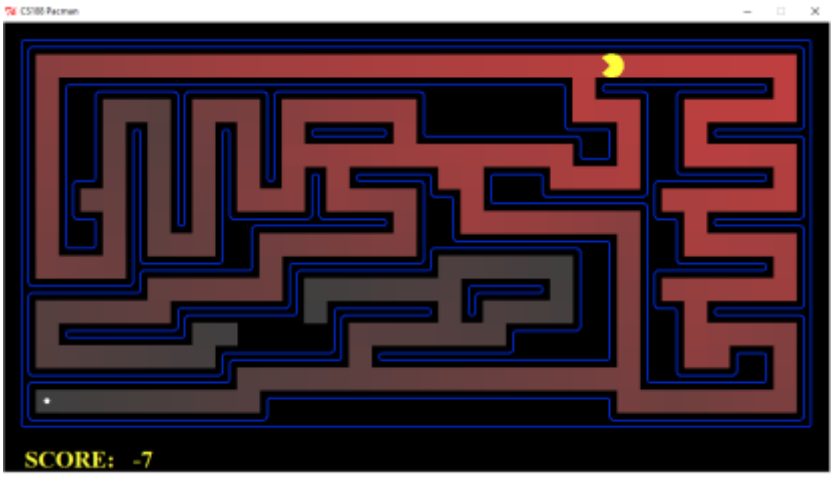

Figura 11 - representação gráfica do caminho do pac-man usado UCS – mediumMaze.

```
C:\Users\saulo\OneDrive\UFOP\18.1\IA\TP 1>pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[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
```

Figura 12 - Resultado da UCS – mediumMaze.

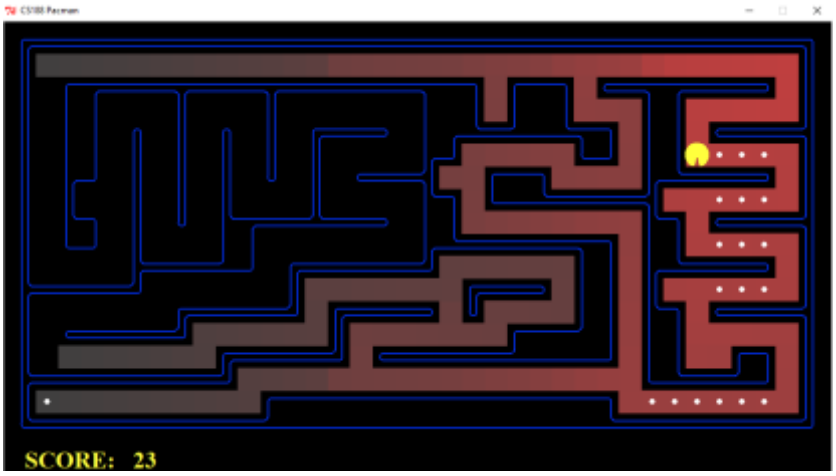

Figura 13 - representação gráfica do caminho do pac-man usado UCS – mediumDottedMaze.

```
C:\Users\saulo\OneDrive\UFOP\18.1\IA\TP 1>pacman.py -l mediumDottedMaze -p StayEastSearchAgent
Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:        646.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figura 14 - Resultado da UCS – mediumDottedMaze.

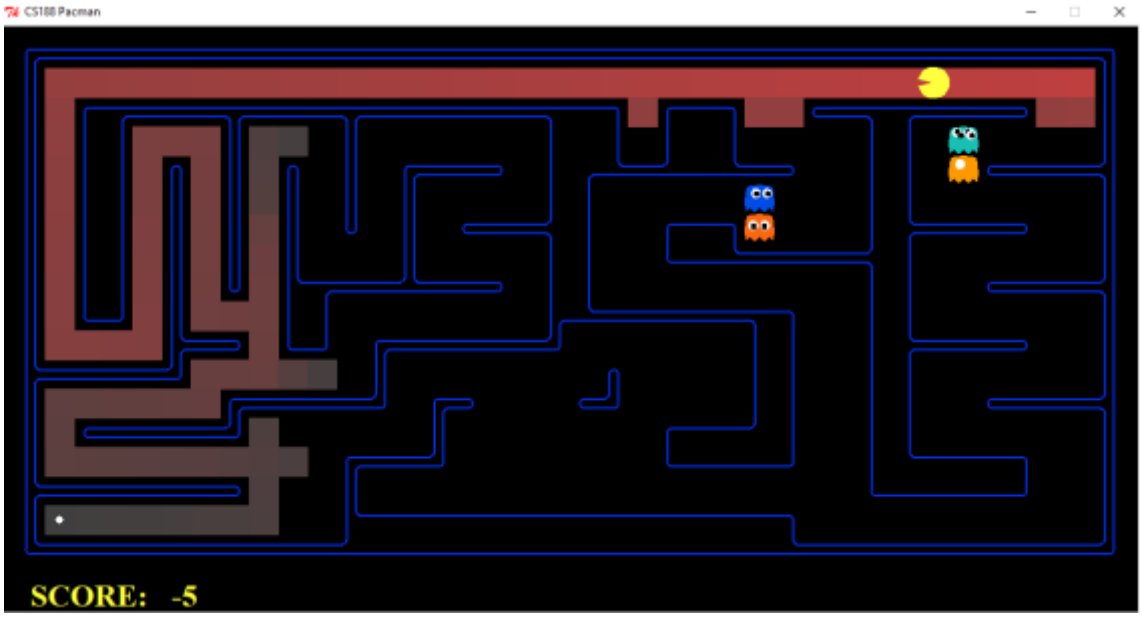

Figura 15 - representação gráfica do caminho do pac-man usado UCS – mediumScaryMaze.

```
C:\Users\saulo\OneDrive\UFOP\18.1\IA\TP 1>pacman.py -l mediumScaryMaze -p StayWestSearchAgent
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores:        418.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figura 16 - - Resultado da UCS – mediumScaryMaze.

Árvore A*

A * é um algoritmo de busca heurística, o que significa que resolve problemas pesquisando entre todos os caminhos possíveis para a solução para aquele que incorre no menor custo. Entre esses caminhos, primeiro considera-se os que parecem liderar mais rapidamente para a solução. É formulado em termos de gráficos ponderados: a partir de um nó específico de um grafo, ele constrói uma árvore de caminhos a partir desse nó, expandindo caminhos um passo por vez, até que um de seus caminhos termine no nó de objetivo predeterminado. Em cada iteração de seu loop principal, A * precisa determinar qual dos seus caminhos parciais se expandirá para um ou mais caminhos mais longos. Ele faz isso com base em uma estimativa do custo (peso total) ainda para ir ao nó objetivo. Especificamente, A * seleciona o caminho que minimiza: F(n) = g (n) + h (n). Onde n é o último nó no caminho, g (n) é o custo do caminho do nó de início para n, e h (n) é uma heurística que estima o custo do caminho mais barato de n para o objetivo. As A * usam uma fila de prioridade para executar a seleção repetida de nós de custo mínimos (estimados) para expandir. Em cada etapa do algoritmo, o nó com o menor valor f (x) é removido da fila, os valores f e g dos vizinhos são atualizados em conformidade e esses vizinhos são adicionados à fila. O algoritmo continua até que um nó de objetivo tenha um valor f menor do que qualquer nó na fila (ou até a fila estar vazia). O valor f do objetivo é então o comprimento do caminho mais curto. A* é ótima se h é consistente e UCS é um caso especial (h = 0 é consistente).

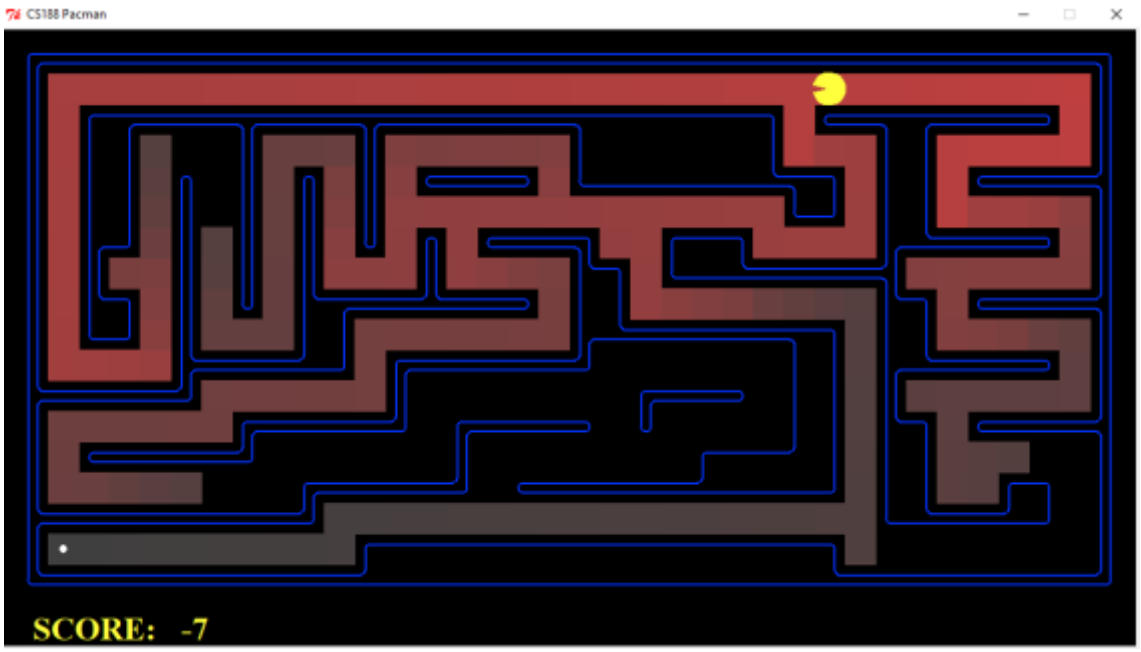

Figura 17 - representação gráfica do caminho do pac-man usado A - mediumMaze.*

```
C:\Users\saulo\OneDrive\UFOP\10.1\IA\TP 1>pacman.py -l mediumMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristi
c
[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
```

Figura 18 - Resultado da A - mediumMaze.*

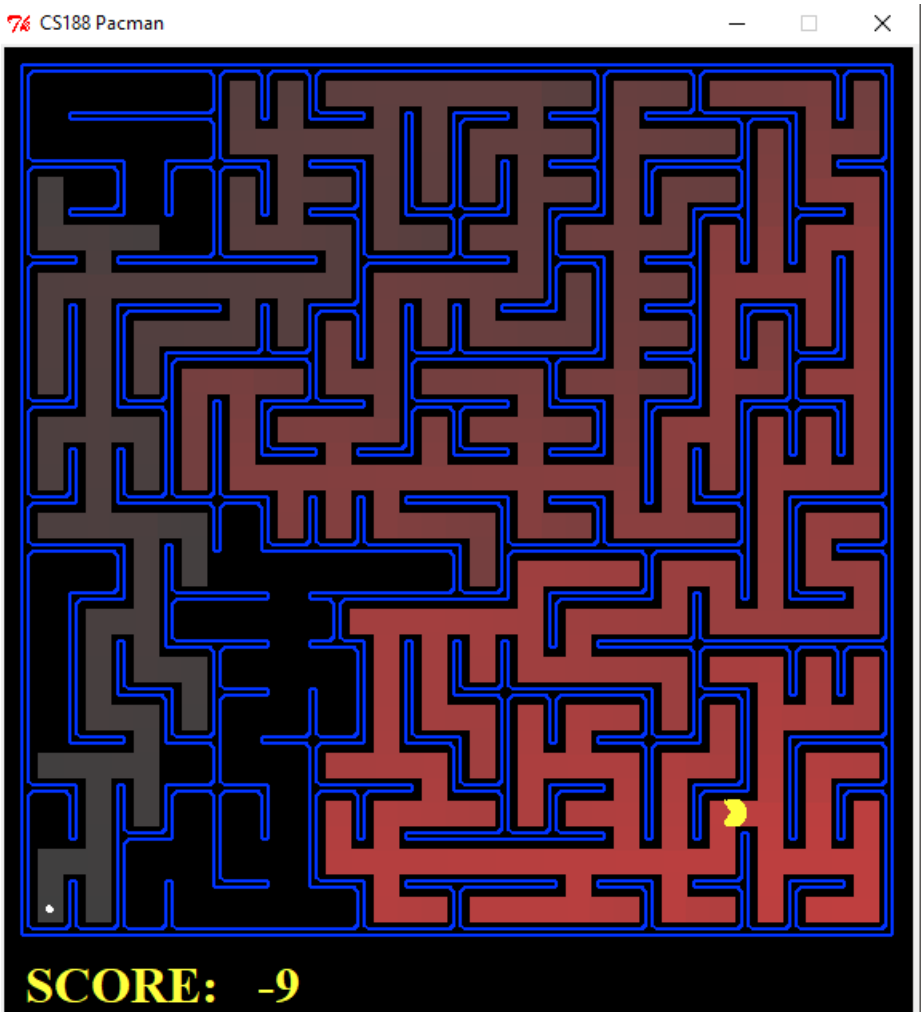

Figura 19 - representação gráfica do caminho do pac-man usado A - bigMaze.*

```
C:\Users\saulo\OneDrive\UFOP\10.1\IA\TP 1>pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuri
stic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 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
```

Figura 20 - Resultado da A - bigMaze.*

Verificações OpenMaze

Em seguida, foram feitas as verificações dos resultados do custo e nós expandidos para cada uma das estratégias nos mediumMaze e bigMaze, obtendo êxito quanto aos valores esperados. Por fim, verifica-se o comportamento do Pacman e os resultados obtidos no openMaze para as várias estratégias, comparando-as e analisando.

Resultados

Os testes de execução do Pacman foram feitos no mapa openMaze considerando as quatro estratégias de busca implementadas no trabalho. Através desses testes foi possível verificar as diferenças entre os caminhos percorridos, os custos totais e a quantidade de nós expandidos para cada algoritmo de busca estudado. As diferenças citadas podem ser vistas na Tabela 1:

Algoritmo	Tamanho da entrada	Custo	Nós Expandidos
DFS	mediumMaze	130	146
DFS	bigMaze	210	390
BFS	mediumMaze	68	269
BFS	bigMaze	210	620
UCS	mediumMaze	68	269
UCS	bigMaze	1	186
A*	mediumMaze	68	221
A*	bigMaze	210	549

Análise

Os testes feitos permitiram observar que o DFS teve um custo aproximadamente 5,5 vezes maior que os demais métodos de busca, fato que foi possível observar pela animação no mapa, já que o Pacman não percorreu o melhor caminho, dando voltas. Como ele não é completo, nem encontra o caminho ótimo, não expandiu tantos nós quanto o UCS e o BFS. Os demais métodos, por sua vez, percorreram o mesmo caminho, obtendo assim o mesmo custo. Isso acontece porque o custo considerado no overMaze para cada função foi unitário na lógica do programa, ou seja, as filas de prioridade (no caso do UCS e A*) se comportarão da mesma forma que a fila (no caso do BFS), caso contrário, o UCS apresentaria um custo menor, por expandir a fronteira

do nó de menor custo, nesse caso, essas três estratégias alcançam o custo ótimo. Pode-se observar, também, que o A* expande menos nós, devido à heurística, uma vez que essa, além de considerar o custo, considera a distância entre o Pacman e o objetivo, facilitando que o melhor caminho seja encontrado.

Conclusão

Pode-se observar, a partir da execução desta etapa do projeto, a aplicabilidade dos algoritmos de busca em grafo estudados. Com conceitos da inteligência artificial, foi possível melhorar o desempenho do Pacman. Além disso, a comparação entre os resultados obtidos em cada método permite constatar as diferenças entre os métodos cegos e heurísticos. Portanto, após a análise dos dados obtidos, observa-se que o desempenho do A* foi o melhor, uma vez que, além de ser uma fila de prioridade (considerando os custos), verifica, também, a viabilidade da expansão do nó, através de uma heurística consistente.