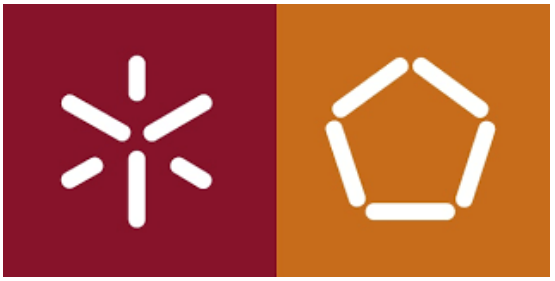
Universidade do Minho



Inteligência Artificial

Trabalho Prático

2ª Fase

Ano letivo 2022/2023

a96106, Miguel Silva Pinto

a97755, Orlando José da Cunha Palmeira

a97613, Pedro Miguel Castilho Martins

1. Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Inteligência Artificial da Licenciatura em Engenharia Informática da Universidade do Minho, e teve como objetivo a resolução de problemas através da conceção e implementação de algoritmos de procura. Neste trabalho foram implementadas estratégias para a resolução de problemas com o uso de algoritmos de procura, partindo da formulação do problema em questão. Ao longo deste documento vai ser explicado todo o processo de desenvolvimento dos mecanismos de procura implementadas neste projeto.

2. Formulação do problema

Uma vez que o problema se encontra num ambiente determinístico e completamente observável, em que o agente "sabe" exatamente o estado em que estará e as soluções pretendidas são sequências, podemos afirmar que este é um problema de estado único. Além disso, a sua formulação pode ser efetuada da seguinte forma:

- **Representação do estado:** Grafo não orientado, em que cada nodo representa uma posição possível no circuito e cada aresta representa o trajeto entre posições.
- **Estado inicial:** A posição inicial representa o local de início da corrida de onde o carro parte.
- **Estado/teste objetivo:** Alcançar uma das posições finais da meta.
- **Operadores:** Deslocação entre posições adjacentes incluindo as diagonais.
- **Solução:** Um caminho válido (sequência de posições percorridas) que comece na posição inicial e termine numa das posições finais.
- **Custo da solução:** Distância total do percurso feito pelo agente.

3. Criação do circuito

O circuito é criado de acordo com um determinado ficheiro que representa esse circuito, com um formato semelhante ao seguinte:

```
XXXXXXXXXX
XXX--X--XX
XP---X---F
X---XX---F
X--------X
XXX--X-XXX
XXXXXXXXXX
```

Figura 1: Exemplo de ficheiro de circuito.

O circuito mostrado na Figura 1 é um circuito com uma posição inicial para o carro (representada pela letra “P”), com duas posições finais possíveis (representadas pela letra “F”) e com as delimitações do circuito representadas pela letra “X”, representando obstáculos que não podem ser transpostos. Existem diversos circuitos possíveis que poderiam ser analisados através do nosso programa, desde que respeitem as seguintes restrições:

- O circuito deverá estar cercado por delimitações “X” ou posições finais “F”.
- O circuito deverá ter um formato retangular.

4. Funcionamento do programa

4.1. Criação do grafo

O programa quando é executado pede ao utilizador para inserir a path do ficheiro onde está descrito o circuito que será utilizado.
Depois de receber essa informação é criado o grafo que representa as posições e possíveis movimentos dos carros no circuito. Todas as arestas do grafo terão um custo de 1.

4.2. Menu

O programa de seguida apresenta ao utilizador o seguinte Menu:

```
**** Menu ****
1-Imprimir Grafo
2-Desenhar Grafo
3-Indicar posições iniciais e finais do circuito
4-Imprimir nodos do Grafo
5-Aplicar algoritmos aos carros
0-Sair
Introduza a sua opcão:
```

Figura 2: Menu do programa.

O Menu apresenta as seguinte opções:

1. **Imprimir Grafo** - Onde mostramos todos os nodos do grafo e os possíveis próximos nodos.
2. **Desenhar Grafo** - Onde utilizamos a biblioteca networkx para desenhar a estrutura do grafo utilizado.
3. **Posições iniciais e finais** - Onde mostramos as posições iniciais dos jogadores e as posições das metas do circuito.
4. **Nodos do Grafo** - Onde mostramos todas as posições possíveis que os jogadores poderão explorar.
5. **Aplicar algoritmos aos carros** - Esta opção permite ao utilizador escolher qual algoritmo de procura cada carro irá utilizar para encontrar o caminho até à meta.

Haverá 4 opções de algoritmos de procura, 2 algoritmos de procura não informados e 2 algoritmos de procura informada.

```
**** Algoritmo para o jogador 1 ****
1-DFS
2-BFS
3-Greedy
4-A*
Introduza a sua opcão:
```

Figura 3: Opções de procura.

O programa irá armazenar os algoritmos escolhidos para cada jogador e executá-los assim que todos os jogadores tenham escolhido o seu algoritmo.
De seguida os algoritmos irão ser aplicados aos respetivos jogadores e será armazenado num objeto **InfoCaminho** o caminho final encontrado pelo algoritmo de procura e a lista dos nodos visitados pelo algoritmo.
Essa informação é imprimida no terminal de forma a comparar o caminho percorrido pelo algoritmo e o caminho final encontrado:

```
Expansão dos nós com o algoritmo A* do jogador 1: (lenght = 13)
(2,1) -> (2,2) -> (2,3) -> (2,4) -> (3,1) -> (4,2) -> (4,3) -> (4,4) -> (4,5) -> (4,6) -> (4,7) -> (4,8) -> (4,9)
Caminho final do jogador 1 com o algoritmo A*: (lenght = 10)
(2,1) -> (3,1) -> (4,2) -> (4,3) -> (4,4) -> (4,5) -> (4,6) -> (4,7) -> (4,8) -> (4,9)
```

Figura 4: InfoCaminho.

O caminho final encontrado pelo algoritmo é de seguida mostrado no terminal dentro do circuito onde está representado pelas letras do alfabeto.
O caminho é representado pelas letras do alfabeto e por cores onde:

- Vermelho - posição inicial.
- Azul - Posição final.
- O percurso terá outras cores onde serão diferentes para diferentes jogadores.

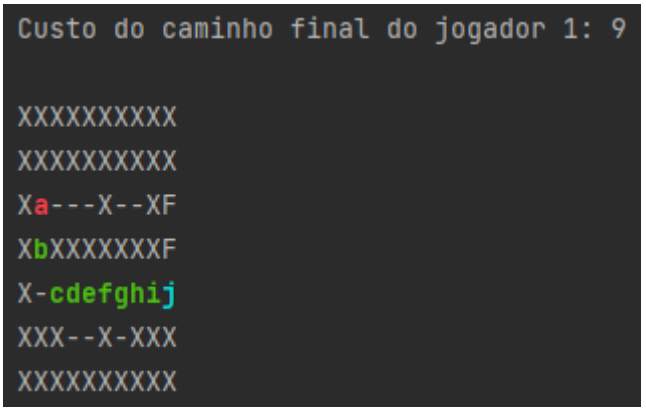


Figura 5: Caminho final.

No caso de existir mais do que um jogador no circuito, o programa irá representar os caminhos dos jogadores individualmente, e no fim, o circuito com os caminhos de todos os jogadores participantes na corrida. A ordem de chegada dos jogadores é calculada conforme o custo dos caminhos dos jogadores e apresentada também no terminal.

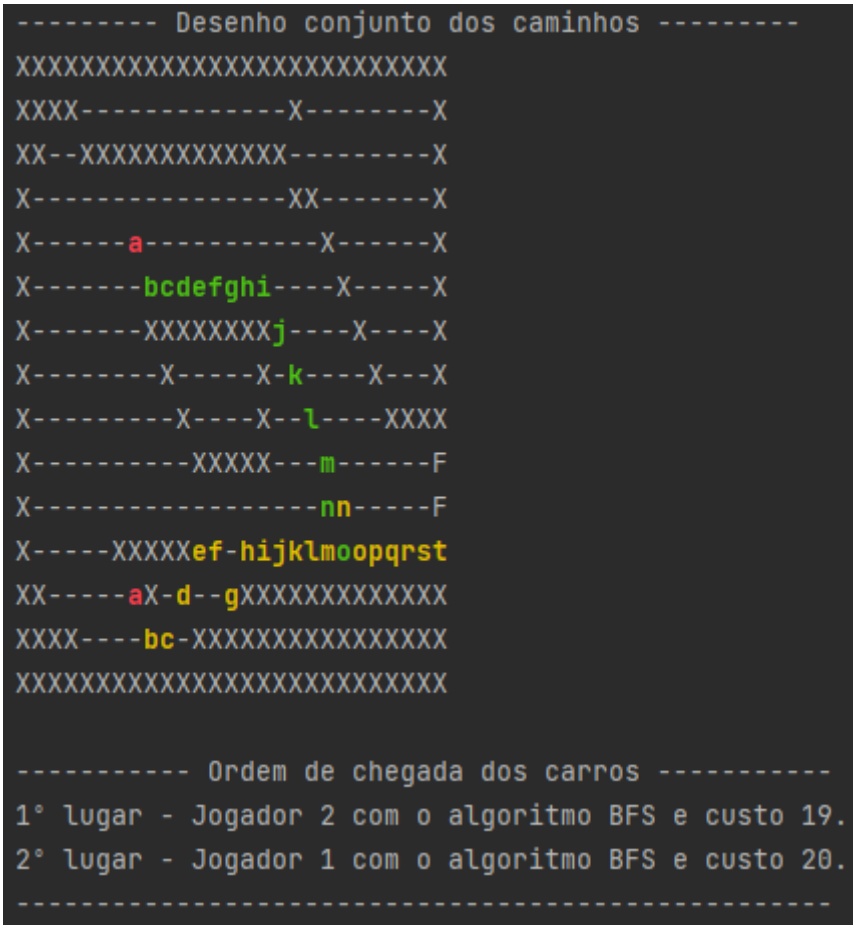


Figura 6: Desenho conjunto dos caminhos finais.

4.3. Deteção de colisões

Em certas ocasiões em ambiente competitivo poderá ocorrer casos onde o caminho encontrado para 2 jogadores intersetam na mesma posição no mesmo instante. Nesses casos é feita a verificação de qual jogador possui a menor velocidade, e esse jogador terá de fazer uma alteração no seu percurso para evitar a colisão.
O desvio no caminho é obtido através de um pequeno algoritmo BFS, que calcula a melhor rota possível evitando a posição de colisão, com o começo na posição anterior e destino na posição seguinte à colisão. Esta verificação é feita até que no final dos caminhos não haja colisões em nenhuma iteração. Decidimos usar o BFS pois é um algoritmo bom para pequenos caminhos e ao mesmo tempo simples o suficiente para que seja possível o desvio dos carros.

5. Estratégias de procura não informada

As estratégias de procura não informadas implementadas neste projeto foram a **Procura em profundidade (DFS)** e a **Procura em largura (BFS)**.

5.1. Procura em profundidade (DFS)

Este tipo de procura usa como estratégia, expandir sempre um dos nodos mais profundos do grafo.
As **vantagens** deste tipo de procura são:

- Pouco uso de memória.
- Bom para problemas com múltiplas soluções, pois a probabilidade de estar a procurar por um caminho possível aumenta.

A **desvantagem** deste tipo de procura são:

- Pouca eficiência em grafos com uma profundidade elevada e poucas soluções.
- A solução obtida pode não ser a solução ótima.

No entanto, no nosso problema o circuito terá vários caminhos possíveis para chegar ao destino final, logo esta estratégia irá conseguir obter um percurso válido e de forma relativamente eficiente.
A nossa implementação do algoritmo de procura em profundidade impede que os nodos que já tenham sido visitados sejam explorados pelo algoritmo de maneira a evitar a ocorrência de loops no processo de pesquisa.

Esta estratégia de procura apresenta as seguintes propriedades:

- Tempo: $O(b^m)$
- Espaço: $O(bm)$

Onde **b** é o número máximo de sucessores de um nodo da árvore de procura e **m** a máxima profundidade do espaço de estados.

5.1.1. Exemplo de uma procura em profundidade (DFS)

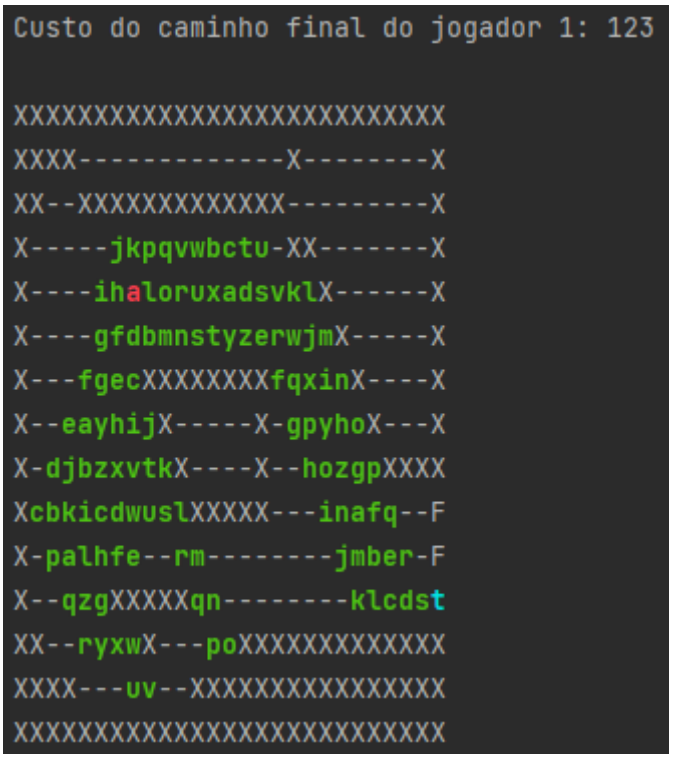


Figura 7: Exemplo de uma procura em profundidade.

Como podemos ver na figura acima, o algoritmo de procura em profundidade obteve um caminho não ideal devido à sua estratégia de expandir sempre para nodos mais profundos do grafo, não sendo a melhor opção para este tipo de problemas uma vez que existem ramificações bastante longas que não vão ao encontro de uma solução ótima.

5.2. Procura em largura (BFS)

Este tipo de procura expande primeiro os nodos de menor profundidade do grafo.
As **vantagens** deste tipo de procura são:

- Solução ótima quando todas as arestas têm custo 1.

As **desvantagens** deste tipo de procura são:

- Tempo de pesquisa elevado visto que tende a percorrer muitos mais nodos do que os necessários para criar o caminho válido.
- Ocupa muito espaço em memória.

No nosso problema o tamanho do grafo irá depender do tamanho do circuito logo este tipo de procura irá tornar-se mais lento para circuitos de grandes dimensões.

Esta estratégia de procura apresenta as seguintes propriedades:

- Tempo: $O(b^d)$
- Espaço: $O(b^d)$

Onde **b** é o número máximo de sucessores de um nodo da árvore de procura e **d** a profundidade da melhor solução.

5.2.1. Exemplo de uma procura em largura (BFS)

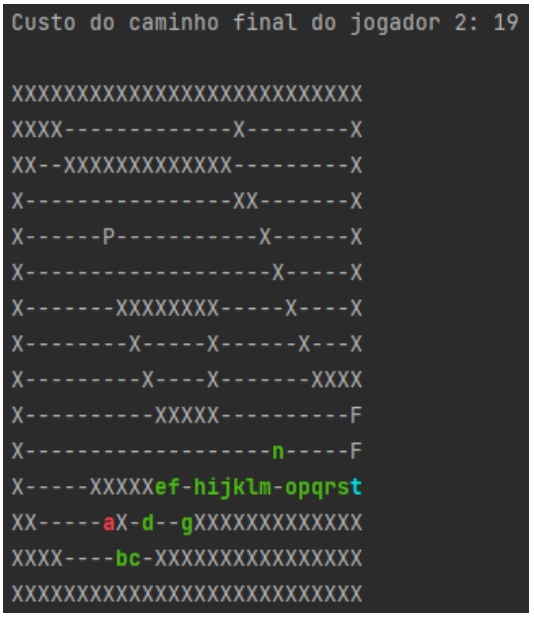


Figura 8: Exemplo de uma procura em largura.

Como podemos ver na figura acima, o algoritmo de procura em largura obteve um caminho ótimo uma vez que no nosso problema todas as arestas têm um custo igual a 1, o que é a melhor circunstância possível para a utilização deste algoritmo, visto que um grafo com esta característica permite ao algoritmo BFS encontrar sempre o caminho mais curto devido à estratégia de expansão dos nodos de menor profundidade em primeiro.

6. Estratégias de procura informada

As estratégias de procura informada implementadas neste projeto foram a **Procura Gulosa (Greedy)** e a **Procura A***. Numa primeira iteração do desenvolvimento das estratégias de procura informada, usámos a heurística estática da **distância de Manhattan** entre os nodos do grafo e os nodos finais. As heurísticas para todos os nodos eram calculadas após a criação do grafo e não eram alteradas durante a procura no grafo.
A **distância de Manhattan** entre 2 nodos é calculado da seguinte forma:
$Nodo\ final:\ (x1, y1)$
$Nodo\ atual:\ (x2, y2)$
$h(n)\ = |x1\ -\ x2|\ +\ |y1\ -\ y2|$
No nosso problema existe a possibilidade de existirem vários nodos finais por isso o valor da heurística é a menor distância entre o nodo atual e um nodo final.
Esta heurística permitiu testarmos a implementação das estratégias de procura que implementámos.

6.1. Heurística

Por forma a tornar a heurística dinâmica tivemos de fazer algumas alterações na forma como calculamos as heurísticas nos nodos.
As heurísticas agora são calculadas durante a pesquisa no grafo e são influenciadas pela **aceleração** e **velocidade** do carro.
A **aceleração** do carro é calculada segundo a próxima posição do carro:
$aceleração\ =\ next\_node\ -\ current\_node$
A **velocidade** do carro é armazenada durante a procura no grafo e é calculada a partir da aceleração:
$velocidade\ =\ velocidade\ +\ aceleração$
Será de notar que tanto a **aceleração** e a **velocidade** são pares (x,y) pois os nossos circuitos possuem 2 dimensões.

De seguida calculamos o vetor **distância** para a posição final mais próxima fazendo essa verificação usando a **distância euclidiana** entre os 2 nodos.
A **distância euclidiana** é obtida a partir da seguinte função:
$Nodo\ atual:\ (x1, y1)$
$Nodo\ objetivo:\ (x2, y2)$
$distância\ = \sqrt{(x1\ -\ x2)^2\ +\ (y1\ -\ y2)^2}$
A **distância** então será calculada da seguinte forma:
$Nodo\ atual:\ (x1, y1)$
$Nodo\ objetivo\ mais\ próximo:\ (x2, y2)$
$distância\ =\ (x2\ -\ x1, y2\ -\ y1)$

Com a **velocidade** e a **distância** calculadas podemos então calcular a heurística do nodo calculando o tempo estimado até ao final com:
$tempo\ estimado = distância\ /\ velocidade$
O tempo estimado é um tuplo com os valores (tempoX,tempoY).
E então por forma a obter um valor singular fazemos:
$tempo\ estimado = tempoX + tempoY$

Esta heurística foi escolhida pois relaciona 2 aspetos importantes numa corrida, a distância até à meta e a velocidade do carro, se a distância for menor o tempo estimado até ao final diminui e se a velocidade for elevada o tempo estimado também diminui, e com isso os algoritmos de procura informada conseguem ter uma boa heurística para fazer as suas decisões.

6.2. Procura Gulosa (Greedy)

Este tipo de procura expande para o nodo que parece estar mais perto da solução utilizando para isso uma heurística.
As **vantagens** deste tipo de procura são:

- Tempo de procura reduzido se a função heurística for boa

As **desvantagens** deste tipo de procura são:

- A solução obtida pode não ser a solução ótima.
- Ocupa muito espaço em memória.

No nosso problema a eficácia da função heurística irá depender do circuito pois parte do seu cálculo tem em conta a distância até à meta e no caso de existirem paredes entre os nodos e a meta poderá levar a uma escolha errada do próximo nodo a ser explorado, e com isso a performance ficaria afetada.

Esta estratégia de procura apresenta as seguintes propriedades:

- Tempo: $O(b^m)$ (com uma boa função heurística pode diminuir)
- Espaço: $O(b^m)$

Onde **b** é o número máximo de sucessores de um nodo da árvore de procura e **m** a máxima profundidade do espaço de estados.

6.2.1. Exemplo de uma procura gulosa (Greedy)

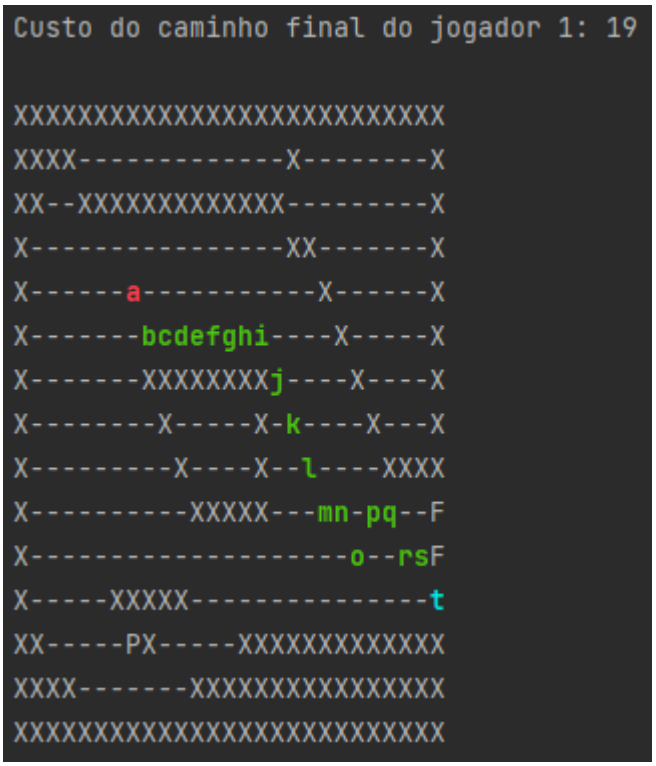


Figura 9: Exemplo de uma procura gulosa (Greedy).

Como podemos ver na figura acima, o algoritmo de procura gulosa encontrou o caminho ideal até à meta devido à heurística utilizada, e a ausência de obstáculos na direção da meta. O algoritmo para estes casos é bastante eficiente pois não explora tantos nodos como o algoritmo BFS e encontra boas soluções.

6.3. Procura A*

Este tipo de procura evita expandir caminhos que são dispendiosos, combinando para isso o algoritmo de procura gulosa com o algoritmo de procura uniforme.
Utiliza então a seguinte função para a escolha do nodo a ser explorado de seguida:
$f(n) = g(n) + h(n)$
Onde:

- g(n) = custo acumulado
- h(n) = custo estimado para chegar ao destino (heurística)

As **vantagens** deste tipo de procura são:

- Obtém a solução ótima se a sua heurística for admissível.

As **desvantagens** deste tipo de procura são:

- Ocupa muito espaço em memória.

No nosso problema a eficácia da heurística irá depender do circuito como já foi explicado na análise da Procura Gulosa.
No entanto este algoritmo de procura para a escolha do próximo nodo, analisa a heurística e o custo acumulado do seu percurso até ao momento. Isso permite ao algoritmo encontrar a solução ótima para o problema.

Esta estratégia de procura apresenta as seguintes propriedades:

- Tempo: $O(b^m)$ (com uma boa função heurística pode diminuir significativamente)
- Espaço: $O(b^m)$

Onde **b** é o número máximo de sucessores de um nodo da árvore de procura e **m** a máxima profundidade do espaço de estados.

6.3.1. Exemplo de uma procura A*

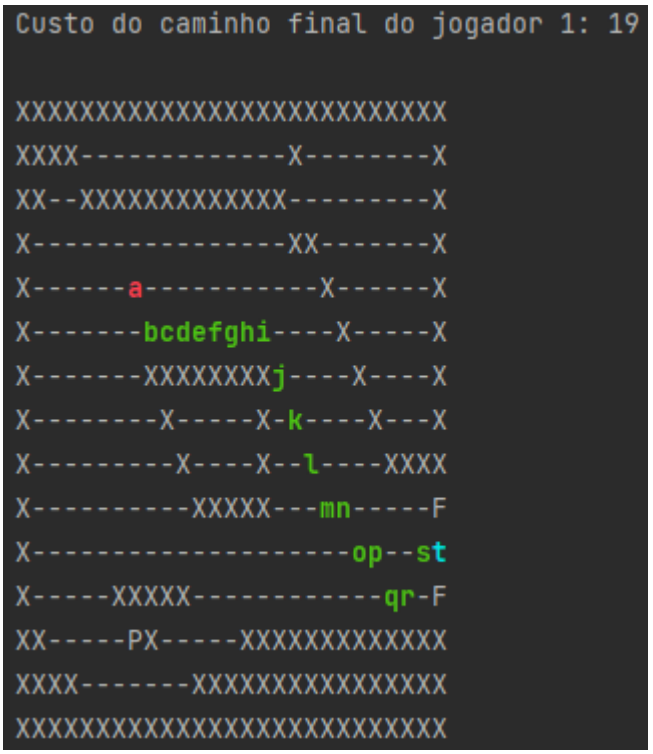


Figura 10: Exemplo de uma procura A*.

Como podemos ver na figura acima, o algoritmo de procura A* obteve uma solução ótima ao problema de procura, devido à heurística utilizada e o armazenamento dos custos acumulados.

7. Conclusão

Em suma, neste projeto o nosso grupo desenvolveu um programa que dado um circuito fornecido em formato de ficheiro, é capaz de criar o grafo de posições possíveis, possibilitando a visualização desse grafo de forma gráfica, mas também mostrar os seus nodos e as respetivas arestas. Mais importante, permite utilizar algoritmos de procura para explorar caminhos possíveis e atingir uma solução, tendo sido descritas todas as vantagens e desvantagens de cada estratégia utilizada.
O programa também possui um modo competitivo onde podemos ter uma comparação direta entre os algoritmos de procura. A implementação desse modo também trouxe consigo outros desafios como a deteção de colisões e como as evitar.
Finalmente, gostaríamos de referir que, quanto à heurística escolhida para os algoritmos de procura informada, tivemos alguns problemas de início a entender como incluir a velocidade do carro no seu cálculo e como isso afetaria o nosso problema. No entanto conseguimos chegar a conclusão que a melhor forma seria juntar o cálculo da distância até à meta, com a velocidade atual do carro e assim, calcular o tempo estimado até à meta.