Universidade do Minho
Escola de Engenharia

Universidade do Minho
Licenciatura em Eng.ª Informática
Grupo 7

INTELIGÊNCIA ARTIFICIAL

AFONSO MARQUES - 94940
PEDRO OLIVEIRA - 98712
TOMÁS DIAS - 89486
VASCO RITO - 98728

Braga, Novembro de 2022

[27 de março de 2024 at 22:47 –]

ÍNDICE

LISTA DE FIGURAS

1

DESCRIÇÃO DO PROBLEMA

Para a unidade curricular de Inteligência Artificial foi proposta a realização de um jogo como parte de avaliação da componente prática.

Deste modo, é pretendido que se desenvolvam diversos algoritmos de procura para a resolução de um jogo de corridas. O VectorRace é um jogo de simulação de carros simplificado que contém um conjunto de movimentos e regras associadas.

Assim, de modo a concretizar a objetividade do problema, é suposto que o corredor chegue à meta o mais rápido possível.

Por se tratar de um ambiente não determinístico considera-se ser um problema de contingência. O corredor perceciona o ambiente à sua volta e com base nas informações que recolhe atualiza o seu estado, ou seja, procura e de seguida executa a sua jogada.

Quanto ao circuito VectorRace gerado, este tem como ponto de partida a letra 'P', linha de meta a letra 'F', limites de pista a letra 'X' e como pista onde se pode efetivamente conduzir, o hífen '-'. Apresenta ainda dois corredores que irão competir entre si para ver qual dos dois o primeiro a chegar à meta. Ambos os corredores não se conseguem sobrepor, ou seja, ocupar a mesma posição na pista.

Abaixo, observam-se dois circuitos escolhido para usar no trabalho prático.

Após o circuito ter sido escolhido falta agora o seu desenho formal. Para isso foram usadas as bibliotecas usadas na aula 'networkx' e 'matplotlib.pyplot'. O *layout* escolhido para desenhar o grafo, foi o *spring_layout* devido a ser aquele em que foi conseguida uma melhor visualição em termos de ligação dos nodos.

Assim, depois de algumas tentativas, o desenho dos grafos encontram-se nas imagens abaixo.

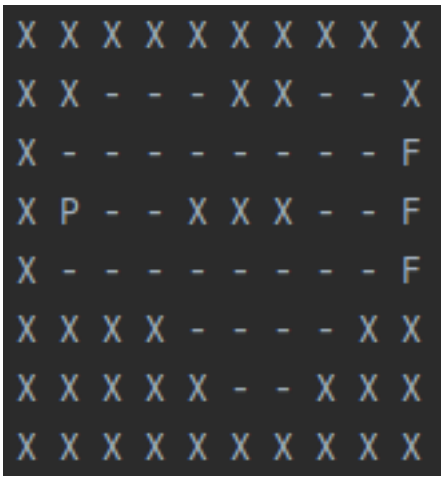

Figura 1: Circuito 1 - Vector Race

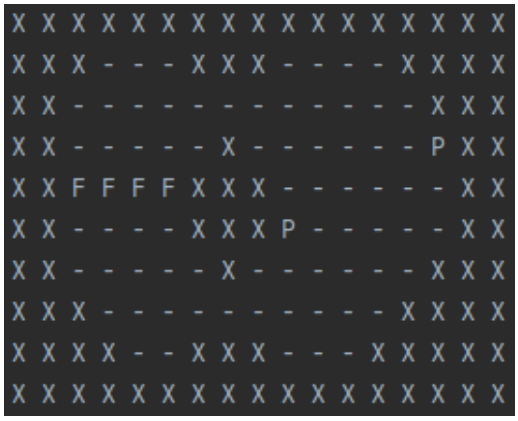

Figura 2: Circuito 2 - Vector Race

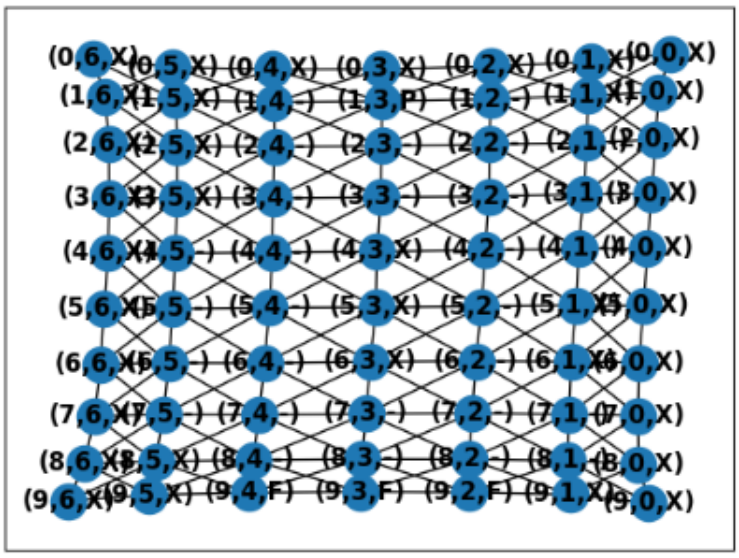

Figura 3: Desenho do grafo do Circuito 1

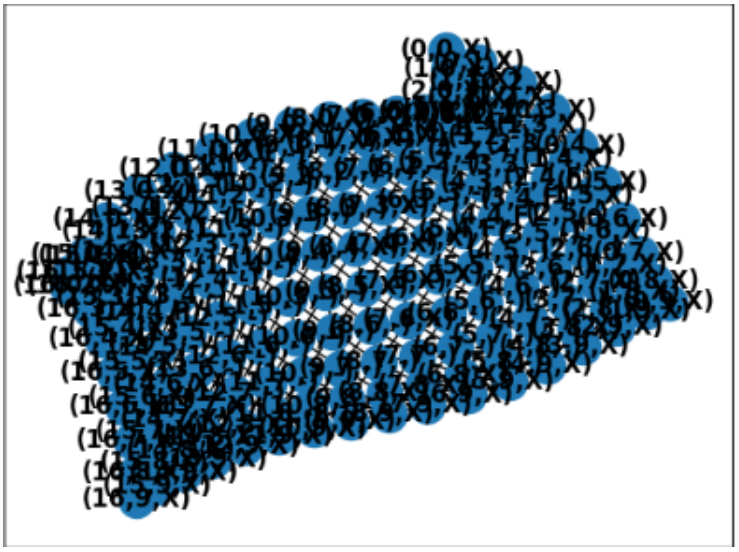

Figura 4: Desenho do grafo do Circuito 2

2

FORMULAÇÃO DO PROBLEMA

Estado Inicial: Carros começam na linha de partida da pista com velocidade 0 e aceleração 0.
Estado Objetivo: Chegar à linha da meta.
Ações:

- Acelerar carro:
  - **Pré-Condições:** N/a
  - **Efeitos:** O carro altera por +1, 0 ou -1 a aceleração no vetor l (representa o vetor linha) e no vetor c (representa o vetor coluna). Isto faz com que a velocidade e posição dele mude em conformidade.
- Retornar à Pista:
  - **Pré-Condições:** Carro Saiu da Pista
  - **Efeitos:** O carro volta para a sua posição anterior, assumindo um valor de velocidade e aceleração zero.

Custo da Solução: Cada movimento de um carro numa determinada jogada, de uma determinada posição para outra, terá um custo de 1 unidade, sendo que quando o mesmo sair dos limites da pista, o custo é de 25 unidades.

3

RESOLUÇÃO DO PROBLEMA

A resolução do problema exposto pelo enunciado partiu da divisão do mesmo em partes. Um dos aspetos mais relevantes foi a criação de classes específicas que sejam representativas dos componentes do VectorRace.

Assim, foram criadas as seguintes classes:

- **UI** - constitui a interface textual para o utilizador poder interagir com o VectorRace. Aqui é possivel escolher de entre várias opções, como por exemplo imprimir o circuito ou escolher qual algoritmo vai ser utilizado para realizar uma corrida.
- **Corredor** - classe que implementa a lógica responsável pela gestão de um corredor. É aqui que se definem métodos que permitem calcular a velocidade e aceleração de um corredor.
- **Graph** - classe correspondente ao grafo que incorpora todos os algoritmos de pesquisa que são utilizados neste programa.
- **Pos** - classe que tem como propósito representar a posição/nodo num grafo.

Após as classes terem sido criadas o grupo partiu para a escolha dos algoritmos que irão compor o projeto.

3.0.1 *Algoritmos 1º Etapa*

3.0.1.1 Depth First Search *(DFS)*

Inicialmente optou-se pelo algoritmo de procura em profundidade - *Depth First Search* (DFS) - que utiliza pouca memória e consiste em expandir sempre um dos nós mais profundos da árvore de procura que representa o circuito. No entanto os resultados obtidos não foram os melhores em termos de custo relativamente ao BFS ou algoritmos informados.

De modo a tentar melhorar os resultados obtidos, foi utilizada uma ordenação para as posições adjacentes de modo a incentivar o carro a ir para "a frente"e não

sair da pista. No entanto, o método utilizado apresenta um problema aquando o início e fim da pista se encontram verticalmente opostos.

Foi notório que o percurso do vector no *DFS* fica caro para pistas de grande escala, o que não é ideal num cenário de corrida. Este mau resultado pode ser explicado pelo facto de o algoritmo de DFS ser pouco eficaz para árvores muito profundas, como é o caso da utilizada. Foi ponderado fazer alterações ao algoritmo como, por exemplo, introduzir limites, passando assim a ter um algoritmo de procura iterativo. Infelizmente os resultados, após testes, não foram significativamente diferentes quando comparado com o *DFS* tradicional, usando ordenação dos nós adjacentes.

3.0.1.2 Breadth First Search *(BFS)*

Adicionalmente, foi implementado outro algoritmo de pesquisa não informada, desta vez o de procura em largura - *Breadth First Search* (BFS) - que tem como lógica expandir todos os nós de menor profundidade primeiro. Como se trata de uma procura muito sistemática, os resultados obtidos foram os mais satisfatórios nesta fase. O vector passou a ser capaz de percorrer o circuito num caminho que utiliza muito poucos nós tendo por isso um menor valor de custo.

Após terem sido testados os três algoritmos (DFS, Iterativa e BFS), nos dois circuitos escolhidos, foi então decidido que estes seriam os algoritmos a manipular nesta versão do VectorRace. Para esse efeito foram criados três modos de corrida: o modo DFS, o modo Iterativo e o modo BFS. Todos permitem ao utilizador escolher qual o circuito que vai ser percorrido. No final de cada modo será imprimido no terminal uma representação visual do circuito na qual é possivel ver o caminho feito pelo corredor, através de números, para chegar a um dos objetivos finais (reresentados por 'F' no circuito).

3.0.2 *Algoritmos adicionados na 2ª Etapa*

Com a realização da segunda etapa do trabalho prático foram então implementados os algoritmos de modo a que haja, simultaneamente, dois corredores na pista.

Adicionalmente, foram implementados outros dois algoritmos informados bastante conhecidos. É o caso do algoritmo Guloso e A*. A heurística utilizada para estes algoritmos foi calculada através da distância, em linha reta, entre a posição do ponto no grafo e as possíveis posições objetivo. Para alem disso também são descartados os nós em que o corredor fica com uma velocidade superior a distancia da meta em linha reta.

3.0.2.1 *Algoritmo Guloso*

O Algoritmo Guloso (*Greedy*) opta por, a cada iteração, decidir pelo nó/posição que tem menor custo na esperança que esta escolha leve à melhor solução possível. No entanto, este algoritmo não garante uma solução ótima e necessita a deteção de estados repetidos para evitar que entre num ciclo (*loops*). De qualquer forma, este é um algoritmo de fácil implementação e rápida execução.

Para a implementação com múltiplos corredores na mesma pista foi ainda preciso restringir o algoritmo para que não escolhesse um estado em que dois corredores ocupassem a mesma posição em simultâneo. No entanto, isto aumenta a possibilidade para que o algoritmo não encontre uma solução, quanto maior o número de corredores.

3.0.2.2 *Algoritmo A**

Foi ainda implementado o Algoritmo A*, sendo um algoritmo que procura um caminho entre um nó/posição inicial e final, ambos fornecidos. Começa por se aplicar ao nó inicial um custo, neste caso 0, e de seguida estima-se a distância até ao nó final em linha reta. Finalmente, utiliza a avaliação heurística para percorrer o grafo, minimizando a soma do caminho já efetuado com o mínimo previsto do que falta até à solução. Este algoritmo de procura fornece uma solução ótima e é completo, no entanto, tem um custo de tempo exponencial e ocupa mais espaço em memória que o Guloso.

3.0.3 *Soluções do Circuito 2 com os diversos Algoritmos*

Assim, as imagens seguintes ilustram as diferenças entre as soluções dos algoritmos. Em especial, o facto dos algoritmos informados apresentarem melhores soluções que os algoritmos não informados. De notar os números representados nas figuras (0 a 9), retratando o caminho percorrido por cada corredor e a diferença no custo entre eles.

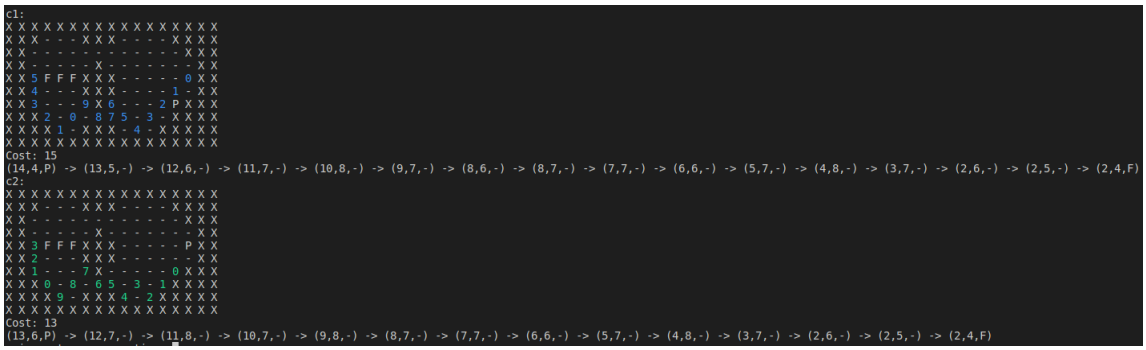

Figura 5: Circuito 2 com DFS

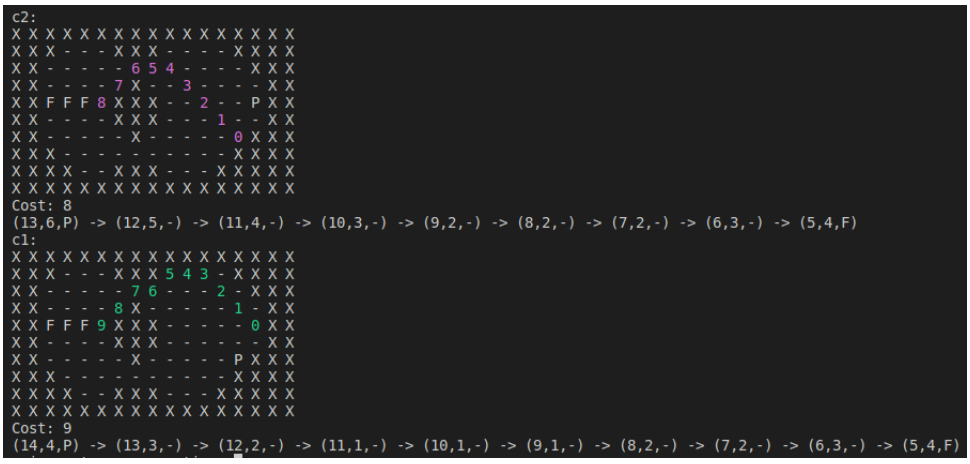

Figura 6: Circuito 2 com BFS

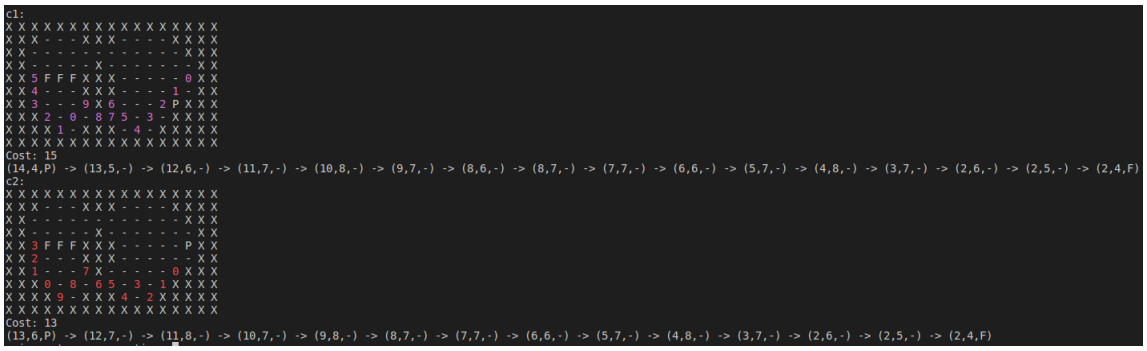

Figura 7: Circuito 2 Iterativa

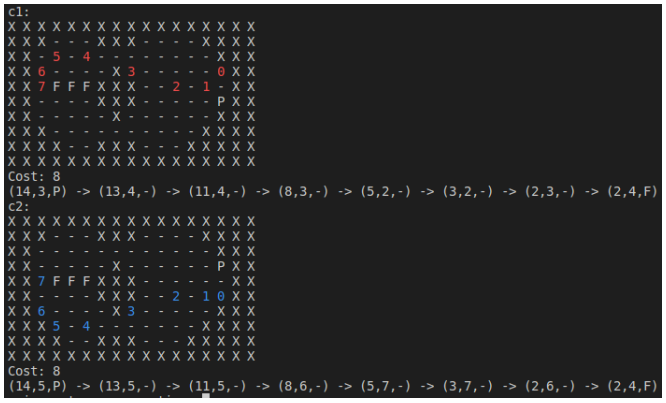

Figura 8: Circuito 2 Guloso

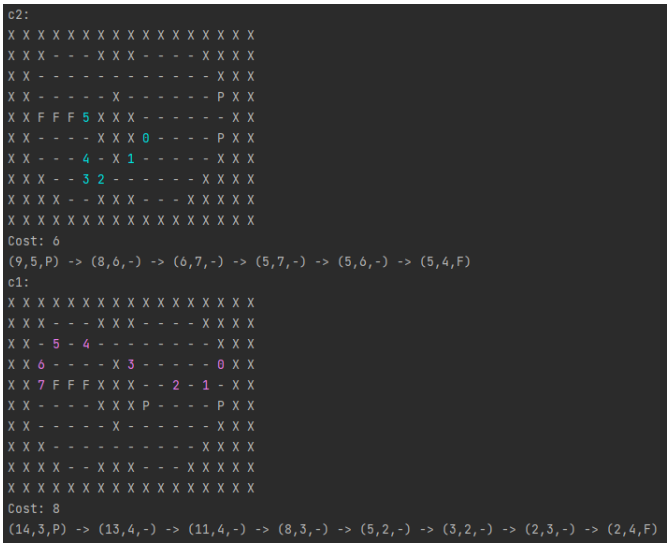

Figura 9: Circuito 2 A*

4

CONCLUSÕES

Com a conclusão das diversas tarefas solicitadas nesta etapa do trabalho prático, o grupo adquiriu um maior conhecimento relativo às dificuldades em escolher os algoritmos corretos para este tipo de problema, bem como a noção da importância na escolha de uma boa heurística e ordenação de nodos adjacentes para os algoritmos que as utilizam.

Um exemplo destes problemas é o facto de se ter implementado um método de procura Iterativo. Após os testes executados, este não se revelou mais benéfico que método de procura BFS, utilizando uma melhor ordenação. Deu também para chegar à conclusão de que os métodos de procura informados apresentam melhores soluções para este contexto de problemas.