



**Universidade do Minho**  
Escola de Engenharia

# Inteligência Artificial

**Trabalho Prático**

**Grupo 26**

**2022/2023**



A91671 - João Manuel Novais da Silva



A91697 - Luís Filipe Fernandes Vilas

# Índice

## Conteúdo

1. Introdução .....	3
2. Descrição do problema .....	4
3. Realização do Trabalho .....	5
3.1. Requisitos .....	5
3.3. Execução do Jogo .....	6
3.4. Algoritmos utilizados.....	7
3.4.1. DFS.....	7
3.4.2. BFS .....	7
3.4.3. A* .....	8
3.4.4. Greedy .....	9
4. Conclusão .....	10

# 1. Introdução

O principal motivo deste trabalho é encontrar uma solução para obter, num melhor tempo e de uma maneira mais eficaz, o caminho mais curto para o mesmo.

É nos apresentado um ficheiro, por exemplo *circuito.txt* que dispõe de várias coordenadas, as quais podem ser – (caminho utilizável), X (parede que não pode ser atravessada), P (posição inicial) e, por fim F (destino).

O F (destino) pode ser composto por vários pontos, ou seja, é possível que existam vários destinos finais.

## 2. Descrição do problema

No âmbito do trabalho, tivemos de definir padrões para aceitar o VectorRace como um jogo, em que tivemos de definir o que seria cada uma das suas componentes: o mapa é tratado como sendo um Grafo, onde cada posição da matriz é um nodo e o percurso do dito carro é um conjunto de nodos desse grafo.

Esse mesmo carro, deveria corresponder à “realidade”, ou seja, deveria aceitar velocidades e acelerações.

O nosso programa calcula a posição através do Grafo, utilizando os diferentes algoritmos por nós implementados. Ou seja, no final da execução deveremos ter a posição final e no seu caminho ideal (caminho mais curto).

Por fim e já nesta segunda fase, é nos requisitado que o jogo suporte mais que um jogador, ou seja, que seja multiplayer. Esta parte implica que o jogo calcule, ao mesmo tempo o melhor caminho de ambos os jogadores, sem a existência de colisões.

## 3. Realização do Trabalho

### 3.1. Requisitos

Para a execução do jogo, foram utilizadas diversas bibliotecas do Python entre as quais:

- Quele
- Networkx
- matplotlib.pyplot

Estas bibliotecas são rigorosamente obrigatórias para que o programa execute e assim sendo, termos o total desempenho da aplicação desenvolvida.

### 3.2. Mapas

Para a rápida execução do programa, precisamos de ter circuitos.

```
XXXXXXXXXX---X
XXP--XX--X-XXX
X-----X
X---XXX-----X
X---F-----F
XXXX-----F--XX
XXXXXXXXXXXXXX
```

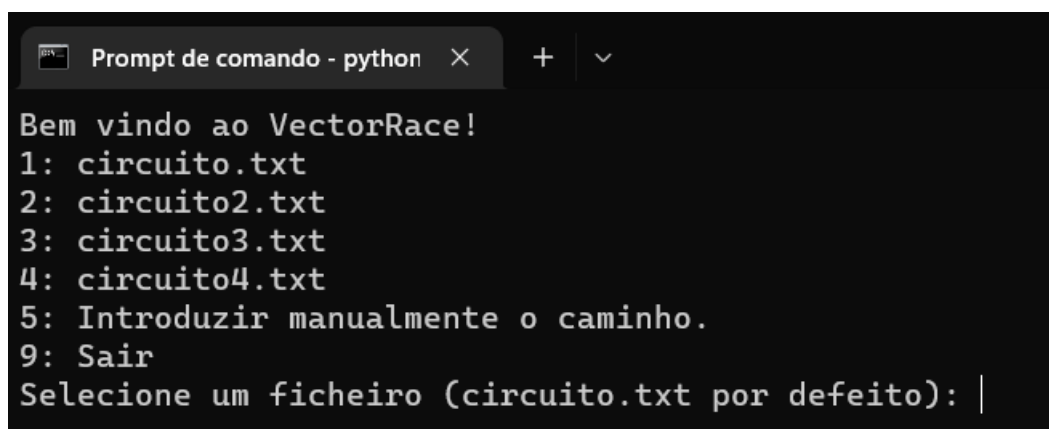
*Figura 1 - Exemplo de Ficheiro de Circuito*

Esses circuitos são lidos localmente, ou seja, estão contidos em ficheiros escritos para o propósito (Figura 1). Esse ficheiro possui essencialmente **X** – parede, **F** – posição final do jogador, **P** – posição inicial do jogador, - - caminho passível de ser percorrido.

### 3.3. Execução do Jogo

O programa inicia – se com a leitura do mapa, que se encontra num ficheiro de texto. O mesmo só é interpretado se o mapa presente neste for validado pelo programa. Através do mesmo conseguimos popular o grafo, ou seja, a cada célula é gerado um nodo no Grafo e, posteriormente, a cada nodo são feitas as ligações aos seguintes, ou seja, as arestas.

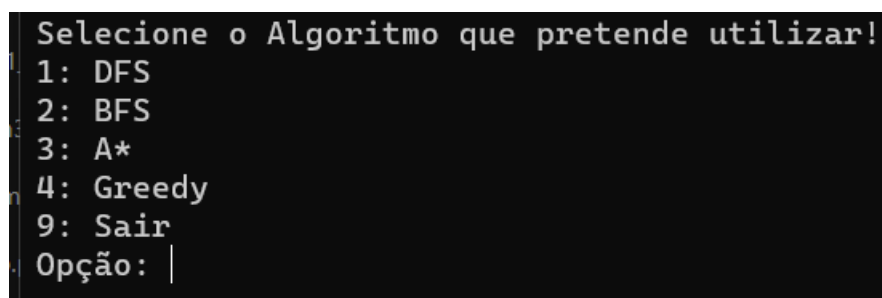
Posteriormente é apresentado ao jogador um menu de jogo onde é possível escolher o circuito ao qual pretendemos fazer os testes (Figura 2).



```
Prompt de comando - python  X  +  v
Bem vindo ao VectorRace!
1: circuito.txt
2: circuito2.txt
3: circuito3.txt
4: circuito4.txt
5: Introduzir manualmente o caminho.
9: Sair
Selecione um ficheiro (circuito.txt por defeito): |
```

Figura 2 - Menu de Jogo

Ao fazermos a seleção do mapa, somos presenciados com outro menu (Figura 3) que nos permite escolher o algoritmo para a locomoção do carro ao longo do mapa, ou seja, as diferentes formas para ser encontrado o melhor caminho até ao destino final.



```
Selecione o Algoritmo que pretende utilizar!
1: DFS
2: BFS
3: A*
4: Greedy
9: Sair
Opção: |
```

Figura 3 - Seleção de Algoritmo

Por fim, é nos apresentado o mapa inicial (pré-travessia), o mapa final, com o caminho percorrido e o custo da operação (Figura 4).

```

ALGORITMO DFS
A coordenada de inicial é: [(2, 1)]
As coordenadas de fim são: [(4, 4), (13, 4), (9, 5)]
Mapa antes da travessia:

#####---#
##P--##--#-###
#-----#
#---###-----#
#---F-----F
####-----F--##
#####

Mapa após travessia:

#####---#
##X--##--#-###
#-X-----#
#-X-###-----#
#-XXX-----F
####-----F--##
#####

O custo desta operação foi: 4

```

Figura 4 - Apresentação Final

### 3.4. Algoritmos utilizados

#### 3.4.1. DFS

Foi implementado o algoritmo DFS, com base nos algoritmos desenvolvidos nas aulas práticas. O mesmo possui uma lista de nodos visitados, que vai populando a cada recursividade. Essa mesma lista não permite que o mesmo nodo seja visitado mais do que uma vez.

Além do algoritmo base utilizado nas aulas, foi adicionada uma verificação adicional, que permite saber se o nodo é ou não uma parede.

#### 3.4.2. BFS

O algoritmo BFS foi, tal como o anterior, muito similar ao que foi demonstrado nas aulas práticas, sem grandes adições.

### 3.4.3. A\*

O trabalho prático apresenta 2 definições do algoritmo A\*, uma com paciências ao código definido nas aulas. A outra foi desenvolvida pelos alunos.

Obviamente será útil explicar a resolução dos alunos. Resumindo, a ideia que os alunos tiveram foi definir a função que irá ditar os custos ou “scores” de cada nodo, esta função é a seguinte:

$$F(n) = G(n) + H(n)$$

Onde  $G(n)$  é a distância entre o nodo inicial e o nodo atual, e  $H(n)$  é a heurística do nodo atual. O raciocínio dos alunos foi que, temos uma tabela que para cada nó, tem o valor de  $G(n)$ ,  $H(n)$ ,  $F(n)$  e o seu nodo pai. As funções  $F$ ,  $G$  e  $H$  são definidas por dicionários em que as keys são os próprios nodos e os values os valores. Inicialmente os dicionários de  $f$  e  $g$  são inicializados a infinito e o  $h$  inicializado com as distâncias ao nodo final usando a medida de Manhattan. A decisão do próximo nodo a ser visitado depende do valor atual das funções para o dado nodo. Com isto é possível percorrer o mapa passando pelos nodos com melhores valores de  $F(n)$ .



#### 3.4.4. Greedy

O algoritmo Greedy, tem uma implementação com base nos algoritmos desenvolvidos nas aulas. Este algoritmo funciona de maneira parecida ao A\*, mas neste caso apenas usa as heurísticas para se orientar pelo melhor caminho possível.

## 4. Conclusão

Para terminar, conseguiu-se desenvolver uma aplicação, na nossa perspetiva, capaz de implementar diversos algoritmos de procura para um determinado problema e, dessa forma encontrar o melhor caminho para o problema em questão.

Ao longo do desenvolvimento fomos presenciados com diversas dificuldades, entre as quais o facto de o mapa conter paredes. Na nossa perspetiva inicial as paredes teriam de pertencer ao grafo, algo que não se verificou na fase final do trabalho, pois estes não seriam caminhos possíveis onde o jogador poderia atravessar.

Por outro lado, também os finais dos mapas se revelaram um problema na realização do exercício, pois estes não poderiam ser ultrapassados, algo que se verificou ser um problema inicial, mas, com algum estudo, foi uma tarefa de fácil resolução.

Por fim, o grupo revelou-se interessado e empenhado na realização da tarefa proposta para esta UC. Apesar disso, o trabalho ainda poderia ser melhorado e, poderiam existir implementações adicionais, que trariam bastante funcionalidade para a aplicação em geral.