# Inteligência Artificial

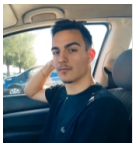
## Relatório do Trabalho Prático - Fase 1

LEI - 2022/2023

## Descrição do Desenvolvimento e Resultados Obtidos

Grupo 15


Rui Chaves A83693 

Maurício Pereira A95338 

Afonso Bessa A95225 

Martim Ribeiro A96113 



*Universidade do Minho*

# Índice

# 1 Introdução

No âmbito da Unidade Curricular de **Inteligência Artificial** foi proposto o desenvolvimento, conceção e implementação de diversos Algoritmos de Procura para a resolução do jogo *VectorRace*.

O seguinte relatório irá retratar e expor algumas tomadas de decisão, bem como, objetivos superados e acontecimento menos conseguidos durante a resolução deste Projeto.

## 2 Descrição do Problema

O *VectorRace* é um jogo de simulação de carros simplificado, que contém um conjunto de movimentos e regras associadas.

O movimento do carro no *VectorRace* é bastante fácil e elementar, uma vez que, as ações desenvolvidas são equivalentes a um conjunto de Acelerações.

O conjunto de Acelerações possível é **-1**, **0** ou **1**, ou seja, considerando a notação **l** que representa a linha e **c** a coluna para os vetores, num determinado instante, o carro pode acelerar -1 ,0 ou 1 unidades em cada direção.

Consequentemente, para cada uma das direções, o conjunto de acelerações possíveis com a = (al,ac) a representar a aceleração de um carro nas duas direções num determinado instante, é o seguinte:

- a = (1,1)
- a = (1,0)
- a = (1,-1)
- a = (0,1)
- a = (0,0)
- a = (0,-1)
- a = (-1,1)
- a = (-1,0)
- a = (-1,-1)

Tendo em consideração **p** como tuplo que indica a Posição de um carro numa determinada Jogada, j

$$j\,(p^j = (p_l, p_c))$$

**Figura 1.**

e **v**, o tuplo que indica a Velocidade do carro nessa Jogada,

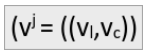


**Figura 2.**

na seguinte Jogada, o carro irá estar na posição:

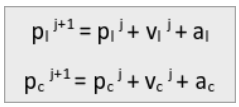


**Figura 3.**

A velocidade do carro num determinado instante é calculada:

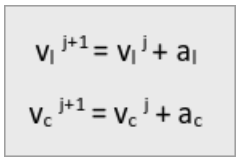


**Figura 4.**

Sendo uma simulação de corrida de carros, existe sempre a possibilidade de o carro se despistar e sair da pista, sendo que, quando tal ocorrer, o carro terá de voltar para a Posição anterior, assumindo um valor de Velocidade zero. Cada movimento de um carro numa determinada Jogada, de uma certa Posição para outra, terá um custo de uma unidade, de modo a que, quando o mesmo sair dos limites da pista, o custo é de vinte e cinco unidades.

## 3 Formulação do Problema

Após uma leitura cuidadosa do enunciado, da sua análise e estudo conjugado com o debate entre todos os elementos do grupo, definimos as seguintes características dos problemas.

- **Estado Inicial:** Ponto de Partida (x,y), posição inicial onde o Jogador '**P**' se encontra definido nas coordenadas cartesianas.
- **Estado Final/Teste Objetivo:** Cruzar a meta, ou seja, o Jogador '**P**' sobrepor a meta '**F**", igualmente definido por coordenadas cartesianas.
- **Operadores:**
  - Esquerda
    O jogador com Velocidade **v** e Aceleração **a** deseja movimentar-se para a esquerda. Se a posição estiver livre, identificada por '**-**', o Jogador movimenta-se, aumentando a sua Velocidade e atualizando a sua Posição. Caso esteja ocupada, identificada por '**X**', o Jogador fica com Velocidade igual zero e mantém-se na mesma posição;
  - Direita
    O jogador com Velocidade **v** e Aceleração **a** deseja movimentar-se para a direita. Se a posição estiver livre, identificada por '**-**', o Jogador movimenta-se, aumentando a sua Velocidade e atualizando a sua Posição. Caso esteja ocupada, identificada por '**X**', o Jogador fica com Velocidade igual zero e mantém-se na mesma posição;
  - Cima
    O jogador com Velocidade **v** e Aceleração **a** deseja movimentar-se para cima. Se a posição estiver livre, identificada por '**-**', o Jogador movimenta-se, aumentando a sua Velocidade e atualizando a sua Posição. Caso esteja ocupada, identificada por '**X**', o Jogador fica com Velocidade igual zero e mantém-se na mesma posição;
  - Baixo
    O jogador com Velocidade **v** e Aceleração **a** deseja movimentar-se para baixo. Se a posição estiver livre, identificada por '**-**', o Jogador movimenta-se, aumentando a sua Velocidade e atualizando a sua Posição, Caso esteja ocupada, identificada por

'**X**', o Jogador fica com Velocidade igual zero e mantém-se na mesma posição;

- **Estado Possíveis:**
  - Parado após bater em obstáculo;
  - Parado após sair dos limites da pista;
  - Movimentar-se para a frente;
  - Movimentar-se para trás;
  - Movimentar-se para a esquerda;
  - Movimentar-se para a direita;
  - Movimentar-se para a diagonal superior esquerda;
  - Movimentar-se para a diagonal superior direita;
  - Movimentar-se para a diagonal inferior esquerda;
  - Movimentar-se para a diagonal inferior direita;
- **Custo da Solução:**
  Como já mencionado anteriormente, cada ação bem sucedida custa uma unidade, caso saia dos limites da pista este tem um custo de vinte e cinco unidades.

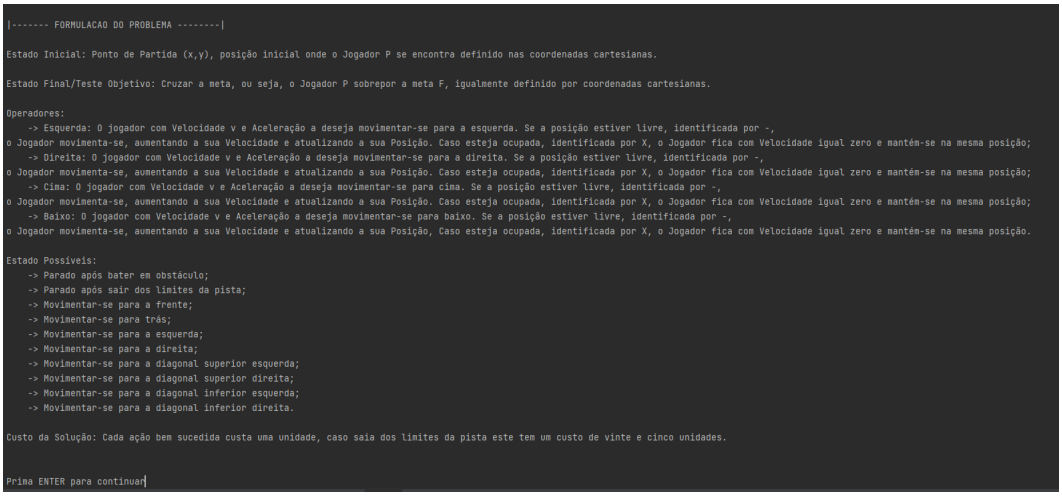



**Figura 5.** Opção 1 do Menu - Formulação do Problema

## 4 Estrutura do Projeto

O Projeto encontra-se partido em dez ficheiros *.py*, bem como, uma pasta Circuitos onde se encontram os ficheiros *.txt*

No *main.py* encontra-se o código para a execução inicial do programa e é aqui que está presente o Menu que será apresentado ao utilizador. Menu este, que possui o tópico já abordado, Formulação do Problema, tal como, Criar *VectorRace*, Pista em Forma de Grafo e Estratégias de Procura, temas que irão de seguida ser abordados.

Nos ficheiros *tk.py, track.py, vectorrace_criar.py* e *vectorrace_ler.py* encontra-se todo o código necessário à representação gráfica dos Circuitos. O primeiro e o segundo ficheiros têm a configuração necessária para o *Tkinter* representar o Circuito da maneira que planeamos. Os dois últimos são semelhantes na sua estrutura, diferindo apenas na parte em que um interpreta um ficheiro *.txt* e o mostra numa interface gráfica e o outro que cria de raíz um Circuito através do input do utilizador e o apresenta também na interface gráfica. De notar que, ao tentar ler um Circuito, está implementado um sistema que impede que isto seja feito caso o mesmo não seja válido, sendo devidamente fornecido *feedback* na forma de *output* pelo sistema. Um circuito pode ser inválido pelas mais diversas razões, nomeadamente apresentar mais do que um Checkpoint.

Já nos ficheiros *Node.py* e *Graph.py* encontram-se as definições de nodos e grafos, respetivamente, e ainda algumas funções indispensáveis e fundamentais para a resolução dos problemas, como, por exemplo, *calcula_custo*, que dado um caminho calcula o custo do mesmo.

O ficheiro *readFile.py* envolve todas as funções necessárias e fulcrais para a leitura, demonstração e determinação de pontos num *.txt*.

O ficheiro *map_creation_tool.py* recorre de novo à *package TKinter* apresentando ao utilizador uma interface gráfica interativa baseada em botões, em que é possível a criação de um Circuito, fornecendo o número de colunas e linhas do mesmo, ou a alteração de um já existente na pasta Circuitos.

Por último, no *Race.py* estão presentes as funções relativas à construção do grafo, com principal destaque para a função *cria_grafo*, que, tal o nome indica, cria o grafo respetivo ao *Circuito.txt* dado,

através da associação de adjacências, não passando mais do que uma vez em cada Posição do *txt*.

mas mudanças

## 5 Tarefas Realizadas

Inicialmente, para começar a implementar o problema formulado, foi necessário a instalação das *packages python* `matplotlib`, `networkx` e `tkinter` através dos seguintes comandos:

```
$ pip install networkx
$ pip install matplotlib
$ pip install tkinter
```

## 5.1 Circuitos *VectorRace*

O grupo decidiu que, primeiramente, após algum debate de possíveis ideias, iriam ser representados graficamente os circuitos e, para isso, partiu-se de dois cenários distintos:

- Gerar um *VectorRace* a partir de um ficheiro *.txt*;
- Permitir a criação de um *VectorRace*, com posterior geração de um *.txt*, dada a informação necessária, como, por exemplo, ponto inicial ou tamanho do Circuito.

No entanto, antes de chegar a estas duas hipóteses, preparamos as representações gráficas.

**Gráficos:**

Após alguma ponderação e investigação *online*, foi óbvia a escolha da biblioteca *Tkinter* para a resolução do problema em causa. Depois de implementar o *Tkinter* de forma a conseguir representar o desejado, foi necessário definir alguns pontos.

Esta biblioteca foi utilizada para representar um circuito, o qual definimos numa classe *Track*. Cada *Track* tem uma dimensão e é constituída por uma outra classe *Point*, que é, de forma simplificada, um tuplo a representar coordenadas. Há, ainda, a possibilidade de implementar barreiras (que definimos numa classe *LineSegment*, que seriam linhas entre dois *Points* e dariam o efeito de barreira pretendido) e de um checkpoint (que seria um ponto de passagem obrigatória).

Com isto tudo implementado e operacional para criar um circuito *VectorRace*, basta chamar a classe *Track* passando como argumento as dimensões, o ponto de chegada juntamente com o ponto de partida, uma lista de barreiras e um checkpoint (os dois opcionais).

**Gerar VectorRace a partir de um ficheiro:**

Com a informação explicada em cima, foi necessário dar uso à classe *Track*. Primeiramente, adaptamos um Circuito no formato *.txt* para uma representação gráfica. Assim, lemos o ficheiro linha a linha e vamos processando e tratando a sua informação. Primeiro, fazemos uma pequena verificação para garantir que o ficheiro cumpre alguns requisitos, entre os quais:

- Confirmar que todo o contorno do circuito corresponde a uma barreira (existe apenas uma excepção no caso de o ponto de partida ou de chegada se encontrarem nestas posições);
- Confirmar que só existe um ponto de partida e um ponto de chegada;
- Confirmar que todas as linhas têm o mesmo tamanho.

De seguida, extraímos as coordenadas de partida e de chegada e, por fim, extraímos as barreiras. Posto isto, é apenas necessário chamar a classe *Track* com toda a informação recolhida.

**Criar *VectorRace*:**

Criar um *VectorRace* com informação recebida por input revelou-se um pouco mais trabalhoso, complicado e desafiante do que o ponto dois, uma vez que, apesar de não termos que processar a informação, tivemos de gerar um ficheiro *.txt* de raiz.

Partindo o problema em partes: a informação é recolhida por inputs. À medida que estes inputs vão sendo chamados, vão também sendo chamadas funções que vão construindo o ficheiro *.txt*, começando pela dimensão, pontos de partida e chegada e acabando nas barreiras. Este ficheiro é, posteriormente, guardado na diretoria Circuitos.

## 5.2 Representação do Circuito em forma de Grafo

Após levantadas algumas questões relativas ao modo de implementação do Grafo do respetivo Circuito, o grupo decidiu, primeiramente, rever a Ficha3 e, mais precisamente, o código do ficheiro *Baldes.py*, uma vez que este servia de motor para a resolução deste tópico.

Inicialmente, partindo totalmente da ideia demonstrada pela equipa docente, idealizamos um ficheiro com o construtor *init*, onde este tinha na sua constituição:

```
def __init__(self):
self.g = Grafo(directed=True)
self.custo = 0
self.posicao = "(0,0)"
self.velocidade = "(0,0)"
self.aceleracao = "(0,0)"
```

Da mesma maneira que no ficheiro *Baldes.py*, teríamos um Grafo orientado uma vez que queríamos saber a direção do nodo, grafo esse que seria importado do ficheiro *Graph.py* também disponibilizado pelos docentes, um inteiro relativo ao custo inicialmente a zero, uma *string* que indicava a Posição atual, uma *string* que indicava a Velocidade atual e, por último, uma *string* que indicava a Aceleração atual. O restante código referia-se à construção do Grafo e funções necessárias para o movimento do Carro de Posição.

Após algum debate e análise, deparamo-nos que para este tópico não iríamos precisar do custo, uma vez que apenas pretendemos construir o gráfico resultante de partir do Ponto Inicial até todos os nodos (ou até a meta neste caso), com velocidades a serem alteradas.

Seguidamente, depois de vários pensamentos e erros obtidos, verificamos que, para cada movimento realizado, a aceleração pode tomar uma das nove opções mencionadas no tópico dois. Desse modo, poderíamos remover a Aceleração do construtor *init* e por cada movimento avançado teríamos que considerar as nove possibilidades, o que nos levou a concluir que na função principal *criar_grafo* teríamos de possuir uma maneira de guardar nós visitados anteriormente de maneira a não termos ciclos ou repetição de nodos desnecessari-

amente. Assim, na função *cria_grafo* apenas avançávamos para a Posição seguinte caso a mesma já não tivesse sido visitada.

Após novos erros e incapacidade de completar o grafo na totalidade e por sugestão dos docentes da UC no Enunciado, reformulamos o código e implementamos tudo o que se caracterizava por coordenadas geográficas através do tipo de dados *tuple* mas após termos encontrado algumas dificuldades a manipular listas de tuplos (possivelmente pela inexperiência em desenvolvimento na Linguagem *python*) foi decidido pelo grupo de trabalho alterar o código para um modelo de *strings* juntamente com *arrays* e listas de *arrays*, nomeadamente para armazenar o conteúdo de um ficheiro *.txt*.

### 5.3 Limitação - Implementação de Velocidade Crescente

Infelizmente, tivemos alguma dificuldade em implementar a velocidade como nos foi proposto no enunciado. Desta forma, apesar de termos muito código pronto para a implementação da mesma e após o termos tentado incessantemente, atualmente a velocidade é tomada como zero em todos os pontos, apenas sendo, no fundo, considerados os 9 valores da aceleração para a definição dos pontos adjacentes. Apesar de termos obtido diversos resultados com os vários testes feitos com a velocidade implementada deparamo-nos principalmente com os problemas de o carro "sobrevoar"barreiras e não respeitar *checkpoints* definidos. Em seguida, exemplos dos problemas anteriores:

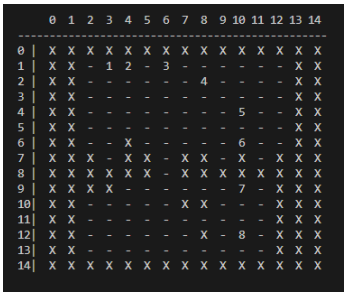



**Figura 6.** Avanço de barreiras indevido entre 6 e 7

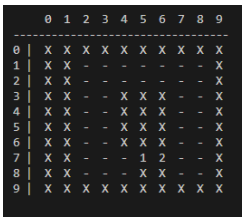



**Figura 7.** Ciclo de P para F imediato, não respeitando *Checkpoints*

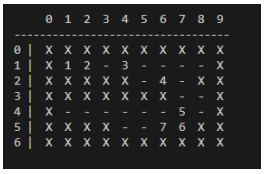



**Figura 8.** Situação Estranha de ignorar movimentação direta de 5 para 7

### 5.4 Estratégia de Procura

Nesta primeira fase do projeto, optamos por implementar uma estratégia não informada. Desse modo, escolhemos o **Algoritmo de Procura em Largura** (BFS).

Num primeiro plano tínhamos como objetivo implementar as duas estratégias não informadas, o **Algortimo de Procura em Profundiade** e o **Algortimo de Procura em Largura**. Porém, durante o desenvolvimento destes métodos, surgiu-nos um problema relativamente à DFS. Resumidamente, na execução dos dois algoritmos, isto é, na execução do DFS anteriormente ao BFS, por algum motivo o sistema conservava alguns dos nodos visitados na primeira estratégia e imprimia esses mesmos nodos na execução da segunda estratégia. Devido ao elevado grau de fragilidade destas funções, não conseguimos descobrir o motivo da errada impressão das ações do **Carro**.

Deste modo, conservou-se, então, como estratégia final implementada, e 100% funcional, o **Algoritmo de Procura em Largura** (BFS). Ficando, assim, o objetivo de implementar todas as estratégias de procura (informada e não informada) na entrega da segunda fase. De notar que o algoritmo BFS tem em conta o caso de existir um Checkpoint no Circuito, tornando obrigatória a pas-


**Figura 9.** Erro na DFS

sagem pelo mesmo e indicando o caminho percorrido e o custo total completos, desde o ponto de início até ao do fim.

# 6 Demonstração dos Resultados Obtidos

Da mesma maneira que no tópico 5.2, inicialmente, reutilizamos o código fornecido pelos docentes nas aulas práticas para servirem como exemplo para o que desejávamos efetuar.

Assim tínhamos como ponto de partida um ***main*** bastante simples onde era apenas possível **Imprimir Grafo**, ou seja, imprimia no terminal todos os nodos e suas ligações a outros nodos do grafo, **Desenhar Grafo**, com recurso à biblioteca *matplotlib* abria-se um novo separador onde era representada a construção do grafo, **Imprimir Nodos do Grafo**, melhor dizendo, no *output* era possível verificar todos os nodos presentes no grafo, **Imprimir Arestas do Grafo**, em outras palavras, imprimia todos as arestas nas ligações entre nodos presentes no grafo e por fim, os **Algoritmos DFS e BFS** falados no tópico 5.3.

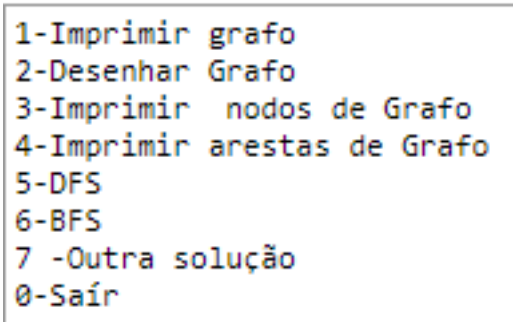



**Figura 10.**

Após uma avaliação e análise do que iríamos ou não manipular, o grupo decidiu elaborar um *main* bastante mais apelativo, bem como, mais elaborado, organizado e requintado. Assim, inicialmente, começou por melhorar a elegância do *output* acrescentando alguns pormenores nas bordas que fazem toda a diferença.

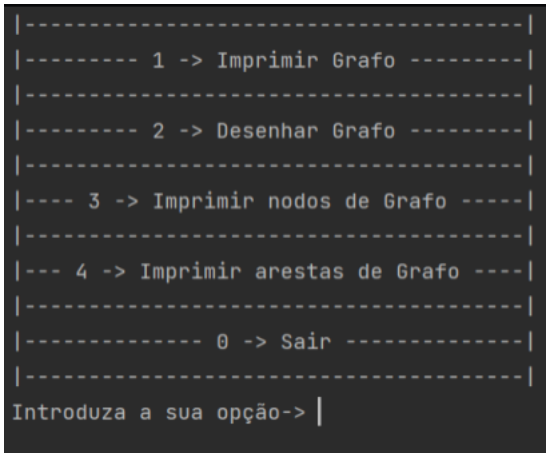



**Figura 11.**

Posteriormente, ficou decidido que seria necessário acrescentar a representação gráfica do Grafo, bem como, a possibilidade de imprimir o *circuito.txt*. Assim, foram criados três novos tópicos: **Imprimir Circuito.txt**, **Imprimir VectorRace** e **Criar VectorRace**. Ficando o *main* com todas estas possibilidades:

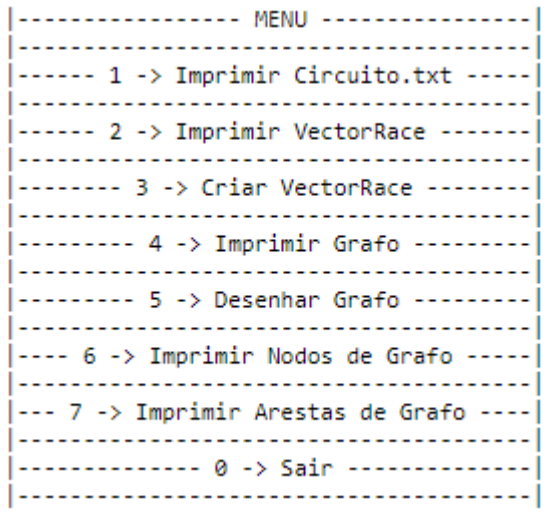



**Figura 12.**

Com o avançar do Projeto, verificamos que ter apenas um Circuito iria ser demasiado básico e incompleto, levando o grupo a reunir e a decidir que iríamos ter também a possibilidade da criação de um novo Circuito, para além de ter a liberdade de escolher um dos nosso dez Circuitos já disponíveis para usar. Este novo Circuito não tem qualquer restrição e o resultado final está apenas dependente do utilizador.

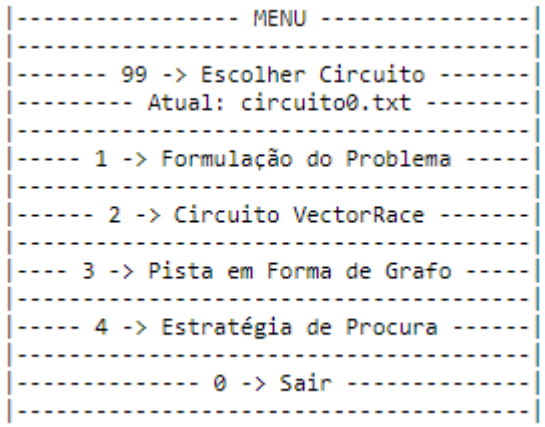



**Figura 13.**

Em suma, após todos estes passos chegamos ao resultado final do nosso *main*. Um *main* constituído por todo o Projeto desde a sua Formulação até às Estratégias de Procura, passando pelo Circuito *VectorRace* e Pista em forma de Grafo. De notar que algumas funcionalidades, apesar de já estarem disponíveis as respetivas opções, ao selecionar apenas é indicado que irá ainda ser implementado. De seguida, através de *prints*, iremos mostrar um exemplo para um *Circuito.txt* e um Manual de Instruções de como utilizar o Menu:

## 6.1 Manual de Instruções do Menu

Através do input 99, é-nos apresentado o Menu em que é possível escolher qualquer um dos Circuitos na pasta Circuitos. Considerando que o número destes presente na pasta vai variando, a lista de opções apresentada é, também, dinâmica. O circuito inicial é tomanho como sendo o Circuito0.txt e o escolhido, em caso de alteração, mantém-se e serve de base para todas as funcionalidade do menu. Em seguida podemos ver um *print* exemplificativo do mesmo.

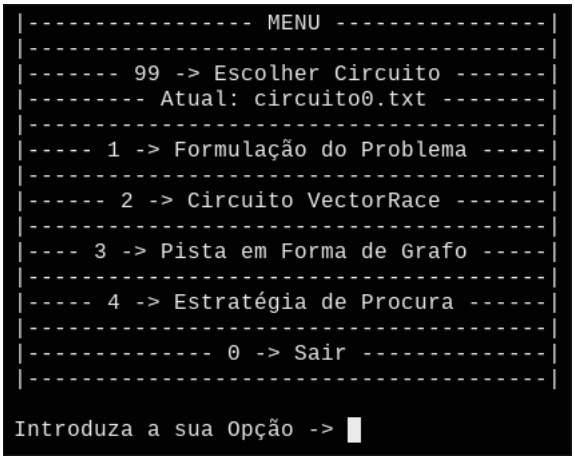



**Figura 14.**

O menu correspondente à escolha da opção Nº2, resposta ao input "2", apresenta as opções Imprimir CircuitoX.txt, *VectorRace* do CircuitoX.txt e Criar *VectorRace*. Em seguida, apresentamos o output destas.

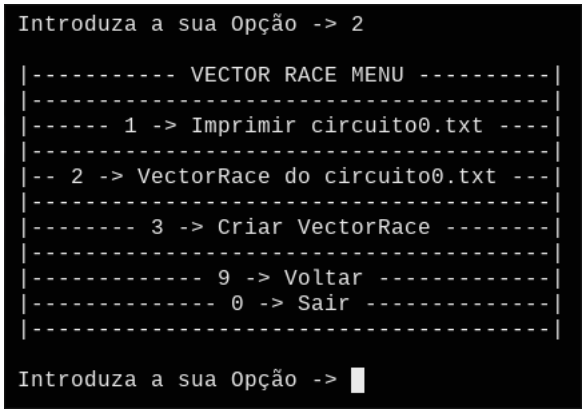



**Figura 15.**

Através do ponto 3 ("3 - Pista em forma de Grafo") do Menu é possível correr as funções explicadas anteriormente, **Imprimir nodos do Grafo** e **Imprimir arestas do Grafo**, mas também podemos pedir uma representação gráfica em forma de Grafo do Circuito, estando representados as adjacências e o custo das mesmas. É, também, possível representar estas adjacências de forma textual através da seleção da opcão 1 - Imprimir Grafo, sendo apresentados os diversos Nodos e as respetivas associações e custo. Em seguida, podemos ver o resultado da execução destas opções.

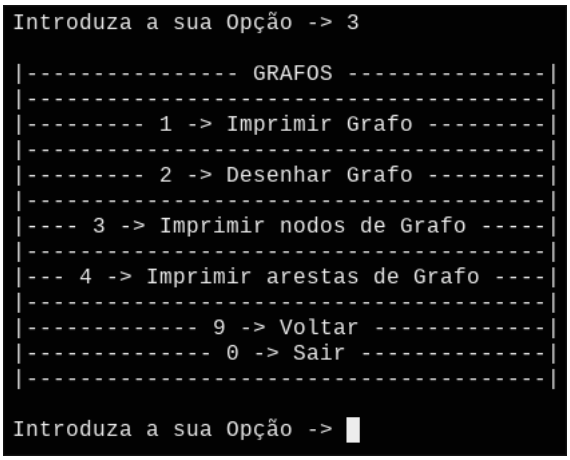



**Figura 16.**

Em seguida, e por último, através do input 4, podemos ver o Menu dos Algoritmos onde é possível escolher de entre os quatro que terão que estar devidamente implementados na segunda fase. Neste momento, como já foi explicado anteriormente, apenas a opção 2 está funcional, obtendo o seguinte output para o *Circuito0.txt*.

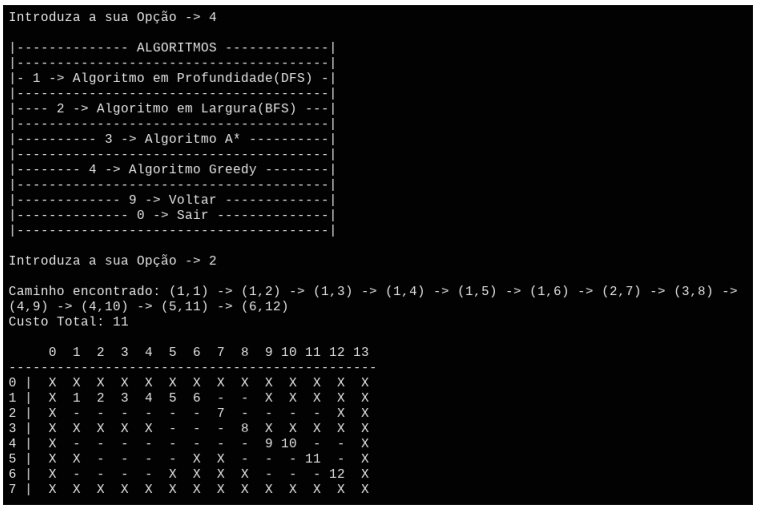



**Figura 17.**

# Conclusões

### *Estratégias*

Após uma longa análise dos algoritmos apresentados, nesta primeira fase, concluímos que a melhor estratégia para a formulação deste problema seria o Algoritmo de Procura em Largura (BFS). Conseguimos a implementação deste algoritmo na sua totalidade e com a máxima funcionalidade.

Apesar de, analisando os testes, os dois algoritmos de procura não informada não serem muito diferentes a BFS acaba por ter um custo reduzido, perto do custo real. Admitimos que esta conclusão poderá ser diferente se o grafo aumentar, dependendo se este aumenta em profundidade ou em largura.

### *Comentário Final*

Em conclusão, gostaríamos de realçar a importância que tiveram os métodos de formulação e resolução de problemas no nosso crescimento, não só dentro desta Unidade Curricular, mas também no espectro mais geral. O grupo de trabalho encontra-se bastante satisfeito por todo o trabalho produzido e todo o esforço dedicado de modo a obter o melhor resultado possível, apesar da estratégia de procura incompleta, bem como, a impossibilidade de mudança de velocidade na criação do Grafo. De certo que para a próxima fase esses resultado menos alcançados serão corrigidos e resolvidos. Estamos desta maneira prontos para enfrentar a próxima fase que aí se avizinha.

# Referências Bibliográficas

1. Draw.io: Ferramenta para construção gráfica da Topologia
2. StackOverflow: Ferramenta para resolver problemas relacionados com código e erros associados
3. Tkinter: Documentação consultada como auxílio na construção da parte gráfica