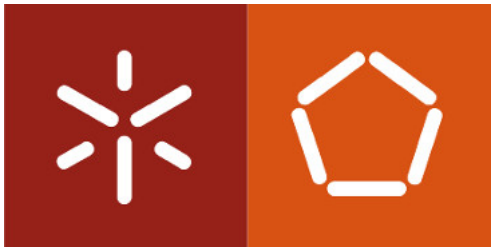

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Inteligência Artificial

Ano Letivo de 2022/2023

Instrumento de Avaliação em Grupo

Grupo 2.06

A96326 Bernard Ambrosio Georges
A97257 João Miguel Ferreira Loureiro
A94870 Rafael Picão Ferreira Correia
A91682 Robert Beniamin Szabo

30 de novembro de 2022

Índice

1 Introdução

Este relatório é referente á primeira fase do Instrumento de Avaliação em Grupo da Unidade Curricular de Inteligência Artificial. Neste projeto, foi nos pedido para apresentar alguns dos vários algoritmos de procura desenvolvidos na linguagem de programação **Python**, aplicando-os ao jogo **VectorRace**.

O VectorRace, também conhecido como RaceTrack, é um jogo de corridas de carro onde o objetivo é ser o primeiro a chegar á meta. A pista, tradicionalmente desenhada em superfícies quadriculares, apresenta um ponto de partida e outro de chegada.

Ao longo de uma partida, os jogadores devem-se movimentar em direção á meta, prestando sempre atenção á sua velocidade e aceleração de forma a evitar sair dos limites da pista.

Esta primeira fase irá abordar o desenvolvimento de dois algoritmos: **A* e BFS**, também como a base do projeto inteiro.

2 Descrição do Problema

Nesta versão do VectorRace, as ações do movimento do carro são equivalentes a um conjunto de acelerações. O carro pode se movimentar conforme uma linha ou uma coluna, podendo ter, para cada uma das direções, uma aceleração de {-1,0,+1}. Juntamente com a velocidades, decidem se o carro está perante um caso de aceleração, abrandamento ou de uma mudança de direção.

Assim, a posição do carro é representada por um tuplo *(**L**inha(x),**C**oluna(y))*, cuja posição é calculada pelas seguintes formulas:

$$PosiçãoSeguinte = (PosicaoAtualL + VelocidadeAtualL + AcelarçãoL, PosicaoAtualC + VelocidadeAtualC + AcelarçãoC)$$

Figura 1: Fórmula que calcula a posição do carro

$$VelocidadeSeguinte = (VelocidadeAtual + Acelarção)$$

Figura 2: Fórmula que calcula a velocidade do carro

Se o carro sair da pista, este volta á sua posição anterior, com a sua velocidade a assumir um valor de zero. Assim, cada movimento que o carro executar terá **um custo de 1 unidade**, mas se sair dos limites da pista terá **um custo penalizador de 25 unidade**.

Os circuitos fornecidos devem ser válidos, ou seja, têm que conter uma posição inicial, espaços fora da pista/obstáculos e posições finais, constituindo a meta.

3 Formulação do Problema

3.1 Estados:

Neste problema consideramos como estado **a união das coordenadas com a velocidade**. Esta união é justificada com o facto de o carro poder passar na mesma posição com velocidades diferentes, formando consequentemente filhos em posições distintas. É representado no código por um tuplo constituído por tuplos (coordenadas: (x,y), velocidade: (vetor x, vetor y))

3.2 Estado Inicial:

O estado inicial deste enunciado é **a posição de partida do circuito**, e neste caso, assumimos sempre que o carro começa de um posição estática, ou seja, **com uma velocidade nula**, significando que é igual a 0 em ambas as direções.

3.3 Estado Objetivo:

O objetivo deste enunciado é **chegar á meta**. Isto acontece quando **o caminho percorrido pelo o carro atinge uma das posições finais (meta) do circuito.** Este estado é peculiar, no sentido em que a velocidade com que o carro o atinge é irrelevante.

3.4 Operadores:

Os operadores deste enunciado são as **acelerações possíveis entre os vários estados do percurso**. Têm o efeito de modificar a velocidade, e consequentemente o trajeto possível num dado momento, uma vez que aceleração altera diretamente a velocidade do instante seguinte, que por sua vez ira alterar a posição do estado subsequente.

3.5 Custo da Solução:

O custo da solução será calculada **consoante o número de "instantes"tomados pelo o algoritmo até chegar ao Estado Final a partir do Estado Inicial.** Como mencionado anteriormente, cada deslocamento tem um custo de 1, enquanto que fugas das pista/colisões têm uma penalização de custo 25.

4 Implementação e Funcionalidades

4.1 Circuito

Os circuitos são gerados através de um ficheiro de texto. Como explicado no enunciado do projeto, este ficheiro deve representar a pista a percorrer, seguindo a seguinte sintaxe:

- a letra **'P'** representa a posição de partida do carro;
- os espaços com **'-'** representam a pista, ou seja, o que o carro pode percorrer;
- as letras **'F'** representam a meta da pista;
- as letras **'X'** representam os obstáculos/o fim da pista, isto é, os espaços penalizadores.

Nesta primeira fase decidimos incluir quatro circuitos de teste: o **Circuito 1**, igual ao incluído no enunciado como exemplo, o **Circuito 2**, um circuito criado á mão com o objetivo de testar algumas situações relevantes, o **Circuito 3**, uma variação do Circuito 2 para testar o seu comportamento se o atalho no circuito anterior não existisse e, por último, o **Circuito 4**, criado com o objetivo de testar o movimento diagonal.

O utilizador pode também, se desejar, optar por usar um circuito diferente dos pré-definidos. Basta escolher a opção respetiva no menu inicial e inserir o caminho correspondente á localização do respetivo ficheiro de texto.

```
X X X X X X X X X X
X X - - - X X - - X
X - - - - - - - - F
X P - - X X X - - F
X - - - - - - - - F
X X X X - - - - X X
X X X X X X X X X X
```

Figura 3: Ficheiro de texto *circuito1* que representa o Circuito 1

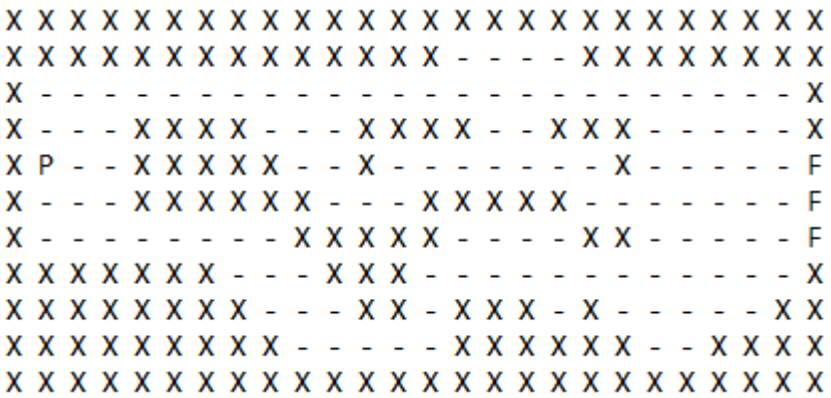

Figura 4: Ficheiro de texto *circuito2* que representa o Circuito 2

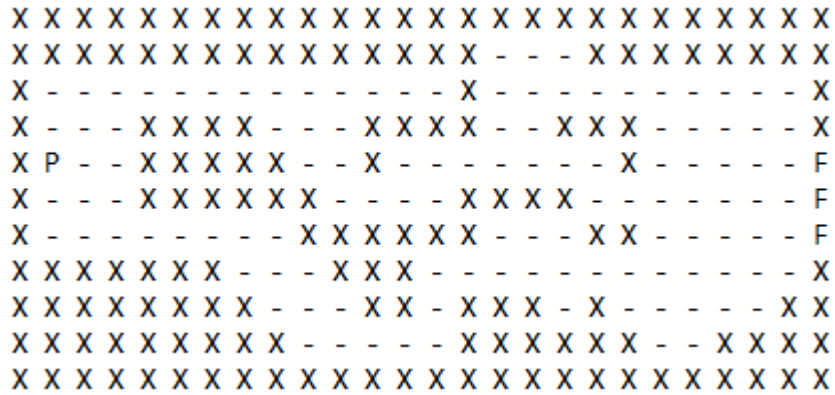

Figura 5: Ficheiro de texto *circuito3* que representa o Circuito 3

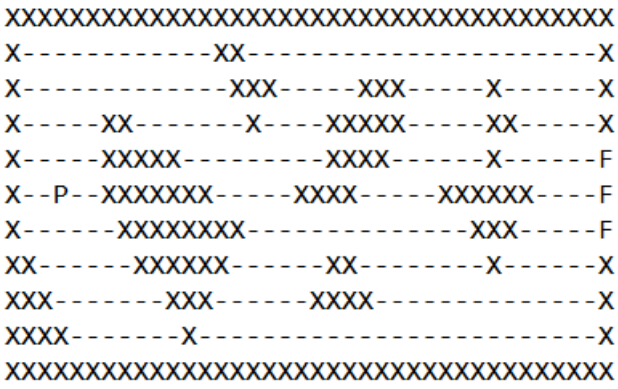

Figura 6: Ficheiro de texto *circuito4* que representa o Circuito 4

Ao iniciar o programa, o utilizador escolhe o circuito que pretende utilizar (através de um menu). Após a escolha feita, decorre o parsing do ficheiro correspondente e a sucessiva criação

de uma matriz correspondente ao circuito e do grafo, sendo o utilizador notificado da posição de partida e das posições da meta.

Nota: Neste projeto decidimos usar a notação de coordenadas (x,y), conforme o exemplo seguinte:

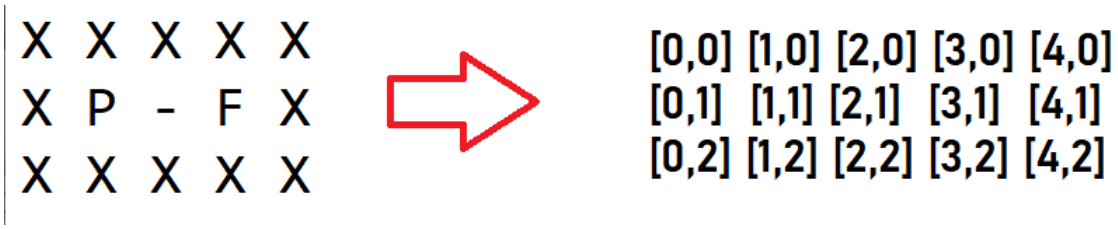

Figura 7: Exemplo da representação das coordenadas de um circuito

4.2 Representação do Circuito VectorRace

Para representar o circuito de uma forma mais apresentável e fácil de ler, decidimos utilizar a biblioteca *pygames* para criar uma representação gráfica do circuito e do percurso encontrado.

Nota: Para utilizar esta e outras funcionalidades do projeto, é necessário instalar as bibliotecas *pygame,networkx e matplotlib* de Python.

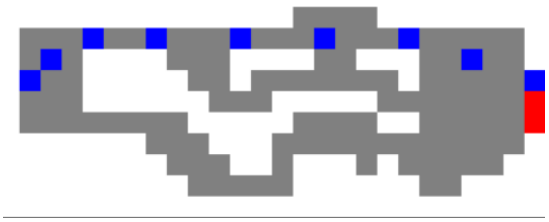

Figura 8: Representação gráfica da solução encontrada pelo o algoritmo no Circuito 2

Para o visualizar, o utilizador, após escolher qual algoritmo pretende utilizar, precisa de escolher a opção "Desenhar circuito em VectorRace"do menu e será, consequentemente, aberta uma janela onde é apresentado uma imagem semelhante á anterior. Quando o utilizador escolher parar de ver a imagem, precisa simplesmente de pressionar Enter no programa principal.

Nota: A janela pygame poderá aparecer como a não responder. Isto, no entanto, é normal, visto que fica a espera de um input que não recebe.

Na representação gráfica do circuito, os quadrados cinzentos representam a pista, os brancos os obstáculos/fora da pista, os vermelhos a meta e os azuis, o percurso do carro.

O utilizador pode também escolher a opção "Imprimir Grafo do circuito"para obter uma representação gráfica do grafo criado através do circuito, gerada através da biblioteca *nx*.

Nota: A imagem gerada poderá ser difícil de ler, dependendo do tamanho do circuito, e consequentemente, do grafo.

4.3 Algoritmo AEstrela

Para a primeira fase, resolvemos implementar o algoritmo de procura informada **AEstrela (A*)**. Escolhemos este algoritmo porque, além de ser um dos mais fiáveis, é mais indicado para encontrar o caminho mais rápido entre dois pontos.

De uma forma muito resumida, o A* estima o custo a partir do nodo de um grafo onde se encontra até o nodo de chegada, utilizando uma heurística e evitando sempre expandir os caminhos mais dispendiosos. Apresenta, no entanto uma complexidade enorme, já que mantém todos os nodos na sua memória.

Como é um algoritmo de procura informada, o A* necessita de uma heurística. Neste problema, como sabemos as coordenadas do Estado Objetivo, decidimos **usar como heurística a distância entre o nodo atual e o nodo que correspondente ás coordenadas do objetivo**.

O algoritmo calcula as várias distâncias dos caminhos possíveis, atribuindo a cada um um peso, por sua vez equivalente á soma dos custos dos espaços percorridos.
Se encontrar um obstáculo/fora da pista num dado caminho, o custo desse caminho é multiplicado por 25 (penalização). O algoritmo escolhe então o caminho menos penalizador, evitando assim as colisões.

Utilizamos esta heurística de forma a otimizar a procura do caminho ótimo. Ao comparar a versão final com uma versão anterior do algoritmo, onde a heurística era só o custo de cada espaço, pode-se observar que o caminho escolhido passou possuir um custo menor:

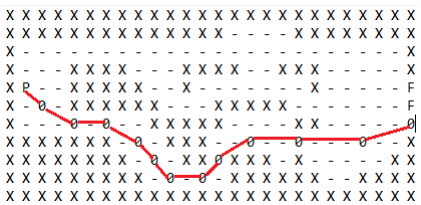

Figura 9: Percurso escolhido com a heurística anterior

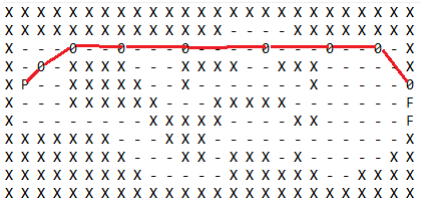

Figura 10: Percurso escolhido com a heurística atual

Um fator que tivemos que considerar na criação do grafo foi o caminho que o carro percorre entre os seus estados. Por exemplo, se entre dois estados (posições) válidos existir um caminho obstruído, essa sequência de estados deixa de ser um caminho válido.

Assim, de forma a resolver esse problema, o algoritmo prevê e esses casos através de uma segunda procura informada, onde o carro apenas move um passo de cada vez.

Para descobrir o custo da solução encontrada pelo o algoritmo, o utilizador deve selecionar no menu a opção "Resolver o problema utilizando um algoritmo", onde lhe é apresentado uma lista de pares de tuplos, onde cada par representa os estados que o carro toma, e o custo total da solução.

4.4 Algoritmo BFS

Visto que o algoritmo A* é de uma procura informada, decidimos implementar o outro algoritmo, por sua vez de pesquisa não-informada.

Assim decidimos implementar o **algoritmo BFS (Primeiro em Largura)**. Como o nome indica, este algoritmo expande a sua procura ao explorar todos os nodos adjacentes de um grafo, podendo assim não encontrar o caminho ótimo. Não possui uma heurística devido ao seu caráter não-informado e a sua complexidade depende diretamente do tamanho do grafo.

Na nossa implementação, usa o mesmo grafo que o algoritmo A*, percorrendo o grafo ao nível da profundidade. Começa por explorar os nodos filhos do nodo de partida, explorando depois os filhos destes, e assim sucessivamente até encontrar um caminho até um dos nodos finais.

4.5 Análise Preliminar dos Resultados

Apresentamos agora os resultados das soluções encontradas para o circuito 4 pelos algoritmos diferentes:

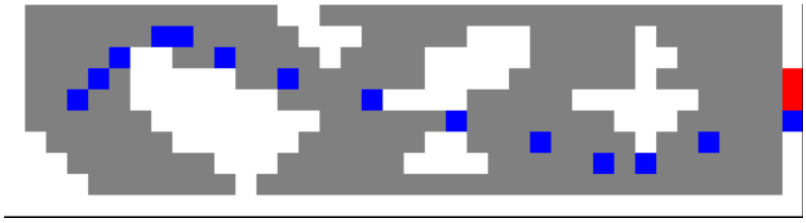

Figura 11: Resultado da aplicação do algoritmo A* no circuito 4

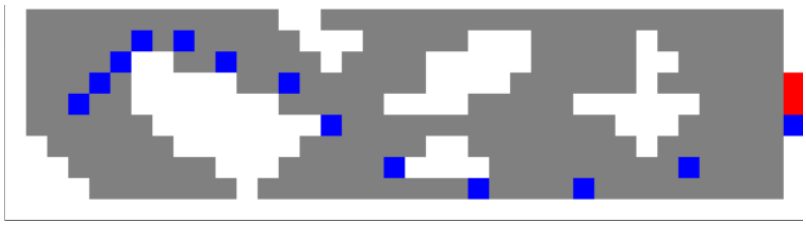

Figura 12: Resultado da aplicação do algoritmo BFS no circuito 4

Como se pode observar, têm um início semelhante, diferenciando-se pelo o facto do A* escolher o caminho do meio e o BFS o de baixo, explicado pela a diferença entre as filosofias de expansão dos dois algoritmos.

O custo da solução do algoritmo A* para este circuito foi 13, enquanto que o do BFS foi 12. Enquanto que o custo do BFS foi menor do que o do A* neste circuito, a diferença é mínima, pelo que nesta primeira fase do projeto não é possível concluir qual dos dois é o melhor.

5 Conclusão

Ao desenvolver esta primeira fase, conseguimos aplicar os conhecimentos que obtemos nas aulas desta UC.

Através da implementação do algoritmo A*, conseguimos analisar e entender melhor o seu comportamento, do qual concluímos que a nossa implementação, apesar de não ser necessariamente a menos dispendiosa, consegue encontrar o melhor trajeto a percorrer até á meta.

Já através da implementação do algoritmo BFS conseguimos observar as diferenças de filosofia de expansão do grafo entre as procuras informadas e não-informadas. Observamos também os custos semelhantes obtidos pelas suas soluções, pelo que os resultados preliminares foram inconclusivos.

Assim sendo, encontramos-nos prontos para começar a desenvolver a segunda fase, onde serão implementados mais algoritmos de procura, com o objetivo de encontrar o algoritmo que encontra a melhor solução.