Universidade do Minho
Escola de Engenharia

Algoritmos de Procura
VectorRace

Trabalho Prático - Fase 1
Inteligência Artificial

Grupo 12

Emanuel Lopes Monteiro da Silva - a95114

Gabriela Santos Ferreira da Cunha - a97393

Miguel de Sousa Braga - a97698

Nuno Guilherme Cruz Varela - a96455


a95114


a97393


a97698


a96455

Conteúdo

1 Descrição do Problema **3**

2 Formulação do Problema **4**
2.1 Estado inicial 4
2.2 Estado objetivo 4
2.3 Operadores 5
2.4 Custo da solução 5

3 Descrição das tarefas realizadas **5**
3.1 *Parsing* dos circuitos 5
3.2 Criação do grafo 6
3.3 Cálculo do próximo estado dada a aceleração 6
3.4 Implementação de algoritmos de procura 7
3.4.1 Algoritmo *Breadth-First Search* 8
3.4.2 Algoritmo *Depth-First Search* 8

4 Manual de Utilização **8**

5 Sumário e discussão dos resultados obtidos **9**

6 Conclusão **11**

1 Descrição do Problema

Para este trabalho prático, foi-nos proposta a implementação de vários algoritmos de procura no contexto do jogo *VectorRace*.

O *VectorRace* consiste num jogo de simulação de carros simplificado em que o carro percorre um circuito predefinido, transitando entre posições definidas por um referencial a duas dimensões.

O carro pode mover-se em qualquer direção, dependendo da aceleração escolhida. 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. Assim, para cada uma das direções o conjunto de acelerações possíveis é {-1, 0, 1} e a é o tuplo que representa a aceleração de um carro nas duas direções nesse instante.

Tendo em conta que p é um tuplo que indica a posição do carro e v um tuplo que indica a velocidade numa determinada jogada j, o movimento do carro rege-se pelas seguintes equações:

$$p_l^{j+1} = p_l^j + v_l^j + a_l$$
$$p_c^{j+1} = p_c^j + v_c^j + a_c$$

$$v_l^{j+1} = v_l^j + a_l$$
$$v_c^{j+1} = v_c^j + a_c$$

Neste problema, tanto a aceleração como a velocidade são grandezas discretas, uma vez que podem apenas tomar valores inteiros.

De forma a simular com o maior realismo possível uma corrida real, é possível que um carro saia da pista, caso em que é introduzido um custo de 25 unidades. Não havendo qualquer situação anormal, o custo de cada movimento, independentemente da deslocação atingida, tem um custo de 1 unidade.

Os circuitos são representados em formato texto, com as posições fora da pista a serem representadas por 'X', os espaços vazios por '-', a posição de partida por 'P' e as diversas posições de chegada por 'F'. Na figura 1 é apresentado o exemplo de um circuito criado pelo grupo.

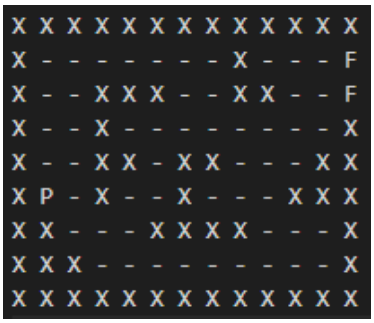

Figura 1: Circuito criado pelo grupo.

Resumindo, o objetivo deste trabalho passa por implementar e comparar diferentes métodos de procura para a resolução do problema apresentado, ou seja, tentar encontrar, no contexto de uma corrida, o melhor caminho para um carro se deslocar desde a casa de partida até à meta.

2 Formulação do Problema

Este é um problema de pesquisa na medida em que queremos encontrar a sequência de ações que nos permite chegar de um estado inicial a um estado objetivo com o menor custo possível. Este problema pode ser formulado, portanto, como um problema de pesquisa num grafo, em que os nodos representam os estados do carro nas diversas fases do seu percurso até ao estado final.

2.1 Estado inicial

Um estado (ou nodo) é caraterizado pela posição (coordenadas x e y) do carro e pela velocidade do mesmo (componentes também em x e em y). Podemos considerar, por isso, que o estado inicial é um estado em que o carro tem uma posição qualquer (representada no mapa) e velocidade 0 nas duas componentes.

2.2 Estado objetivo

Um estado (ou nodo) objetivo é uma dada posição do mapa, sendo caraterizado pelas suas coordenadas x e y. A velocidade neste nodo não é relevante, uma vez que pretendemos apenas que ele chegue a essa posição, independetemente da velocidade com que o faça.

2.3 Operadores

Os operadores de mudança de estado são as possíveis deslocações que o carro pode sofrer, estando dependentes da aceleração que o carro obtém naquele instante. As equações que regem a mudança de estado são as apresentadas na secção 1. Há 9 valores diferentes para a aceleração do carro (3 para a aceleração em x e 3 para a aceleração em y), pelo que, a partir de um estado origem, há, no máximo, 9 estados destino para os quais pode expandir. No entanto, há a possibilidade de escolhas diferentes para a aceleração resultarem num mesmo estado resultante, com posição e velocidade iguais.

Figura 2: Carro na posição (1.5, 1.5)

Por exemplo, na figura acima o carro começa na posição (1.5, 1.5) com velocidade (0,0). O carro pode assim tomar um de 9 deslocamentos (norte, sul, este, oeste, nordeste, noroeste, sudeste, sudoeste e ficar na mesma posição). No entanto, para todos os deslocamentos exceto (+1,0) e (0,0), o carro bate numa parede e volta à posição inicial (1.5,1.5).

2.4 Custo da solução

O custo da solução é dado pela soma dos custos de cada deslocação. Normalmente, o custo de cada deslocação é 1, contudo, esse valor pode ser 25, caso o carro saia da pista na sua deslocação.

3 Descrição das tarefas realizadas

3.1 *Parsing* dos circuitos

De modo a fazer a leitura de um circuito, optamos por representar os circuitos em forma matricial, utilizando uma lista de listas para guardar todas as peças do mesmo.

Estas peças são representadas, na matriz, da seguinte forma:

- 'P' (posição inicial) se no mapa se encontra um 'P' nessa posição;
- 'F' (posição final) se no mapa se encontra um 'F' nessa posição;

- 'X' (obstáculo/fora da pista) se no mapa se encontra um 'X' nessa posição;
- '-' (posição livre) se no mapa se encontra um '-' nessa posição.

Implementamos, então, um *parser* que vai ler um circuito de um ficheiro e devolver a matriz do circuito, juntamente com a posição inicial ('P') e as posições finais ('F').

3.2 Criação do grafo

De modo a encontrar um caminho até à posição final, isto é, encontrar uma solução, é necessário gerar o grafo de todos os estados possíveis. Para tal, começamos por definir uma função, explicada mais à frente, que calcula o próximo estado a partir de outro e de um par de aceleração. Com isto, a função de expandir um estado através das 9 possíveis acelerações já pode ser definida e, portanto, o grafo pode ser construído.

O grafo é construído com a ajuda de duas estruturas, uma para guardar os estados e outra para guardar os estados já visitados. Enquanto houver estados na primeira estrutura referida, vamos retirar um estado para ser expandido. Para cada nodo expandido a partir deste estado, criamos uma nova aresta que vai ser adicionada ao grafo. No final, verificamos se o nodo está presente na estrutura dos visitados, de forma a adicionar ou não nessa estrutura. Este último passo é feito para evitar ciclos infinitos na criação do grafo.

3.3 Cálculo do próximo estado dada a aceleração

De modo a gerar o grafo que representa os vários estados possíveis (posição e velocidade) e as transições entre esses estados, começamos por definir uma função auxiliar que, dada uma posição e um deslocamento, calcula a posição final do carro. Esta posição assume que o carro pode deslocar-se na diagonal. Assumimos também que o carro parte do centro de um quadrado e termina no centro de outro quadrado ou do quadrado de onde partiu.

O algoritmo calcula de forma iterativa todos os pontos de fronteira dos quadrados por onde o carro passa.

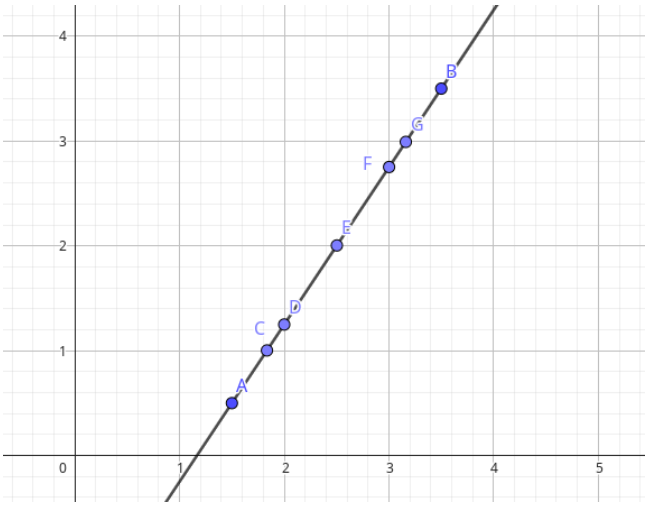

Figura 3: Pontos de fronteira de cada posição

Por exemplo, no deslocamento apresentado na figura, onde o carro parte da posição A(1.5, 0.5) para a posição B(3.5, 3.5), são verificados os pontos C, D, E, F e G.

Para cada ponto, o algoritmo verifica qual é o objeto no próximo quadrado (parede, espaço vazio ou posição final). Ele sabe qual o quadrado a considerar através da posição de fronteira e do deslocamento. Deste modo, caso o objeto encontre uma parede na casa seguinte, mantém-se na casa anterior. Caso encontre uma posição final, pára nessa posição final. Caso a casa seja vazia prossegue o algoritmo. Isto é feito até alguma destas condições de paragem se verificar ou até o deslocamento terminar.

Considerando, por exemplo, o ponto E da reta, o algoritmo deverá verificar qual o objeto existente no quadrado cujo centro é o ponto (2.5, 2.5), testando as condições acima apresentadas.

3.4 Implementação de algoritmos de procura

Como explicitado anteriormente, um dos objetivos do jogo passa por encontrar uma solução para chegar ao final, ou seja, determinar um caminho até às posições finais do circuito. De modo a este caminho ser calculado, temos de implementar algoritmos de procura no grafo.

Nesta primeira fase, decidimos implementar apenas os algoritmos de pesquisa não-informada BFS (*Breadth-First Search*) e DFS (*Depth-First Search*), de modo a construir soluções até às posições finais.

3.4.1 Algoritmo *Breadth-First Search*

O algoritmo *Breadth-First Search* é um algoritmo de procura num grafo em largura. Esta estratégia vai começar por procurar em todos os nodos de menor profundidade. Portanto, vai percorrer todos os nodos de um certo nível de profundidade e quando estes forem percorridos passa para o próximo nível. Normalmente esta procura demora muito tempo e ocupa muito espaço, pelo que apenas deve ser utilizada em problemas pequenos. Contudo, apresenta uma procura muito sistemática.

3.4.2 Algoritmo *Depth-First Search*

Como já foi referido acima, este foi um dos algoritmos usados na execução deste trabalho e o seu objetivo é sempre expandir por um dos nodos mais profundos da árvore. No entanto, como todos os algoritmos, este tem vantagens e desvantagens. Uma das vantagens é a utilização de pouca memória, o que faz com que seja um algoritmo ideal para problemas com muitas soluções. Por outro lado, este algoritmo não pode ser usado em grafos com profundidade infinita, fazendo com que por vezes entre em caminhos errados.

4 Manual de Utilização

Assim que é executado o programa, o utilizador tem acesso a um menu inicial onde poderá carregar o mapa do jogo. O mapa deve seguir as regras apresentadas na secção da descrição do problema. Assim, o mapa deve ser submetido através de um ficheiro de texto contido numa diretoria que deverá ser indicada assim que o sistema pedir, como podemos observar no exemplo da figura 4

```
========================
1... Construir grafo
0... Sair
========================
Introduza a sua opção: 1
Indique a diretoria do ficheiro do circuito: ../circuits/circuito.txt
```

Figura 4: Menu inicial.

Em seguida, caso o mapa seja carregado com sucesso, o utilizador é notificado e é gerado o grafo. O utilizador terá acesso a um novo menu onde poderá imprimir o grafo, os respetivos nodos e arestas, desenhar o grafo ou aplicar os algoritmos de procura referidos na secção anterior. O utilizador poderá também voltar para o menu inicial caso queira alterar o mapa de *input*. É importante referir que existe uma opção "Sair" em ambos os menus, de modo a que o utilizador possa encerrar o programa de forma amigável.

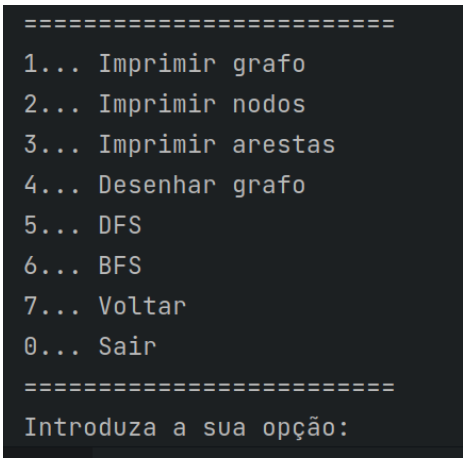

Figura 5: Menu do grafo.

5 Sumário e discussão dos resultados obtidos

Nesta primeira fase, o nosso grupo focou-se na criação do grafo que representa uma dada pista, preparando o terreno para, na próxima fase, testar e comparar os diferentes algoritmos de procura. Para cada estado, o fator de ramificação será 9, pois teremos no máximo 9 estados destino, dependendo das escolhas da aceleração. Assim, não foi surpreendente o facto de o grafo gerado ter um número de arestas bastante elevado, pois este cresce exponencialmente no tamanho de posições válidas do circuito.

Relativamente aos algoritmos implementados, os resultados obtidos para o circuito apresentado foram os seguintes:

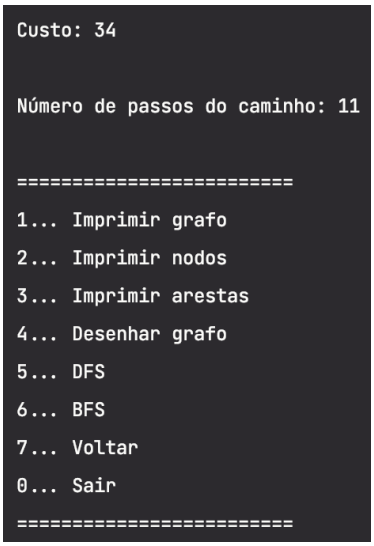

Figura 6: Resultado da procura BFS.

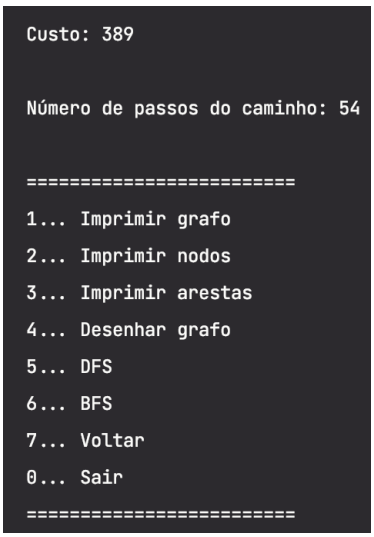

Figura 7: Resultado da procura DFS.

Conforme esperado, verificamos que o algoritmo DFS encontra uma solução com um custo bem maior do que o custo da solução encontrada com o algoritmo BFS. A procura realizada pelo algoritmo DFS é realizada em profundidade, o que o leva a escolher, por vezes, caminhos pouco eficientes. Quanto ao algoritmo BFS, procurará sempre minimizar a profundidade da solução obtida, pelo que é previsível que o número de passos seja menor.

6 Conclusão

Em jeito de conclusão, consideramos que este trabalho permitiu ao grupo consolidar o conhecimento adquirido ao longo das aulas, nomeadamente na geração de grafos e na implementação de algoritmos de procura. Consideramos, por isso, que foram atingidos os objetivos propostos para esta fase. Deste modo, estamos aptos para avançar com a implementação de novos algoritmos de procura informada e não informada em circuitos mais complexos.