Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Inteligência Artificial

Ano Letivo de 2022/2023

Trabalho Prático Grupo 59

Cláudio Bessa (A97063) Carlos Ferreira (A89509)
Gustavo Pereira (A96867) Vítor Lelis (A90707)

5 de janeiro de 2023

IA

Resumo

Este relatório procura descrever o processo de realização da primeira e segunda fase do trabalho prático no âmbito da disciplina de Inteligência Artificial ao longo do primeiro semestre, do terceiro ano, da Licenciatura em Engenharia Informática na Universidade do Minho.

O objetivo do trabalho prático foi sensibilizar e motivar os alunos a conceber resoluções de problemas com algoritmos de procura através de um contexto de jogos de corrida, tendo como referência o *Vector-Race*.

O documento irá descrever de forma sucinta e explicativa o contexto do problema proposto, as soluções adotadas pelo grupo para resolver o mesmo, assim como a motivação para decisões, alterações e adições entre as fases.

Índice

Lista de Figuras

1 Introdução

1.1 Contexto

Nos presentes dias, muitas plataformas de *software* visam otimizar as suas decisões e os caminhos para alcançar os seus objetivos recorrendo à inteligência artificial. Isto implica não só conhecer as possibilidades existentes mas também saber reconhecer qual ou quais são os mais favoráveis no contexto do problema. Para este reconhecimento são utilizados algoritmos de procura.

Com intuito de melhor compreender e aplicar o conceito por trás dos algoritmos de procura, é-nos proposto, pelo dado trabalho prático, a aplicá-los no contexto de jogo de corrida chamado *RaceTrack* ou *VectorRace*.

O *VectorRace*, trata-se de um jogo de simulação de corrida com regras simples onde o vencedor é aquele que consegue sair do ponto inicial e chegar ao ponto final com o caminho calculado mais otimizado. Sendo uma corrida, é necessário haver mais do que um carro a correr, algo que é implementado nesta segunda e última fase do projeto.

1.2 Descrição do Problema

Para construir o *VectorRace* no âmbito do projeto proposto, foi feito o seguinte levantamento: o mapa no qual os carros se encontram é tratado como um grafo, onde cada posição da matriz corresponde a um nodo, e o percurso feito pelos carros é um conjunto de nodos. Também se deve ter em conta que, tal como na física, à medida que

um carro acelera, vai acumulando velocidade.

Dito isso, para cumprir o objetivo do jogo de chegar em um determinado ponto pelo caminho mais otimizado, devemos adotar estratégias de travessia para calcular o resultado. Anexada a estas estratégias de travessias existe a condição de estas possuírem duas variáveis, sendo elas velocidade e aceleração.

Em adição ao cálculo de travessias, outro critério a cumprir trata-se da conversão ficheiros textuais representantes de mapas em grafos. Este ficheiro conterá informações de como mapa será representado e compete ao sistema converter em uma estrutura de grafos populando-o com os devidos nodos e arestas.

No que toca à segunda fase é preciso implementar a possibilidade da simulação contemplar dois ou mais jogadores calculando, em simultâneo, o melhor caminho dado os seus respetivos pontos iniciais e evitando colisões entre si.

2 Implementação

2.1 Requerimentos

Para facilitar e enriquecer o trabalho desenvolvido, recorremos a diversas ferramentas externas, cada qual com o seu propósito. O seu papel na implementação do resultado final será discriminado em detalhe mais adiante.

As principais ferramentas utilizadas foram:

- **noise (1.2.2)** Utilizado na geração de mapas pseudo-aleatórios;
- **numpy (1.23.5)** Manipulação de matrizes;
- **pygame (2.1.2)** Facilita a interatividade com o software;
- **pygame-menu (4.2.8)** Facilita a criação de interfaces para o usuário;

Importante salientar que estas se tratam de librarias imprescindíveis para executar o programa.

2.2 Ficheiros de Mapas

Os mapas, guardados em ficheiros de texto, são a cruciais para o programa. A sua sintaxe contempla os 4 tipos de ”peças”essenciais para a sua representação, correspondentes às quatro entidades existentes no programa, sendo elas o **jogador** (ou carro), a **parede**, a **pista** e a **linha de meta**, representadas por ”P”, ”X”, ”−”e ”F”, respetiva-

mente.

2.2.1 Geração dos mapas

Apesar de se tratar de uma atividade simples, a criação de mapas com ficheiros de texto é cansativa por ser repetitiva. Mais ainda quando se deseja que os mapas se apresentem mais complexos ou interessantes para a procura de caminhos mais curtos. Tivemos, por isso, a necessidade de desenvolver um gerador de mapas que nos poupasse este esforço e ao mesmo tempo apresentasse mapas coerentes e interessantes no contexto do problema.

Na sua essência, o mapa que pretendemos gerar trata-se de uma matriz cujos valores indicam se é pista ou não. Com isto em mente, podemos popular uma matriz com valores binários aleatórios, i.e. valores que indicam se é pista ou não. O grande problema é que gerar valores puramente aleatórios não iria ajudar a alcançar o resultado pretendido uma vez que queremos um circuito minimamente navegável. Optamos, então, por utilizar um algoritmo que gerasse ruído mais controlado, tendo por base o mesmo conceito que os algoritmos Simplex e Perlin Noise, comummente usados para a geração de mapas em jogos.

Este algoritmo associa valores aleatórios a índices de números discretos e, nos índices de números contínuos, associa valores de transição. Este conceito aplicado a duas dimensões resulta no mapa que se observa na Figura 2.1.

Figura 2.1: Aplicação do algoritmo de ruído

Para obter o resultado desejado resta-nos agora definir um valor *threshold*, i.e. estabelecer um valor limite para o qual uma célula é pista,

desta forma, a nossa matriz terá limites bem definidos para representar o que pertence ao circuito, como se vê na Figura 2.2

Figura 2.2: Aplicação do *threshold* na matriz de valores

Após termos a nossa matriz de valores bastou criar uma função auxiliar para converter os valores nos respetivos caracteres e gerar o ficheiro de texto do mapa, no qual se poderá inserir a localização do player e da meta.

2.2.2 Leitura dos mapas

A leitura dos mapas é possível tendo um qualquer ficheiro de texto que represente um mapa válido, seja ele feito por nós ou pelo gerador. Apesar de não haver uma verificação da validade do mapa, esta é confirmada pelo utilizador conferindo o seu aspeto.

A leitura para memória é feita por um parser que converte o ficheiro de texto numa matriz de valores - esta é necessária na representação visual utilizando o *pygame* para diferenciar as entidades. À medida que lê cada célula da matriz do mapa, cria o nodo no grafo, guardando a sua posição e o seu tipo. Tendo criado todos os nodos adiciona as arestas.

Nesta secção do projeto ponderamos criar um grafo incluindo apenas

nodos "úteis", i.e. onde o carro pudesse navegar, excluindo os nodos que representam o exterior da pista, porém, essa hipótese foi descartada dada a necessidade de verificar colisões com paredes como ditado pelo enunciado. Assim sendo, constam no grafo gerado todos os nodos pertencentes ao mapa, com custo 1 caso perteçam à pista e custo 25 caso estejam fora da pista.

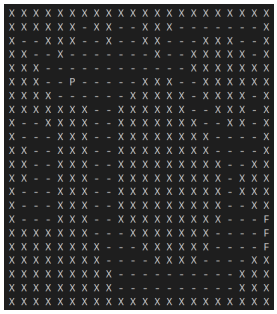

Figura 2.3: Mapa representado por um ficheiro de texto

Figura 2.4: Matriz do mapa representada com *matplotlib*

2.3 Estratégias Adotadas

Numa fase inicial foi-nos solicitada a implementação de um algoritmo de procura, informada ou não informada, ao que optamos por implementar ambas. Já na segunda fase, implementamos mais dois tipos de pesquisa informada. Para a procura informada usamos um algoritmo que entitulamos de *GCA+* que é uma adaptação do algoritmo

Breadth-First (BFS) e para a não informada o algoritmo BFS, lecionado e estudado nas aulas práticas, quer na UC de "Inteligência Artificial"como previamente lecionados em outra linguagem de programação na UC de "Algoritmos e Complexidade". Já numa fase final implementamos de igual forma o algoritmos *Depth-First* (DFS), para uma procura não informada e como informada implementamos os algoritmos mais conhecidos, *Greedy* e *A**.

2.3.1 *Breadth-First*

A implementação deste algoritmo foi bastante similar ao estudado nas aulas práticas, tendo sido apenas sujeito a algumas alterações. Este recebe um nodo de início e uma lista com os diversos nodos de chegada, uma vez que se trata de uma linha de chegada e não apenas um ponto em específico. É utilizado um *set* para guardar os ids de nodos visitados e uma *queue* à qual são adicionados os nodos por visistar. Possui, de igual modo, um dicionário para fazer a relação entre nodo pai e filho, assim sabemos através de que nodo alcançamos um outro. Começando por inserir o ponto de partida na *queue*, vai-se tirando o nodo à cabeça da mesma e verificando qual das adjacências contribuem para o caminho mais curto, colocando a pretendida no dicionário de parents. Este processo é repetido até que a queue esteja vazia ou até encontrar um dos nodos guardados na lista de nodos da meta. Definidos também que apenas prossegue a investigação por determinado nodo se este não for do tipo *WALL* ou seja, não saindo para fora do mapa limite.

Visualização do algoritmo de *breadth-first* em `https://visualgo.net/en/dfsbfs?slide=4`.

2.3.2 *Depth-First*

De igual modo ao algoritmo mencionado acima, o *Depth-First* posuí uma implementação basntante idêntica ao que foi lecionado e realizado durante as aulas práticas iterando de forma recursiva e onde apenas

é acrescentado a restrição de o nodo adjacente é não visitado e não é parede.

Visualização do algoritmo *DFS* em `https://visualgo.net/en/dfsbfs?slide=4`.

2.3.3 *GCA+*

O nosso algoritmo de procura informada é bastante similar ao atualmente implementado na pesquisa não informada, apenas tendo acesso a algumas informações que são chave para o desenvolvimento das suas opções. Das poucas diferenças existentes, é visível que este algoritmo apenas tem acesso ao vizinho que é suposto escolher segundo as restrições de não sair do mapa e segundo as velocidades e acelerações, não como um fator externo mas sim como um fator interno.

2.3.4 *A-star*

De igual forma, foi efetuada uma alteração no que o método retorna e uma restrição acerca do tipo que será melhor explicada seguidamente.

A pesquisa informada efetuada pelo algorito *A-star* recebe de igual forma um ponto inicial e uma lista de possíveis pontos de chegada. Tem uma lista de nodos que já foram visitados mas os seus vizinhos ainda não foram todos visitados e define também um *set* com os nodos que já foram visitados e os seus vizinhos também. Define também um dicionário com as distancias desde o nodo inical até ao final e de forma similar aos outros algoritmos um dicionário para a associação entre nodos i.e nodo ser pai de outro. Primeiramente procura-se na lista de nodos que ainda não tem todos os vizinhos inspecionados qual é que tem menor heuristica, essa procura e/ou cálculo é feito tendo em base a fórmula das distâncias de *Manhattan* onde é a soma do custo do atual caminho com a soma das heuristicas de todos os nodos que o algoritmo já atravessou. As verificações que são efetuadas são dependendo se o nodo está ou não quer na lista de nodos visitados

e vizinhos inspecionados ou não. Finalizando quando encontra um dos nodos presentes na lista de nodos possíveis para ponto de chegada, onde é efetuado a reversão da adjacência de nodos utilizados no caminho, uma vez que assim teremos como cabeça da lista o ponto inicial.

Visualização do algoritmo *A-Star* em `https://qiao.github.io/PathFinding.js/visual/`.

2.4 Interface

A inrterface foi feita utilizando a biblioteca *pygame* e busca mostrar as opções possíveis para o utilizador através de um menu disponibilizado. Dito isso, há dois tipos de menus que são fornecidos: O menu geral e o de criação de mapa.

2.4.1 Geral

Este menu servirá para o início e escolha das opções mais gerais:

- **Mapa**: Para escrever o nome do mapa
- **Carregar Mapa**: Para carregar o mapa especificado
- **Novo Mapa**: Para ser construído um novo mapa
- **Tipo de procura**: Para especificar a estrategia de procura que será adotada
- **Quit**: Para sair do jogo

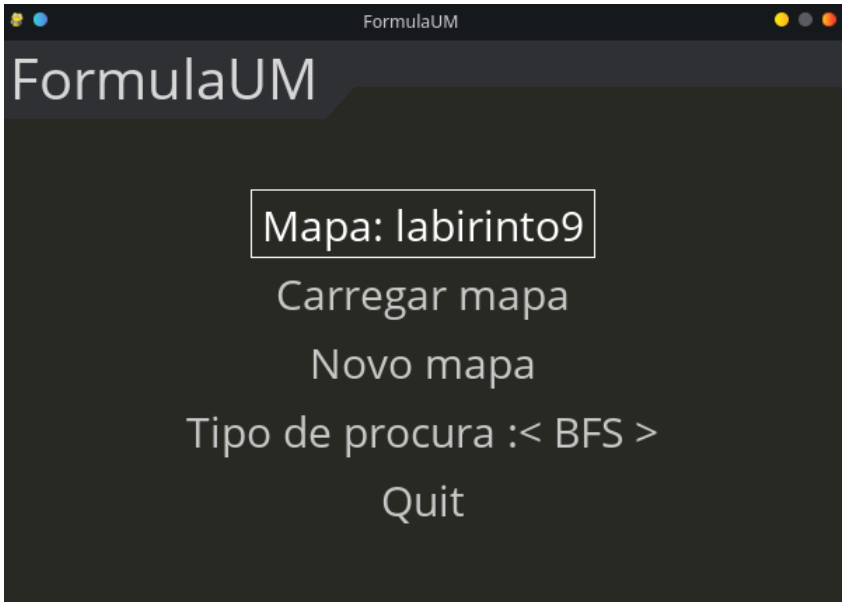

Figura 2.5: Menu Geral

2.4.2 Novo Mapa

O menu de Novo Mapa é responsável pelas definições para serem criados novos mapas.

- **Altura**: Definir a Altura do novo mapa
- **Largura**: Definir a Largura do novo mapa
- **Gerar Mapa**: Para criar este novo mapa

Figura 2.6: Menu Novo Mapa

3 Conclusão

Relativamente à primeira fase do trabalho, o grupo deparou-se com o desafio de desenvolver uma simulação de corrida e calcular o caminho ótimo para um jogador. Graças ao esforço contínuo e ao estudo no âmbito da cadeira, o grupo conseguiu alcançar o objetivo desejado e implementar a primeira fase. Conseguimos de igual forma adiantar mais do que o suposto da primeira fase uma vez que é visível uma *interface* já bastante apelativa e intuítivo ao jogador. Fica assim necessário apenas a implementação de mais algoritmos para obter melhores soluções na procura de caminho desde a posição inicial até uma das posições referentes à linha da meta e também a implementação de múltiplos jogadores. O grupo encontrou-se bastante satisfeito com o resultado alcançado visto que não só implementou requisitos da fase seguinte mas também concluiu os objetivos dados na primeira fase.

Seguindo e argumentando agora sobre a segunda fase, uma vez que já tinhamos algum trabalho e requisitos adiantados, pudemos levar esta parte com mais calma. Iniciando assim pela implentação de multi-jogadores e posteriormente a criação e adaptação de algoritmos de procura informada e não informada. Apesar das iniciais dificuldades na adaptação, uma vez que não seria simplesmente fazer um *loop* para vários jogadores, sendo que estando a percorrer, os outros carros seriam como um "obstáculo dinâmico".

Entretanto, diferente de conferir colisões com as paredes, a colisão entre os carros se mostrou um desafio que não foi superado pelo grupo no momento até a entrega devido a dificuldade de comparar os nodos em que se encontravam no determinado instante. Sendo assim, a possibilidade de colidirem e terem que recalcular o caminho fica como um trabalho futuro.

Apesar disso conseguimos posteriormente adaptar os algoritmos criados e/ou dados nas aulas práticas, acrescentando assim mais alguns para a nossa diversidade para a procura entre a posição inicial e a meta final.

Em suma, mais uma vez o grupo encontra-se satisfeito e bastante contente com o resultado final obtido deixando para um possível trabalho futuro a implementação da execução em tempo real dos algoritmos assim como as colisões entre os carros.