Mummy Maze

Alexandre Jerónimo
Instituto Politécnico de Leiria, ESTG
Campus 2 - Morro do Lena, Alto do Vieiro, Apt 4163, Edifício D
2411-901 Leiria
2201799@my.ipleiria.pt

Gonçalo Paulino
Instituto Politécnico de Leiria, ESTG
Campus 2 - Morro do Lena, Alto do Vieiro, Apt 4163, Edifício D
2411-901 Leiria
2201803@my.ipleiria.pt

Palavras-chave

Tile corresponde a quadrículas onde podem existir entidades, como inimigos, herói, armadilhas, etc.

Sub-tile corresponde ao espaço entre cada *tile*, onde existem as portas e as paredes.

1. INTRODUÇÃO

1.1 Mummy Maze

O Mummy Maze é um jogo composto por vários níveis, onde o herói tem o objetivo de chegar à saída evitando quaisquer perigos que possam ocorrer: ser morto por um inimigo ou cair numa armadilha. O jogo é jogado por turnos, onde o herói se move primeiro e em seguida deslocam-se os inimigos.

1.2 Objetivo do projeto

Este projeto tem como finalidade o desenvolvimento de uma aplicação que, recorrendo a determinados algoritmos de procura informados, seja capaz de jogar o jogo Mummy Maze.

Através do uso destes algoritmos, será realizado um estudo comparativo do desempenho dos vários algoritmos de procura, bem como das heurísticas implementadas.

2. REPRESENTAÇÃO DOS ESTADOS

Optámos por implementar um enumerado para tratar as diferentes categorias de *tile* e *sub-tile*. Considerámos as seguintes representações:

2.1 Entidades

Tabela 1. Entidades e respetivas representações

Nome	Identificador	Representação
Herói	HERO	H
Múmia branca	WHITE_MUMMY	M
Múmia vermelha	RED_MUMMY	V
Escorpião	SCORPION	E
Armadilha	TRAP	A
Chave	KEY	C
Vazio[1]	EMPTY	.
Saída[2]	EXIT	S

2.2 Estados secundários

Um estado secundário é um estado que existe entre entidades.

Tabela 2. Estados secundários e respetivas representações

Nome	Identificador	Representação
Porta vertical aberta[3]	V_DOOR_OPEN	‘
Porta vertical fechada[3]	V_DOOR_CLOSED	(
Porta horizontal aberta	H_DOOR_OPEN	_
Porta horizontal fechada	H_DOOR_CLOSED	=
Parede vertical	V_WALL	|
Parede horizontal	H_WALL	_
Passagem livre[4]	OPEN	*espaço*

3. MÉTODO DE IMPLEMENTAÇÃO

Como algumas das regras do funcionamento do jogo eram vagas, tivemos de tomar certas decisões na implementação do projeto. Destas decisões constam:

A possibilidade de todos os inimigos abrirem portas ao passar por cima de uma chave;

Sempre que duas múmias lutam, ganha a que chegou à *tile* em primeiro lugar. Numa luta entre uma múmia e um escorpião, estava já definido que ganhava a múmia.

Os inimigos movem-se na seguinte ordem após uma ação do herói: primeiro avançam todos os escorpiões, depois todas as múmias vermelhas e no final as múmias brancas.

Todas as interações dos inimigos (lutas, chaves, herói) são efetuadas imediatamente após um movimento. Por exemplo, no final de cada um dos movimentos das múmias que se movem duas vezes, são efetuadas um número de verificações para controlar e agir, quando necessário, conforme o resultado dessas interações.

---

[1] Vazio corresponde a uma tile sem nada, livre para movimento.
[2] A saída aparece fora do tabuleiro.
[3] Este estado secundário permite a passagem do herói e de inimigos.
[4] Passagem livre corresponde a um estado entre entidades em que não existe nem portas nem paredes, ou seja, permite sempre a passagem do herói e de inimigos.

4. HEURÍSTICAS

4.1 Distância do herói ao objetivo

Esta heurística avalia o quão perto o herói está da saída – quanto mais perto, menor é a heurística.

A implementação consiste na soma da diferença entre a coluna do herói e coluna da saída e da diferença entre a linha da saída e a linha do herói.

4.2 Distância do inimigo mais próximo

Esta heurística avalia o quão longe o herói está do inimigo mais próximo – quanto mais longe, menor é a heurística.

A implementação consiste na soma da diferença entre a coluna do herói e coluna do inimigo e da diferença entre a linha do inimigo e a linha do herói, para todos os inimigos.

Escolhendo o menor destes valores, subtrai-se o menor desses valores ao valor máximo possível da distância de modo que quanto mais longe o inimigo estiver, menor a heurística.

4.3 Junção das duas heurísticas anteriores

Esta heurística soma as duas heurísticas anteriores, de modo a avaliar tanto a distância do herói ao objetivo como a distância ao inimigo mais próximo.

5. RESULTADOS OBTIDOS

Nós consideramos um teste como uma combinação única de um algoritmo de pesquisa com uma heurística (quando aplicável), de modo que em cada nível foram executados 13 testes diferentes (4 algoritmos não informados + (3 algoritmos informados x 3 heurísticas disponíveis).

Decidimos criar três gráficos para que pudéssemos tirar conclusões a partir dos mesmos: Tamanho Máximo da Fronteira para cada Teste em cada Nível, Nós Explorados por cada Teste em cada Nível, Duração em Milissegundos de cada Teste em cada Nível. Ao realizar esta tarefa apercebemo-nos de que existiam *outliers* que estavam a dificultar este processo, por isso decidimos omiti-los: no segundo gráfico omitimos os algoritmos *Depth-first Search* e *Iterative Deepening Search* e o nível 16 por completo, no terceiro gráfico omitimos o algoritmo *Iterative Deepening Search* em todos os níveis e o algoritmo *Depth-first Search* nos níveis 15, 17 e 21v1.

O algoritmo Iterative Deepening Search no nível 21v1 não encontrou solução (como esperado) contudo suspeitamos que tenha entrado num ciclo infinito pelo que tivemos que o parar manualmente, pois já estava a correr há 8 horas na tentativa mais recente.

5.1 Conclusões

Concluímos que o tamanho máximo da fronteira é maior nos seguintes casos: no algoritmo *Depth First Search*, *A** que utiliza a heurística de saída ou uma descendente dessa mesma, e o algoritmo *Greedy Best Search* que utiliza a heurística dos inimigos.

O IDS e o *Depth First Search* são os que demoram mais tempo a executar em todos os testes tornando-se *outliers*.

Dentro dos casos normais, o IDA* é o que demora mais tempo.

6. CONTRIBUIÇÃO DE CADA MEMBRO

Optámos por distribuir o projeto em diferentes etapas de desenvolvimento.

6.1 Alexandre

Leitura dos ficheiros de níveis e preenchimento da matriz.

Implementação do movimento das múmias

Implementação da "condição de vitória" do herói em cada nível.

Implementação do escorpião e do movimento dos mesmos.

Planeamento e implementação das diferentes interações do herói e das portas com o ambiente.

Implementação do suporte para múltiplas portas.

Desenvolvimento do botão de gerar resultados para um nível em específico.

6.2 Gonçalo

Implementação das ações possíveis do herói, e do seu movimento.

Otimização do movimento do herói e múmias.

Implementação de armadilhas, paredes e portas.

Planeamento e implementação das diferentes interações dos inimigos com o ambiente.

Implementação das lutas entre inimigos.

Implementação do suporte para múltiplos inimigos.

Desenvolvimento do botão de gerar resultados de todos os níveis.

6.3 Em conjunto

Desenvolvimento do relatório.

Otimização dos algoritmos de pesquisa

Implementação das diferentes heurísticas.

Resolução de bugs no trabalho de ambos.

Análise e realização dos diferentes testes necessários para os resultados que obtivemos.

7. FUNCIONALIDADES EXTRA

Melhorámos o código dos algoritmos fornecidos na aula.

Desenvolvemos um botão "Test all levels" que procura uma solução em todos os níveis com todos os métodos de pesquisa e todas as diferentes heurísticas, criando uma pasta com um ficheiro CSV para cada nível, que contém os resultados dos testes nesse mesmo nível. Entrando em mais detalhe:

O programa começa por pedir ao utilizador para selecionar a diretoria onde será guardada a pasta referida acima e, após isso, efetuar a operação descrita acima.

Caso a execução das pesquisas seja interrompida, informa no ficheiro de output em que trabalhava que a execução foi interrompida.

O nome da diretoria criada é a data e hora no seguinte formato: "dd_MM_yyyy-HH_mm_ss" seguidos de um número aleatório de cinco dígitos, de modo a evitar conflitos e a ajudar na identificação de cada teste. O nome de cada ficheiro dentro dessa diretoria é o mesmo que o ficheiro do nível, mas com a extensão ".csv".

Desenvolvemos também um botão que realiza todos os testes para um nível selecionado pelo utilizador, para permitir que pudéssemos ter várias instâncias do programa a correr e a testar diferentes níveis em simultâneo. Funciona de forma muito similar ao botão descrito acima tendo apenas as seguintes diferenças: O programa apenas pede ao utilizador o nível que deseja testar e, realizando os testes da mesma forma que o botão acima, guarda a informação dos mesmos num ficheiro na pasta “Testes” com a seguinte nomenclatura: o nome do ficheiro do nível, seguido da data e hora, seguidos de um número aleatório de cinco dígitos.

8. Anexos

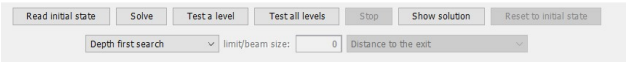

Ilustração 1 - Layout dos botões

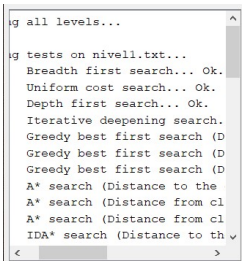

Ilustração 2 - Output da operaçao "Test all Levels"

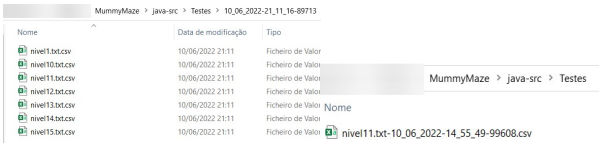

Ilustração 3 e 4 - Ficheiros com os resultados de um "Test all Levels” (esquerda) e de um “Test a level” (direita)

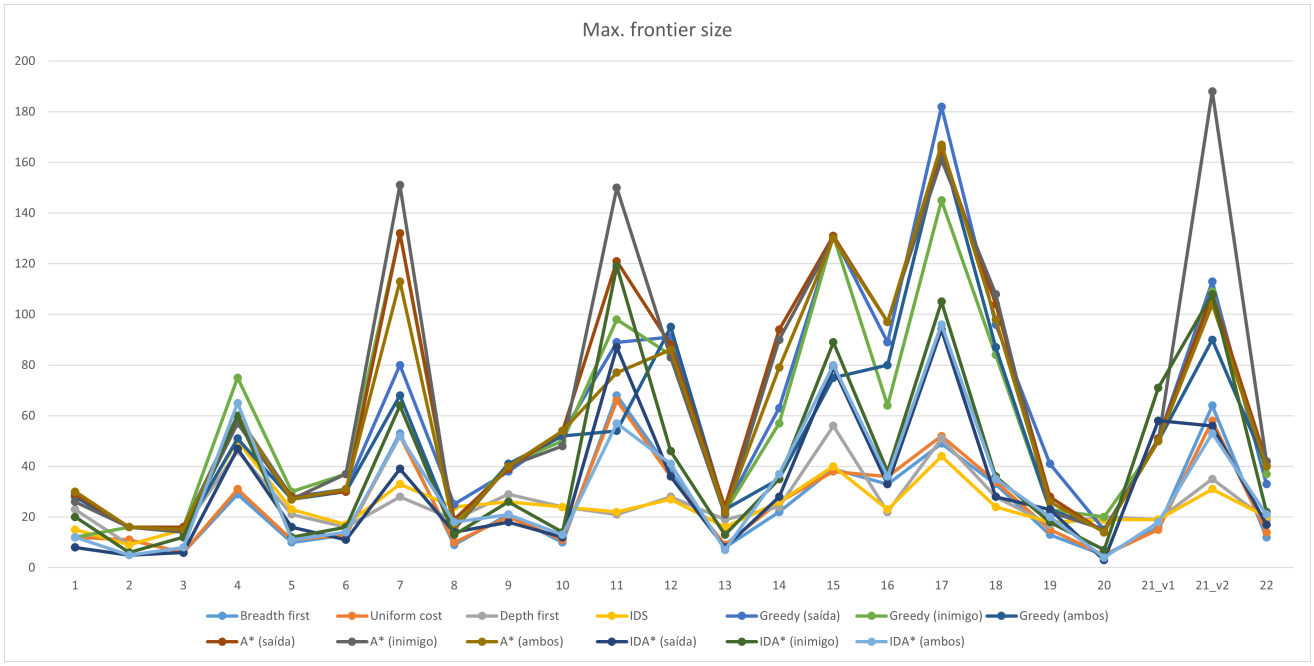

Gráfico 1 - Tamanho Máximo da Fronteira para cada Teste em cada Nível

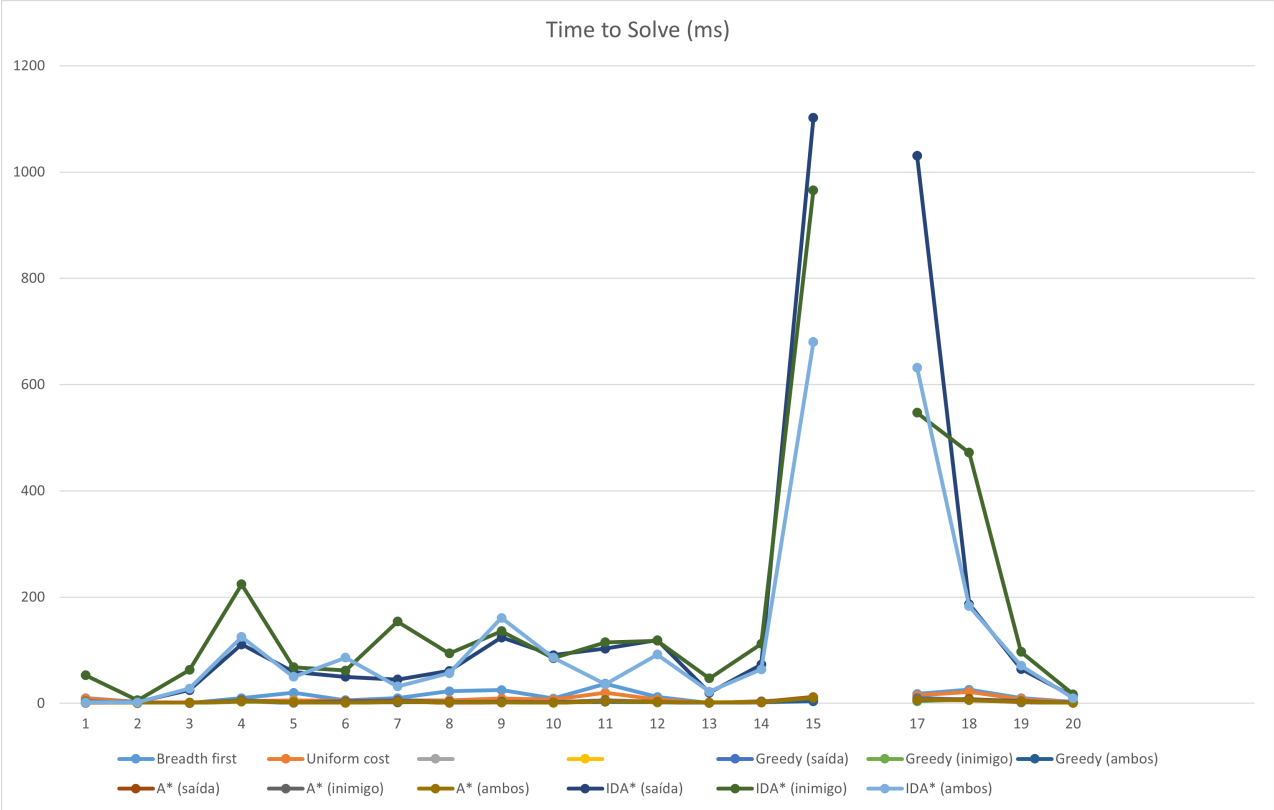

Gráfico 2 - Duração em Milissegundos de cada Teste em cada Nível

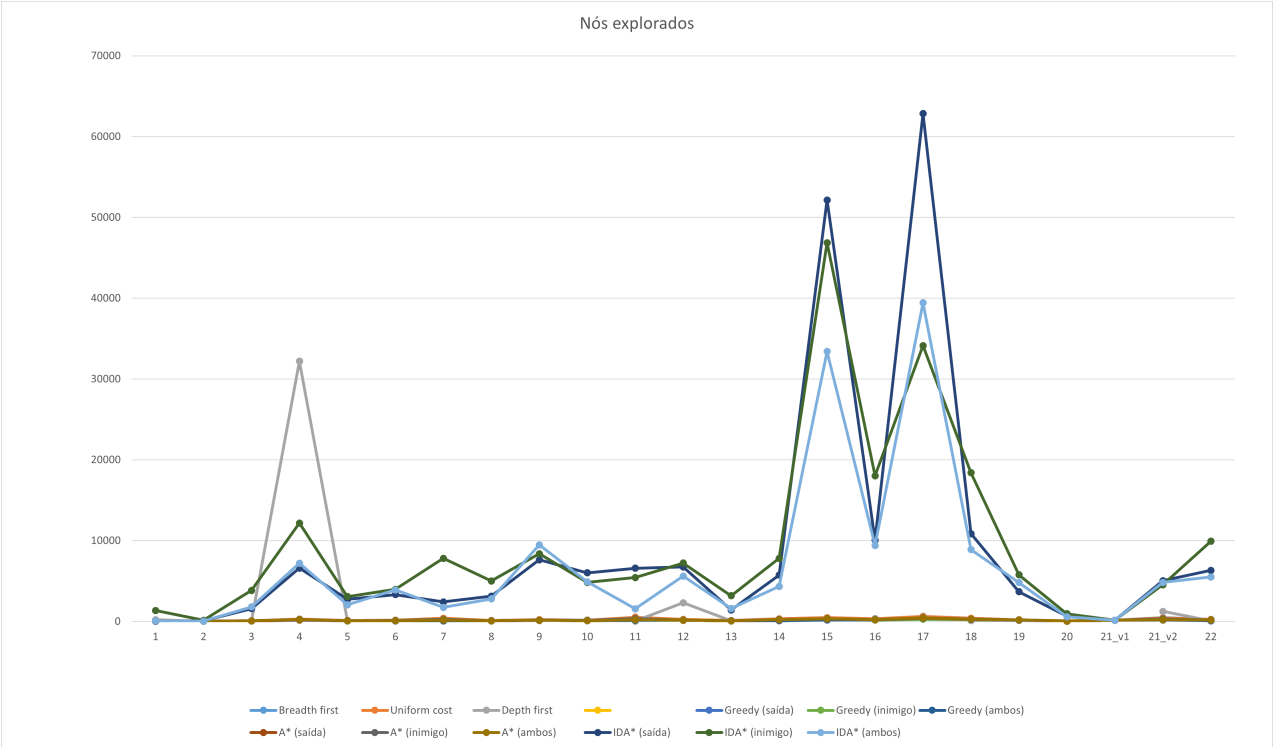

Gráfico 3 - Nós Explorados por cada Teste em cada Nível