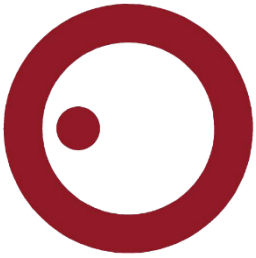
Projeto Mummy Maze Solver

Inteligência Artificial

Anabela Bernardino

Carlos Grilo

Rolando Miragaia



JOAQUIM RODRIGUES 220491

Contents

Introdução

O Mummy Maze é um jogo onde o herói é um caçador de tesouros. Enquanto busca por tesouros perdidos, a personagem principal do jogo vai atravessando vários níveis onde tem de evitar ser
apanhado por inimigos e pisar armadilhas.

Em cada nível ou câmara está uma passagem para o nível ou câmara seguinte. Desta forma, o objetivo do herói é deslocar-se para a zona de acesso ao próximo nível evitando quaisquer perigos
que possam ocorrer: ser morto por um inimigo ou cair numa armadilha. A zona de acesso ao nível seguinte é a célula que tem uma escada adjacente.

Este jogo é jogado por turnos, onde primeiro se move o Herói e em seguida se deslocam os inimigos. Em cada turno, o Herói é sempre o primeiro a deslocar-se e os inimigos são sempre os últimos, mesmo que aquele já tenha chegado à célula objetivo.

Recursos

Para a realização de testes foi utilizado um portátil Asus Zenbook com as seguintes especificações:

- Processador: AMD Ryzen 7 4700U with Radeon Graphics 2.00 GHz
- RAM: 16GB
- Sistema operativo: Microsoft Windows 10 Home
- Versão do Sistema operativo: 10.0.19044 N/A Build 19044

Problema

O herói tem de chegar até à saída sem ser apanhado por um dos inimigos (múmias brancas, múmias vermelhas e escorpiões) e sem pisar as armadilhas espalhadas pelo campo.

Solução

O herói verifica se pode mover-se para a direção que deseja (cima, baixo, esquerda, direita) e valida as seguintes situações:

O herói faz o processo descrito anteriormente até:

- Chegar ao objetivo dele, neste caso é a saída.
- Ser morto por 1 dos inimigos ou pisando uma armadilha.

Caso o herói não encontre caminho possível para a saída, o programa termina sem solução encontrada.

Estrutura de Dados

Este projeto para representar os objetos contém a seguinte estrutura de classes:

- MatrixPosition
  - Enemy
    - RedMummy
    - Scorpion
    - WhiteMummy
  - Item
    - Key
    - Trap
  - Obstacles
    - Door

Contém também variáveis únicas:

- lineHero e columnHero
- lineExit e columnExit

Estado

A representação do estado do problema encontra-se na classe MummyMazeState.

O herói tem à sua disposição 5 ações disponíveis:

- ActionUp
- ActionDown
- ActionLeft
- ActionRight
- ActionStay

O herói começa por realizar os métodos "isValid" relacionados com as suas ações disponíveis. Foi criado os metódos canMoveUp, canMoveDown, canMoveLeft, canMoveRight para verificar as seguintes condições:

- Verifica se está na última casa disponível para a direção que quer ir.
- **Só avança** nessa direção se a próxima posição for a saída.
- Caso não seja a saída e não seja uma posição fora do puzzle, faz as seguintes validações:
  - Verifica se existe uma parede entre a posição em que se encontra e a posição que pretende ir.
  - Verifica se a posição para onde quer ir é uma casa vazia ou uma chave.

Se o método "isValid" da ação devolver true, vamos realizar as funções de mover o herói nessa direção:

- moveUp
- moveDown
- moveLeft
- moveRight

Se nenhuma das 4 ações de mover retornar true no seu método isValid, o herói pode optar por escolher ficar na mesma casa.

Após o herói mover-se, é a vez dos seus inimigos se moverem, através do método moveEnemies.

No método moveEnemies, começamos a mover os inimigos na seguinte ordem:

1. Múmias brancas
2. Múmias vermelhas
3. Escorpiões

Cada múmia tem 2 turnos para se mover, enquanto os escorpiões apenas se movem 1 vez.

As múmias brancas e os escorpiões começam por tentar mover-se de forma horizontal primeiro, e depois de forma vertical.

As múmias vermelhas fazem o oposto, tentam mover-se primeiro de forma vertical e depois de forma horizontal.

Para tornar os métodos genéricos é usado um booleano para verificar se o herói se pretende mover de forma positiva (baixo, direita) ou de forma negativa (cima, esquerda) na matriz.

Caso 1 inimigo morra, **as suas variáveis line e column passam ao valor "-1"** para evitar futuras interações.

Múmias podem matar o herói, outras múmias e outros escorpiões, enquanto o escorpião apenas pode matar o herói e outros escorpiões.

Cada vez que é feito 1 movimento pelo herói ou pelos inimigos é necessário fazer uma atualização da posição onde estava e da posição para onde vai.

Existem os seguintes cenários:

- Herói
  - Pisa uma armadilha, **o herói morre**, ou seja, a sua linha e a sua coluna passam a -1 e o programa termina
  - Pisa uma chave, as portas reagem, se estão abertas passam a fechadas, se estão fechadas passam a abertas.
  - Sai da posição onde estava a chave, a chave volta a reaparecer no puzzle.
- Inimigos
  - Pisar armadilhas, não acontece nada a nível de jogabilidade, as armadilhas desaparecem e quando sai de cima da armadilha voltam a reaparecer.
  - Pisar chaves, se for múmia faz o mesmo processo que o herói, se for escorpião as portas não reagem.

As interações dos inimigos estão a ser realizadas nos métodos **mummyFoundSomethingOnTargetPosition** e **scorpionFoundSomethingOnTargetPosition**.

Heurísticas

Para este projeto foram implementadas 2 heurísticas:

- HeuristicTileDistance
- HeuristicTilesDistanceToClosestEnemy

HeuristicTileDistance

Calcula a distância do herói para o seu objetivo, que neste caso, é a saída.

A heurística é feita através da seguinte formula:

(LinhaHeroi - LinhaSaida) / 2.0 + (ColunaHeroi - ColunaSaida) / 2.0

É aplicado a divisão por 2 para subtrair as linhas e colunas que estão entre as posições ao qual o herói pode andar. São posições onde colocamos os obstáculos como portas e paredes.

HeuristicTilesDistanceToClosestEnemy

Calcula a distância do herói para os seus inimigos, que neste caso, e guarda o valor da heurística do inimigo mais próximo.

Quanto mais perto está o inimigo, maior é o seu valor de heurística.

A heurística é feita através da seguinte formula:

1 / ((LinhaHeroi - LinhaInimigo) / 2.0 + Math.abs(ColunaHeroi - ColunaInimigo) / 2.0)

É aplicado a divisão por 2 para subtrair as linhas e colunas que estão entre as posições ao qual o herói pode andar. São posições onde colocamos os obstáculos como portas e paredes.

Extras

Foi implementado uma criação de 1 ficheiro .xlsx que permite verificar estatísticas relacionadas à resolução do problema.

O ficheiro consiste em N spreadsheets com os resultados de cada algoritmo de procura realizado. O nome da spreadsheet será o nome do ficheiro do nível do utilizador.

Caso o ficheiro não exista é criado no diretório do projeto, dentro da pasta “stats”.

Caso o ficheiro exista, ele reescreve a linha respetiva à pesquisa feita.

Exemplo:

Nivel1.txt

	A	B	C	D	E	F	G
1	Search Name	Cost	Expanded Nodes	Frontier	Generated States	Heuristic	
2	Beam search	No solution found	1	0	5	Tiles distance to final position	
3	Beam search	No solution found	1	0	5	Tiles distance to Closest Enemy position	
4	IDA* search	No solution found	0	0	1	Tiles distance to final position	
5	IDA* search	No solution found	0	0	1	Tiles distance to Closest Enemy position	
6	Breadth first search	11.0	73	14	198		
7	Uniform cost search	11.0	73	13	198		
8	Depth first search	13.0	420	23	661		
9	Limited depth first search	13.0	209	23	337		
10	Iterative deepening search	11.0	3760	15	7254		
11	Greedy best first search	11.0	17	29	66		
12	A* search	11.0	18	29	70	Tiles distance to final position	
13	A* search	11.0	48	28	176	Tiles distance to Closest Enemy position	
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30							
31							
32							
33							
34							
35							
36							

nivel1.txt | nivel2.txt | nivel3.txt

Conclusão

Foram testados 22 problemas com os vários algoritmos de pesquisa, onde podemos observar podemos observar que o algoritmo com melhor custo em todos é o algoritmo A*.

Os testes podiam ter sido feitos com melhor precisão e detalhe, mas devido à falta de tempo não foi possível.

Os testes estão disponíveis dentro da pasta “stats” com o nome “mummymazeStatsCOMPLETE19_06_2022”.

Bibliografia

[1] Apache poi

https://poi.apache.org/

[2] Slides da Unidade Curricular de Inteligência Artificial

[3] Projeto Puzzle8 realizado nas aulas