Mummy Maze

Martim Silva

Nº 2200681

Samuel Martins

Nº 2200674

***ABSTRACT***

This report aims to present the implementation of the Mummy Maze carried out within the scope of the Artificial Intelligence curricular unit.

Throughout this report is intended to present a brief explanation about our methods of implementation, what was implemented, the contribution of each member and all the developed extras.

**Palavras-chave**

Keywords are your own designated keywords.

# INTRODUÇÃO

## Mummy Maze

Lançado em 2002, o jogo *Mummy Maze* é um jogo de estratégia por turnos cujo objetivo é levar o herói até à saída do nível, sem que este seja apanhado e morto pelos perigos presentes (inimigos e/ou armadilhas).

Sendo este um jogo por turnos, começa-se por mover o herói e de seguida movem-se os inimigos.

## Objetivo do projeto

Recorrendo aos algoritmos de procura (informados e não informados) lecionados nas aulas, este projeto tem como objetivo jogar o jogo Mummy Maze de forma automática, para que seja possível realizar um estudo comparativo do desempenho dos vários algoritmos e heurísticas implementadas no projeto.

# REPRESENTAÇÃO DOS ESTADOS

## Níveis

Na base dos diferentes níveis disponíveis está uma matriz 13x13, lida a partir de um ficheiro de texto (.txt) que contém as diferentes entidades e estados. Estes elementos são representados por caracteres que constituem o nível.

## Entidades

As entidades correspondem aos elementos posicionados nas células principais (chão) da matriz. Estes elementos podem-se mover de célula a célula se existirem condições para o fazer.

Tabela 1. Entidades e seus identificadores

|  |  |  |
| --- | --- | --- |
| **Nome** | **Identificador** | **Caractere representativo** |
| Herói | HERO | H |
| Chão | FLOOR | . |
| Saída | EXIT | S |
| Múmia Branca | WHITE\_MUMMY | M |
| Múmia Vermelha | RED\_MUMMY | V |
| Escorpião | SCORPION | E |
| Armadilha | TRAP | A |
| Chave | KEY | C |

## Estados Secundários

Estes estados são aqueles que se encontram entre as posições das entidades e representam elementos que não se movem. No entanto, alguns podem sofrer alterações.

Tabela 2. Estados secundários e seus identificadores

|  |  |  |
| --- | --- | --- |
| **Nome** | **Identificador** | **Caractere representativo** |
| Porta Vertical aberta | VERT\_DOOR\_OPEN | ) |
| Porta Vertical fechada | VERT\_DOOR\_CLOSED | “ |
| Porta Horizontal aberta | HORIZ\_DOOR\_OPEN | \_ |
| Porta Horizontal fechada | HORIZ\_DOOR\_CLOSED | = |
| Parede Vertical | VERT\_WALL | | |
| Parede Horizontal | HORIZ\_WALL | - |

# ALGORITMOS DE PROCURA

Neste projeto foram implementados os algoritmos de procura estudados nas aulas práticas.

Algoritmos de procura não informados:

* Procura em Largura (*Breadth First Search*);
* Procura Uniforme (*Uniform Cost Search*);
* Procura em Profundidade (*Depth First Search*);
* Procura em Profundidade (*Depth Limited Search*);
* Procura por Aprofundamento Progressivo (*Iterative Deepening Search*).

Algoritmo de procura informados:

* Procura Sôfrega (*Greedy Best First Search*);
* Procura em Feixe (*Beam Search*);
* A\* (*AStarSearch*);
* IDA\* (*IDAStarSearch*).

# IMPLEMENTAÇÃO

Ao longo do projeto, foram tomadas várias decisões que não constavam no enunciado do projeto. Neste capítulo são descritas essas decisões.

## Classe Cell e Agent

Para uma melhor gestão dos elementos do jogo, foram criadas duas novas classes:

A classe *Cell*, é responsável por: armazenar o caracter que representa o elemento, e a posição do mesmo na matriz (linha e coluna).

A classe *Agent*, foi desenvolvida para ser utilizada na criação do herói e dos inimigos. Esta classe, conta com uma variável booleana para verificar se o herói (ou inimigo) está vivo. Esta variável é utilizada mais tarde para remover inimigos das suas listas caso, tenham morrido em lutas.

## Movimentação dos inimigos e herói

Para a ordem de movimentação dos inimigos, optámos por: primeiro moverem-se as múmias brancas, seguido das múmias vermelhas e por fim os escorpiões.

Por cada movimentação, é verificado se existem chaves ou armadilhas que necessitam de ser repostas na matriz, caso a *Cell* não esteja a ser ocupada por um inimigo ou herói. Outra verificação que é feita após cada movimentação, é a verificação se o inimigo ou herói está em cima de uma chave, para assim proceder à mudança de estado (abrir ou fechar) de todas as portas do nível.

Por fim, depois de herói e todos os inimigos terminarem os seus turnos, é removido das listas de inimigos, aqueles que perderam lutas.

## Luta entre inimigos

Não havendo regras explicitas no enunciado, decidiu-se que cada inimigo pode matar qualquer outro tipo de inimigo. Para além disso, o sistema luta foi implementado de forma a matar o inimigo que se está a mover para cima de outro inimigo.

# HEURÍSTICAS

## Distância ao objetivo

Esta primeira heurística avalia a distância a que o herói se encontra da saída. Sendo que, quanto mais próximo o herói se encontrar da saída, melhor o resultado.

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

## Distância aos inimigos

Esta heurística calcula a distância entre o herói e inimigo mais próximo. Para esta heurística, foi tomada em consideração que quanto mais perto o inimigo estiver do herói, pior o resultado.

A implementação da heurística consiste em calcular a distância do inimigo mais próximo do herói. Para isso, são percorridas todas as listas de inimigos, armazenando a distância numa variável auxiliar. Finalmente, é subtraído da distância máxima possível na matriz, o valor armazenado na variável auxiliar. Caso não existam inimigos, ou o herói estiver na saída, a heurística devolve zero.

## Distância ao objetivo e aos inimigos

Esta heurística consiste na soma das duas heurísticas mencionadas anteriormente. Assim, com o reaproveitamento destas heurísticas, é possível avaliar a distância ao objetivo e a distância aos inimigos simultaneamente.

# RESULTADOS OBTIDOS

Com esta implementação foi possível encontrar solução para todos os níveis os disponíveis, exceto o nível “nivel21\_v1”.

Na secção dos anexos estão disponíveis os resultados gravados em ficheiro Excel, obtidos pela funcionalidade extra de testes.

\*\*Falar de forma geral sobre o desempenho dos algoritmos (quais apresentam custos menores e afins)?

# CONTRIBUIÇÃO DE CADA ELEMENTO

## Martim Silva

Movimentação das múmias

Lutas entre inimigos

Implementação das portas

## Samuel Martins

Movimentação dos escorpiões

Reposição de elementos na matriz (chaves e armadilhas)

## Em conjunto

Relatório

Heurísticas

Leitura do estado inicial do nível

Algoritmo IDA\*

Otimização do algoritmo *A\** e *GreedBestFirstSearch*

Teste automático de níveis

# EXTRAS

## Algoritmo IDA\* (*IDAStarSearch*)

Dado que este algoritmo não foi implementado nas aulas, e de forma a completar o estudo, foi decidido desenvolver o algoritmo *IDA\** (*IDAStarSearch*).

## Melhoria de alguns algoritmos

Foram feitas algumas alterações aos algoritmos *A\** e *GreedBestFirstSearch*, tornando-os mais eficientes.

## Teste de um nível

De forma a simplificar o estudo do desempenho dos algoritmos, desenvolveu-se uma funcionalidade que permite testar um nível de forma automática. Esta funcionalidade, ao carregar num botão, torna possível testar o nível selecionado com todos os métodos de pesquisa e heurísticas (no caso dos algoritmos de pesquisa informados), guardando os resultados num ficheiro Excel (.csv) na diretoria “tests”, também gerada automaticamente, com todas as estatísticas relevantes ao estudo.

# Anexos

\*\*NOTA: Não é uma imagem, mas sim um “objeto” criado a partir do ficheiro de teste obtido pela funcionalidade.

\*\*NOTA2: O tamanho é muito pequeno?

