IA Projeto 1

Search with adversary - 18 Ghosts

Trabalho realizado por:
Inês Gaspar
João Félix
Lourenço Gonçalves

Especificação do jogo

- Este jogo consiste num tabuleiro de 5x5, em que cada jogador tem 9 fantasmas de 3 cores diferentes disponíveis para colocar no tabuleiro.
- O objetivo é um dos jogadores conseguir passar um fantasma de cada cor pelo portal. Contudo, os portais rodam e para ser possível o fantasma atravessar o portal este tem que estar virado na direção do fantasma.
- No início do jogo, os fantasmas têm que ser colocados em casas com a mesma cor que a sua
- Um fantasma pode lutar com outro fantasma (da sua equipa ou da do adversário) se se mover para uma casa que já está ocupada.
- No tabuleiro existem casas com espelhos que permitem aos fantasmas que se movem para essas casas passarem para outra casa do tabuleiro que também contenha espelhos.

Referências

- Encontramos o seguinte link que contém uma descrição completa das regras do jogo:
    - https://tesera.ru/images/items/25338/18GHOSTS_EN_r1.pdf

Formulação como um problema de pesquisa

- **Representação do estado:** o tabuleiro é um array de dicionários que guarda informação sobre a posição em que se encontra (tipo da peça, cor, etc). Guarda-se também o número e cor dos fantasmas de cada jogador nas masmorras e que já escaparam
- **Estado inicial:** board com as casas todas vazias e o todos os fantasmas na masmorra (que é o valor inicial de cada jogador)
- **Objetivo:** pelo menos um fantasma de cada cor atravesse o portal

- **Operadores (nomes, pré-condições, efeitos e custo):**
- Move:
  - **pré-condições:** respeitar os limites do tabuleiro, mover apenas uma casa na ortogonal ou de espelho a espelho e a posição destino não conter um fantasma da mesma cor;
  - **efeitos**: alteração da posição no tabuleiro, possivelmente morrer, matar outro fantasma ou escapar;
  - **custo**: aumentar/diminuir em uma unidade a distância total ao portal.
- Place_from_dungeon;
  - **pré-condições:** haver fantasmas na masmorra e espaços vazios no tabuleiro com a cor do fantasma que se quer colocar;
  - **efeitos**: remover um fantasma da masmorra e ocupar um espaço do tabuleiro;
  - **custo**: aumentar/diminuir a distância ao portal da cor do fantasma.
- **Heurística/função de avaliação:** função heurística tem como critério o número de casas que faltam ao fantasma para chegar ao portal, sendo que se aplica a fantasmas de cores que ainda não tenham sido libertados.

Implementação do que fizemos até agora

- O tabuleiro e os fantasmas definidos, bem como as posições de cada casa guardadas na estrutura de dados que serve de representação do jogo.
- As funções de movimento e colocação dos fantasmas no tabuleiro, já com a verificação se os movimentos são válidos.
- Rotação dos portais que os fantasmas têm que atravessar.
- Estamos a desenvolver o Projeto no Pycharm, estamos a implementá-lo em Python com o auxílio do Pygame.

Algoritmos implementados

Os algoritmos de pesquisa com adversários que implementamos foram os seguintes:

- **Minimax com cortes Alpha-Beta e ordenação de movimentos**
- **Monte Carlo Tree Search**

As heurísticas que utilizamos para estes dois algoritmos foram as mesmas:

- Número de casas de distância entre o fantasma e o portal da mesma cor que o fantasma (isto referente a fantasmas de cores que ainda não tenham fugido), bem como o número de fantasmas que escaparam até ao momento.

Análise de Resultados - Minimax vs MCTS

Relativamente à análise de resultados para os algoritmos que implementamos para o jogo, consideramos que o mais interessante de avaliar seria o número de vitórias de um algoritmo em relação ao outro.

Assim, verificamos que o Minimax apresenta melhores resultados do Monte Carlo Tree Search para a heurística que definimos. É também importante salientar que o Minimax implementado é muito eficiente por causa da ordenação dos movimentos possíveis que melhora o desempenho relativo aos cortes Alpha-Beta e desta forma reduz o tempo de execução do algoritmo.

Tal pode ser explicado devido às heurísticas utilizadas, visto que no Minimax quando o fantasma tem possibilidade de escapar pelo portal opta por realizar esse movimento, enquanto que o Monte Carlo Tree Search nem sempre opta por essa, o que o leva a numa última instância a perder contra o Minimax.

Para recolhermos estas conclusões, executamos 7 vezes o jogo no modo Computador - Minimax vs MCTS, nos quais o vencedor foi sempre o Minimax (tendo em conta que antes da jogada vencedora o jogo estava empatado, em 5 dos 7 casos testados).

Assim, concluímos que possivelmente a escolha de uma heurística diferente para o MCTS poderia trazer melhores resultados para este algoritmo, principalmente depois de dois dos seus fantasmas já terem escapado.

Conclusão

Com este Projeto, conseguimos perceber melhor as diferenças entre os algoritmos de pesquisa com adversários, bem como a importância da escolha e teste de várias heurísticas de forma a chegarmos ao melhor resultado. Ajudou-nos também a verificar o impacto que estes algoritmos têm na performance global dos jogos e de formas de tentar torná-los mais eficientes e menos dispendiosos em termos de recursos.