Inteligência Artificial

Universidade de Évora
19 de Junho de 2020

Diogo Rafael nº 37859 e João Conceição nº 38052

Jogo Ouri

Implementação

Neste trabalho optámos por tratar o problema com duas listas, uma para cada jogador, e os pontos de cada jogador.
Em cada posição da lista, temos um buraco que contém as suas coordenadas(jogador e posição) também como quantas sementes tem.
Para efetuar uma jogada, fazemos os seguintes passos:

- primeiro verificamos as regras do jogo, neste caso, em quais casas podem ser efetuadas jogadas considerando a **regra de não poder jogar em casas de 1 semente na presença de casas com mais sementes** e a **regra de caso o campo do adversário estar vazio, temos de jogar de forma a mandar sementes para o outro lado**, efetivamente cortando todas as jogadas que sejam contra as regras.

- pegamos nas **X** sementes de um buraco e distribuimos pelos seguintes **X** buracos, o sentido em que a lista anda depende do jogador inicial(é feita uma inversão de sentido quando as sementes chegam ao limite da lista ou quando muda o jogador).

- recolhemos os pontos efetuados na jogada, **SE** houve pontos efetuados.

- o **estado terminal** é detetado caso nenhuma das regras possam ser cumpridas(ou haja conflito nas regras) ou caso não haja a possibilidade de fazer quaisqueres jogadas "produtivas"(ambos os jogadores não podem passar peças para o outro lado).

Algoritmo

Para este programa optámos por utilizar o minimax "simples"(sem corte alfabeta), neste caso para cada dificuldade optámos por aumentar a profundidade, para cada dificuldade temos:

- Dificuldade 1: **Profundidade 3**.
- Dificuldade 2: **Profundidade 4**.
- Dificuldade 3: **Profundidade 6**.

Entre estas dificuldades há alguma diferença de tempo, embora que a primeira jogada seja sempre demorada(pois há um número muito grande de possiblidades nas primeiras jogadas).
Decidimos utilizar o algoritmo minimax porque calcula a melhor jogada para ele mesmo tendo em conta um certo número de jogadas à frente.
Nesta parte o computador é sempre o primeiro jogador.

Extensão

Algoritmo

Para a extensão do trabalho utilizámos o algoritmo minimax com corte alfabeta, este vai ser mais rápido(e em princípio melhor que o minimax) porque corta algumas possibilidades não ótimas e não vai continuar a procurar nesse seguimento de jogadas.
Devido a este algoritmo ter uma maior eficiência, decidimos aumentar a dificuldade no 3º nivel, acabámos por implementar o seguinte:

- **Profundidade 6** nas primeiras 14 jogadas.
- **Profundidade 7** nas jogadas 15 a 20.
- **Profundidade 8** nas restantes jogadas.

Devido a problemas temporais, não aumentamos a profundidade para além de 8, pois apesar de ser um algoritmo mais eficiente, uma maior profundidade quebraria as regras de tempo fornecidas no enunciado.
Todas as adições ao programa foram implementadas como pedido.

Instruções e Resultados

Resultados

Dificuldade 1:
Minimax começou, Alfabeta ganhou 20-25.
Alfabeta começou, Alfabeta perdeu 25-20.

Dificuldade 2:
Minimax começou, Empate 24-24.
Alfabeta começou, Empate 24-24.

Dificuldade 3:
Minimax começou, Alfabeta ganhou 16-26.
Alfabeta começou, Alfabeta ganhou 26-16.

Instruções

Para o alfabeta, correr [**ouriAlfaBeta.pl**]**.** , seguido de:

- 1. **ouri.** para começar o pedido de inputs do programa.
- 2. **1/2/3**(selecionar um) para selecionar a dificuldade.
- 3. **1 ou 2** para selecionar o jogador que começa primeiro(1 é o jogador e 2 é o CPU).
- 4. Escolher qualquer um de **1/2/3/4/5/6/0** para jogar na casa correspondente(o 0 faz o que é mencionado no enunciado) até o jogo acabar.

Para o minimax os passos são iguais mas corre-se o **ouriMinMax.pl**. (aqui ignorar o passo 3 pois o CPU começa sempre).

A versão que escolhemos para o torneio é a versão **Extendida**.