Aula 2 - Buscas em Árvores

Docupedia Export

Author:Silio Leonardo (CtP/ETS)
Date:17-Oct-2023 16:40

Table of Contents

1 Tópicos em Teoria dos Jogos

Vamos começar discutindo algo um pouco relacionado a IA que será útil para a disciplina: Teoria dos Jogos. A teoria dos jogos é uma teoria matemática que busca estudar a natureza dos jogos, seus estados e estratégias. Vamos usar IA para implementar vários desses jogos, ou melhor, uma forma de você jogá-los contra máquina. Podemos fazer buscas entre os estados dos jogos e analisar as posições desta forma.

Vamos iniciar resolvendo jogos imparciais. Jogos Imparciais são jogos onde qualquer jogada pode ser feita por qualquer um dos jogadores. Um Jogo Parcial tem peças, como o xadrez, onde somente um dos jogadores pode mover uma peça ou jogo da velha, onde se um jogador pode jogar com 'x' o outro não poderá usar a mesma peça. Assim, em jogos imparciais todas as jogadas podem ser feitas por todos os jogadores sem distinção. Vamos trabalhar também com jogos de informação perfeita. Ou seja, nenhum dos jogadores tem informações adicionais em relação ao outro e todas as informações da partida estão expostas. Isso não acontece quando jogamos um jogo como batalha naval ou um jogo de cartas onde temos informações escondidas de um ou ambos jogadores.
Existem duas convenções para esses jogos. Em geral: Na convenção normal, o primeiro que não puder jogar perder; No Misère, o primeiro jogador que não puder jogar ganha. Então num primeiro momento estaremos tratando de jogos imparciais, de informação perfeita sobre a convenção normal.
Existe um teorema chamada Teorema de Sprague-Grundy que diz que todo jogo imparcial é equivalente a um Nim de 1 pilha. Assim, este jogo, Nim, será inicialmente estudado e faremos uma IA para jogar Nim como se fosse um bom jogador.
Um jogo Nim funciona com algumas pilhas de elementos. A quantidade de pilhas, de elementos por pilhas ou o que será o elemento é indiferente. Podemos usar fósforos e fazer as pilhas com 3, 4 e 5 fósforos respectivamente ou fazer uma única pilha com 20 pedras. No jogo, um jogador deve remover quantos elementos quiser de uma única pilha, mas nunca de duas. Os jogadores se alternam escolhendo uma das pilhas que ainda contendo elementos e retirando de 1 a todos os elementos da pilha. O jogador que fizer a última retirada, isto é, que deixar o oponente se peças a retirar ganha o jogo. Simples assim.
Vamos aprender agora o que poderíamos fazer para implementar uma IA que joga NIm.

2 MiniMax

O MiniMax considerará a jogadas/movimentações por uma árvore onde dois agentes buscam objetivos diferentes. Note que existem jogos de estados reversíveis o que tornaria a árvore um grafo (onde é possível retornar e não só descer na árvore). Mas mesmo assim, usaremos árvores repetindo os estados. Ao invés de buscas simples como fizemos anteriormente, poderemos considerar **árvores infinitas** ou muito grandes onde é impossível gerar todas as possibilidades. Vamos considerar fazer buscas progressivas, ou seja, de profundidade limitada. Além disso, vamos gerar árvores dinamicamente, ou seja, gerando novos estados enquanto fazemos as buscas. É um problema muito mais complexo do que só buscar o menor caminho. O MiniMax será implementado de forma recursiva mas pode ser feita de forma iterativa.
O miniMax dependerá de uma heurística que será uma avaliação do quão bem um jogador estará na posição. Use valores positivos se o jogar está bem, valores negativos se o oponente está bem, zero se a partida está empatada e valores infinitos se um jogador está "ganho". Você pode usar 1 e -1 para posições ganhas também, fazendo uma linearização da avaliação da posição.

```
miniMax(nó, maximizar, profundidade)
    Se profundidade = 0 ou no não tem filhos
        retorna avaliação do nó
    Se maximizar:
        valor = -infinito
        Para cada filho do nó
            valor = máximo de valor e miniMax(filho, false, profundidade - 1)
        retorne valor
    Senão (minimizar):
        valor = +infinito
        Para cada filho do nó
            valor = mínimo de valor e miniMax(filho, true, profundidade - 1)
        retorne valor
```

Podemos implementar o NegaMax, este usa uma identidade para simplificar a lógica do MiniMax mas só funciona para jogos com dois jogadores, ou seja problemas de dois atores:

```
negaMax(nó, profundidade)
    Se profundidade = 0 ou no não tem filhos
        retorna avaliação do nó

    valor = -infinito
    Para cada filho do nó
        valor = máximo de valor e -negaMax(filho, false, profundidade - 1)
    retorne valor
```

3 Alpha-Beta Prunning

O Alpha-Beta Prunning é um MiniMax que faz uma optimização "podando" alguns lances que o algoritmo considera desnecessários para a avaliação da posição. Ela cria um túnel de valore máximos e mínimos em profundidade que ajudam a ignorar vários nós. Ele pode ser impreciso quanto a avaliação de estados, contudo, ele sempre fará uma escolha concisa quanto as jogadas escolhidas.

```
alphabeta(nó, maximizar, profundidade)
    return alphabeta(nó, -infinito, +infinito, maximizar, profundidade)

alphabeta(nó, alfa, beta, maximizar, profundidade)
    Se profundidade = 0 ou no não tem filhos
        retorna avaliação do nó
    Se maximizar:
        valor = -infinito
        Para cada filho do nó
            valor = máximo de valor e alphabeta(filho, alfa, beta, false, profundidade - 1)
            Se valor > beta
                Pare o laço de repetição
            alfa = maior de alfa e valor
        retorne valor
    Senão (minimizar):
        valor = +infinito
        Para cada filho do nó
            valor = mínimo de valor e alphabeta(filho, alfa, beta, true, profundidade - 1)
            Se valor < alfa
                Pare o laço de repetição
            beta = maior de alfa e valor
retorne valor
```