22 Março 2019 – João Paulo Santos Monteiro | Nuno José Moreira Bernardes

CONNECT-FOUR

INTRODUÇÃO

Neste relatório iremos apresentar as conclusões e justificações relativas à simulação de jogos com oponente, passando por caracterizar um problema deste género e mostrar alguns algoritmos usados e os seus fatores de fiabilidade.

Descrição de jogos com oponente e algoritmos para resolvê-los

i. O principal foco do estudo de jogos com oponente situa-se na **incerteza**, pois quando jogamos contra alguém, não nos é possível prever com máxima certeza qual será a jogada do adversário.

ii. Apesar do um o problema de um jogo ser diferente de um problema de busca, é possível definir um jogo como problema de busca, para isto é necessário a existência dos seguintes componentes:

   - estado inicial, isto é, a configuração do tabuleiro e a informação de qual jogador tem a vez de jogar;
   - uma função que retorna os descendentes, uma lista de jogadas a partir da jogada inicial;
   - o teste se é solução;
   - função *utility* que retorna quem ganhou o jogo;
   - limite de profundidade;

iii. Se nos problemas de busca do trabalho anterior existia um algoritmo Guloso, para este tipo de problema também existe esse tipo de algoritmo, porém o seu método não leva em consideração as jogadas do oponente levando-o a escolher o único caminho que lhe interessa a ele, ignorando o adversário.

iv. Não podendo usar os métodos usados no trabalho anterior, por serem pouco eficazes neste contexto, foram-nos apresentados outros algoritmos de forma a resolver o problema do oponente, foram estes:

   - Minimax;
   - Alpha-Beta;
   - Monte Carlo three search (MCTS);

ALGORITMOS

Minimax:

v. O Minimax é um algoritmo que verifica todas as possibilidades, ou seja, ele vai jogando de acordo com a sua jogada e a do adversário, assumindo sempre que o seu oponente joga de forma ótima. Este método expande todas as jogadas possíveis até as folhas da árvore onde atribui um valor *utility indicando* vitória, empate ou derrota. Após isso, o algoritmo sobe um nó e identifica quem foi o último a jogar. Vai repetindo este processo repetidamente identificando qual o melhor caminho para a vitoria. Este algoritmo tem complexidade temporal de $O(b^m)$ e tem complexidade espacial O(bm), onde m é a profundidade máxima a que chegou a árvore e b é número de jogadas até esse ponto.

Alpha-Beta

vi. O corte Alpha-Beta é um algoritmo que consegue "cortar" para metade a árvore de jogo criada no algoritmo Minimax. Este método funciona a partir do Minimax, eliminando da árvore os nós que não têm qualquer interferência na decisão final. Tem geralmente complexidade temporal de $O(b^{m/2})$ podendo chegar a $O(b^m)$ no pior caso e complexidade espacial de O(bm), onde m é a profundidade máxima a que chegou a árvore e b o número de jogadas até esse ponto.

Monte Carlo tree search (MCTS)

vii. A base do MCTS está na análise dos movimentos mais promissores, expandindo a árvore de busca com base numa amostra aleatória do espaço de busca. A cada jogada a árvore é expandida até ao final com movimentos aleatórios, o resultado é depois usado para escolher qual os nós com melhores chances de serem escolhidos em jogadas futuras. Cada jogada consiste em quatro etapas:

- **Seleção:** a partir da raiz seleciona nós descendentes sucessivos até que uma folha seja alcançada;
- **Expansão:** se a folha não for um estado terminal (ganhar, empate ou derrota) cria um nó descendente;
- **Simulação:** escolhe movimentos aleatórios até um estado terminal;
- **Retro propagação:** usa o resultado para atualizar o resultado heurístico dos nós anteriores.

CONNECT-FOUR

viii. O 4 em linha (Connect-Four) é um jogo em que cada participante começa com 21 peças de cores diferentes e tem de as colocar num tabuleiro com 6 linhas e 7 colunas. Um movimento neste jogo consiste em soltar uma peça numa coluna a escolha atingindo assim a mesma a parte inferior da coluna. Os participantes jogam alternadamente e tem como objetivo conectar 4 peças, independentemente da direção (horizontal, vertical e diagonal).

MINIMAX, ALPHABETA E MCTS APLICADO AO CONNECT-FOUR

ix. Para nos ser possível o uso dos algoritmos mencionados anteriormente, usamos uma implementação bastante simples para manipulação de objetos (neste caso, tabuleiros de 4 em linha). Usamos mais uma vez a linguagem Java, pelas mesmas razões que citamos no trabalho anterior (capacidade de criação e manipulação de objetos muito avançada). Para os algoritmos Minimax e Alpha-Beta usamos uma matriz para guardar cada peça no tabuleiro e manipulamos todos os valores em torno dela. Usamos funções variadas tais como *player* (que retorna qual jogador vai jogar num determinado tabuleiro), *actions* (que retorna uma lista de ações possíveis de ser executadas num determinado tabuleiro), *result* (que retorna o tabuleiro que resulta duma ação aplicada), *terminalTest* (que retorna se um tabuleiro conseguiu chegar a um 4 em linha) e a função *utility* (que trata de retornar o valor de heurística para cada jogador). Para o algoritmo Monte Carlo usamos dois tipos de objeto, Nó e Árvore (*Node* e *Tree*).

x. Neste algoritmo seguimos o pseudo-código facultado pela professora e criamos funções úteis para que o mesmo funcionasse corretamente tais como *isLeaf* (verifica se um nó é folha na árvore), *getUCB* (verifica qual dos nós num determinado nível da árvore tem UCB mais elevado), *getNValue* (retorna a quantidade de vezes que a raiz da árvore foi visitada, para cálculo do UCB), *getChild* (retorna a lista de nós filhos de um determinado nó), *getniValue* (retorna o número de vezes que o nó atual foi visitado), *getActions* (retorna as ações que são possíveis de realizar num determinado nó), *getfstNewChild* (retorna o primeiro filho não visitado de um determinado nó), *getBest* (retorna a ação que tem melhor valor de vitória).

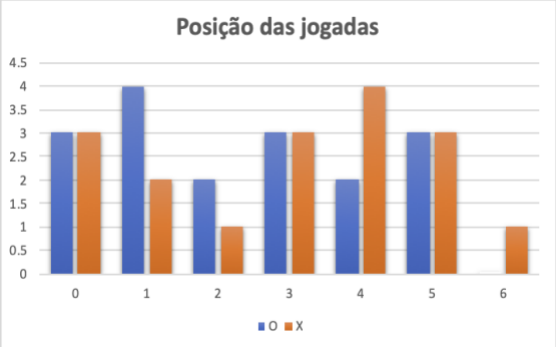
xi. Para o cálculo da heurística e valor para o qual a nó de uma árvore é explorado usamos o valor UCB:

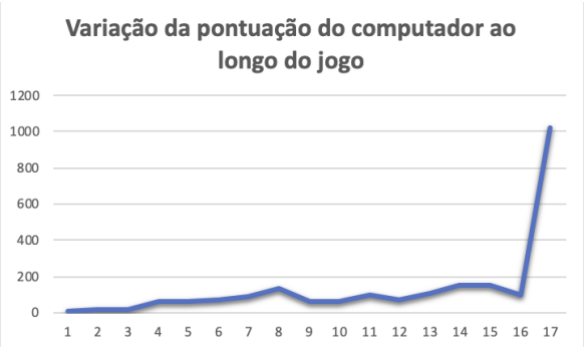
$$UCB = \frac{W_i}{V_i} + C\sqrt{\frac{\ln N}{n_i}}, C = 2$$

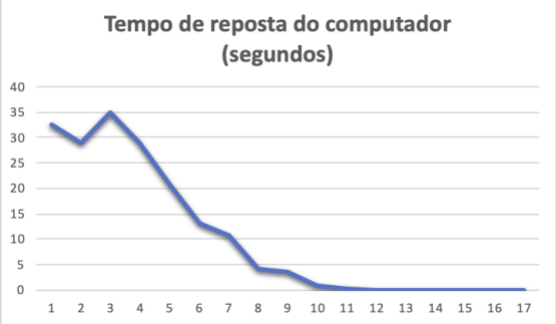
W_i = Win Score, V_i = Visit, N = Visit Root, n_i = Visit current node

CONCLUSÕES E COMENTÁRIOS FINAIS

xii. Para retirarmos conclusões, fizemos um jogo intensivo com cada um dos algoritmos para obter gráficos que nos auxiliem nas conclusões. Para o algoritmo *Minimax* obtemos o seguinte:

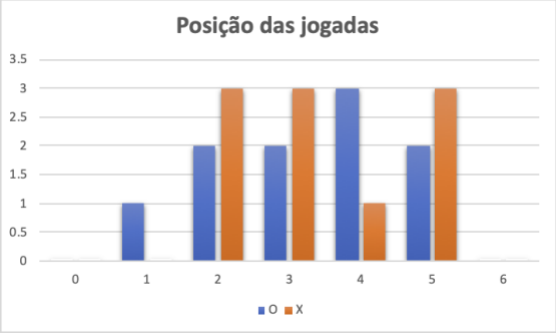






Para este algoritmo conseguimos concluir que à medida que as jogadas são feitas o algoritmo é cada vez mais rápido, que a sua pontuação sofre poucas alterações independentemente da jogada do oponente e que maioria das jogadas que o algoritmo faz são nas colunas onde o oponente jogou. Para o teste foi usada profundidade 8 no algoritmo.

xiii. Para o algoritmo alfa-beta:



É um pouco mais inteligente nas escolhas das jogadas, o tempo de resposta para a ação que vai executar é quase reduzido a metade comparado com o *Minimax* e ganha o jogo em menos jogadas. Este algoritmo tem estas características por causa das duas variáveis alfa e beta que vão manter os melhores valores de escolha de jogada, evitando pesquisar caminhos repetidos e tornando-se mais eficiente.

Entre todos os algoritmos que usamos, o Monte Carlo mostrou-se ser o algoritmo mais eficiente, que demonstrava as jogas mais coordenadas, mesmo assim falhando variadas vezes em jogadas óbvias.

BIBLIOGRAFIA

O nosso relatório foi baseado no livro recomendado pela professora no início da unidade curricular:

- *Artificial Intelligence: A Modern Approach* (3ª Edição) *Stuart Russel & Peter Norvig;*

Todas as conclusões, estruturas de dados e algoritmos usados foram confirmados neste livro, pois têm justificações muito completas que tornam fácil o entendimento deste trabalho.