UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DE COMPUTAÇÃO E MATEMÁTICA COMPUTACIONAL

JEAN MARCOS SILVA SANTOS

BUSCA COM ADVERSÁRIOS

Usando o método de busca Minimax para o jogo Othello

São Carlos, SP

2020

Resumo

O objetivo desde trabalho é apresentar os resultados da aplicação do algoritmo minimax para o jogo Othello no contexto da busca com adversários em Inteligência Artificial. Para isso foi realizada uma implementação do algoritmo usando técnicas de poda alfa-beta e função de avaliação heurística, e foi realizada a implementação do ambiente e do controle para permitir os testes. Verificamos nos resultados que, mesmo usando as técnicas mencionadas, o crescimento exponencial de tempo e uso de memória ocorreu muito rápido, de maneira que usando uma profundidade máxima de 10 para a árvore do jogo, o tempo de processamento do algoritmo para o segundo movimento do jogo já foi de 119s, já se tornando inviável na prática. Concluímos então que mesmo para problemas já solucionados, como a busca com adversários para o jogo Othello, onde os algoritmos já superaram os seres humanos, essa área contínua sendo de extremo interesse da IA, onde melhorias de algoritmos, heurísticas e técnicas podem sempre surgir.

Palavras-chave: inteligência artificial, jogos, minimax, othello.

Sumário

1. INTRODUÇÃO

Jogos são uma área especialmente interessante para a Inteligência Artificial. Isso porque os jogos, em geral, trazem uma série de elementos que os tornam atrativos para os pesquisadores, como uma fácil representatividade, pequenos números de ações que são regidas por regras precisas [7] e facilidade na determinação do vencedor.

Nas abordagens para jogos dentro da IA, examinamos um ambiente multiagente competitivo, em que os objetivos dos agentes estão em conflito. Esse comportamento é empiricamente observável, por exemplo, quando jogamos xadrez contra algum programa de computador, onde o objetivo de ambos é vencer a partida, porém como existe apenas um vencedor, os objetivos estão em conflito. Para resolver esse problema, existe dentro da área de busca da IA, a busca competitiva ou busca com adversários.

Em IA, os jogos mais comuns e que possuem as características mais atrativas, são os jogos que chamamos de soma zero. Jogos de soma zero ocorrem em ambientes determinísticos totalmente observáveis em que dois agentes agem alternadamente e nos quais os valores de utilidade no fim do jogo são sempre iguais e opostos [7]. Por exemplo, em um jogo de damas, se um jogador ganhar, o outro necessariamente perde. Outra característica importante a ser observada nos jogos soma zero, é que não existe sorte envolvida, como é o caso de um arremesso de dados para determinar a aplicação de uma regra.

Apesar de serem considerados menos complexos do ponto de vista da resolução por IA, os jogos de soma zero não deixam de ser desafiadores, principalmente devido a limitações do tamanho do espaço de busca e de tempo. Dentro desse contexto, o objetivo desse trabalho é apresentar o uso do método de busca minimax no jogo Othello.

2. FUNDAMENTAÇÃO TEÓRICA

São apresentados neste capítulo os conceitos de Inteligência Artificial que foram utilizados para o desenvolvimento deste trabalho. A apresentação inicia falando sobre a área da IA que trata da resolução de problemas de jogos, avança definindo e detalhando o método de busca utilizado no trabalho e finaliza explicando como funciona o jogo Othello, que é o jogo alvo do estudo de caso.

2.1 Busca com Adversários

Os problemas e algoritmos que trabalham com oponentes torna o processo mais complicado e interessante se compararmos com os primeiros problemas em que a IA se introduz. O adversário introduz incertezas e suas decisões são realizadas em uma igual tentativa de ganhar o jogo. Na busca com adversários examinamos problemas que surgem quando tentamos planejar com antecedência em um mundo no qual outros agentes estão fazendo planos contra nós [7].

Os jogos também são extremamente interessantes porque podem ser muito difíceis de resolver. Vejamos o caso de um jogo de xadrez que é um jogo de zero soma. O xadrez tem um fator médio de ramificação de 35, e podem chegar a 50 movimentos por jogador. De acordo com a estimativa feita por Claude Shannon, conhecida como Número de Shannon, o tamanho da árvore total de busca é de aproximadamente 10^{120} nós, esse número é maior que a estimativa da quantidade de átomos do universo. Uma máquina realizando uma operação por microssegundo levaria 10^{90} anos para calcular o primeiro movimento [9].

Para os jogos de zero soma, os quais este trabalho foca, Russell e Norvig [7] definem um jogo como um problema de busca com os seguintes componentes:

- Estado Inicial: determina a posição das peças no tabuleiro
- Jogadores: define quem são os jogadores e quem fará o primeiro movimento
- Ações: o conjunto de movimentos válidos em um estado
- Resultado: o modelo de transição que define o resultado de um movimento
- Teste de Término: verifica o fim do jogo. Os estados em que um jogo termina são chamados de estados terminais
- Função de utilidade: também chamada de função objetivo, define o valor numérico para um jogo que termina no estado terminal *s* por um jogador *p*.

O estado inicial, a função ações e a função resultado definem a árvore do jogo, em que as folhas são os estados terminais e a raiz é o estado inicial. Chamaremos os jogadores de MIN e MAX. A figura 2.1 mostra um pedaço da árvore de jogo do jogo da velha. A partir da raiz, ou do estado inicial, MAX tem nove movimentos possíveis. Alternadamente um X é colocado por MAX e um O é colocado por MIN até que cheguemos as folhas da árvore que representam os estados terminais, os quais são um jogador ter três símbolos em uma linha ou não houver mais quadrados disponíveis. Em cada nó folha, o número representa o valor utilidade daquele estado terminal para o jogador atual. Valores altos são bons para MAX e valores baixos são bons para MIN (daí o nome dos jogadores e do algoritmo que veremos a seguir).

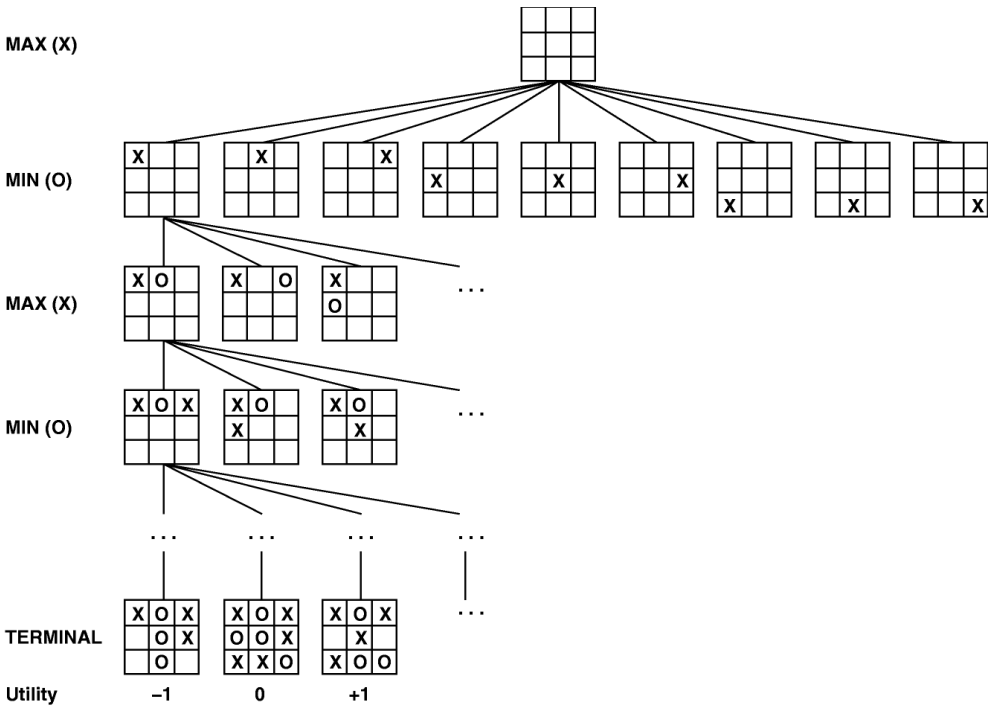

Figura 2.1 – Uma árvore de busca parcial para um jogo da velha

Para que uma estratégia seja ótima, ela deve especificar o movimento inicial de MAX, e seus movimentos seguintes respondendo a cada ação de MIN. O valor minimax em cada nó, é a utilidade de se encontrar uma sequência de estados que leve ao melhor estado final. Através do valor minimax é possível definir uma estratégia ótima [6].

2.2 Algoritmo Minimax

O algoritmo minimax foi apresentado em 1944 por John von Neumann e Oskar Morgenstern na obra Theory of Games and Economic Behavior (Teoria dos Jogos e Comportamento Econômico) [1]. Ele utiliza computação recursiva para tomar a decisão minimax do estado atual. A recursão percorre desde o caminho atual até as folhas, e então retorna propagando os valores minimax. O objetivo da estratégia é, para cada nível da árvore de jogo, tentar maximizar a pontuação do jogo, caso seja o MAX, e minimizar a pontuação caso seja o MIN. Observe a figura 2.2.

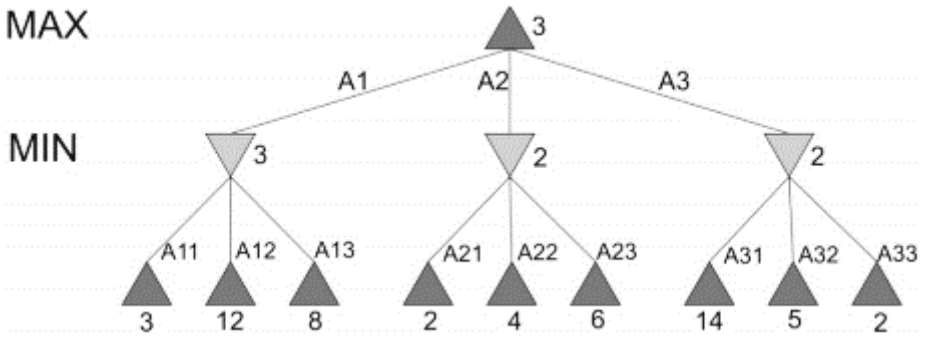

Figura 2.2 – Uma árvore de jogo com apenas duas jogadas [3]. Primeiro, é feita a criação de toda a árvore de jogo, com todas as jogadas e as respectivas respostas de cada jogador. Então, é aplicada a função de utilidade aos nós folhas e esses valores são propagados para o nó pai de acordo com o interesse do MIN, ou seja, escolher sempre o menor valor.

Podemos analisar na figura que, mesmo que o jogador MAX queira obter a melhor pontuação possível, que iniciaria através da aplicação da ação A3, ele não poderá continuar por esse caminho, pois o jogador MIN poderá seguir como resposta o caminho A33. Dessa maneira, após serem propagados os valores dos nós folhas até o nó corrente, o jogador MAX irá escolher o maior valor entre eles.

Veja na figura abaixo o algoritmo minimax.

```
begin miniMax (no_corrente)
    if ehFolha(no_corrente) then
        return pontuacao(no_corrente);
    end
    if ehNoMin(no_corrente) then
        return min(miniMax(filhosDe(no_corrente)));
    end
    if ehNoMax(no_corrente) then
        return max(miniMax(filhosDe(no_corrente)));
    end
end
```

Figura 2.3 – Algoritmo Minimax.

A complexidade de tempo do algoritmo minimax é $O(b^m)$, onde m é a profundidade máxima da árvore, e b a quantidade de movimentos válidos em cada ponto. O tempo exponencial é totalmente impraticável em jogos reais, mas podemos usar o algoritmo para servir como base para análises e algoritmos mais práticos.

2.3 Jogo Othello

Othello ou Reversi é um jogo de tabuleiro de soma zero com informação perfeita. Jogos com informação perfeita significam ambientes determinísticos completamente observáveis [7], ou seja, cada jogador visualizar todo o estado do jogo a qualquer momento, diferente de jogos de cartas em os jogadores escondem suas cartas.

O jogo é disputado em um tabuleiro 8x8 com 64 discos que possuem uma cor diferente de cada lado. Ganha o jogo quem ficar com mais peças da sua cor ao fim. Veja abaixo a configuração inicial de um tabuleiro.

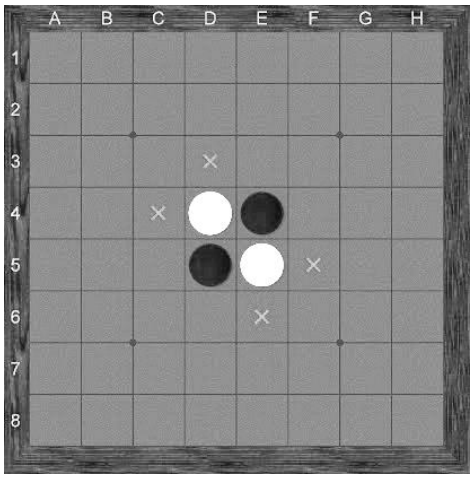

Figura 2.4 – Estado inicial de uma partida de Othello. Os campos assinalados com X marcam as primeiras jogadas válidas para o jogador com as peças pretas.

O jogo se inicia com 4 peças no centro do tabuleiro, 2 brancas e 2 pretas. O jogador com as peças pretas inicia a partida colocando uma nova peça preta ao lado de uma peça branca. Todas as peças do oponente que estejam em uma linha reta (horizontal, vertical ou diagonal) entre a peça recém-colocada e qualquer outra peça do jogador que fez o movimento são viradas e passam a ser da cor deste jogador.

A partida termina quando nenhum jogador puder fazer mais movimentos. Isso acontece quando não existe mais espaço no tabuleiro ou se todas as peças do tabuleiro estiverem com a mesma cor virada para cima. Se no fim do jogo, ambos tiverem a mesma quantidade de peças, o jogo termina em empate.

3. ESTUDO DE CASO

Esse capítulo de dedica a apresentar como foi feita a implementação do algoritmo minimax para o jogo Othello. A implementação completa pode ser encontrada em https://github.com/jhemarcos/minimax-othello. O jogo também está disponível na WEB para ser jogado, bastando acessar o link https://jhemarcos.github.io/minimax-othello/.

3.1 Ambiente

O foi escrito na linguagem javascript e o ambiente de atuação dos agentes, isto é, o tabuleiro e as iterações foram desenvolvidos usando as tecnologias padrões da WEB, sendo html, css e javascript. O ambiente pode ser visto na figura 3.1.

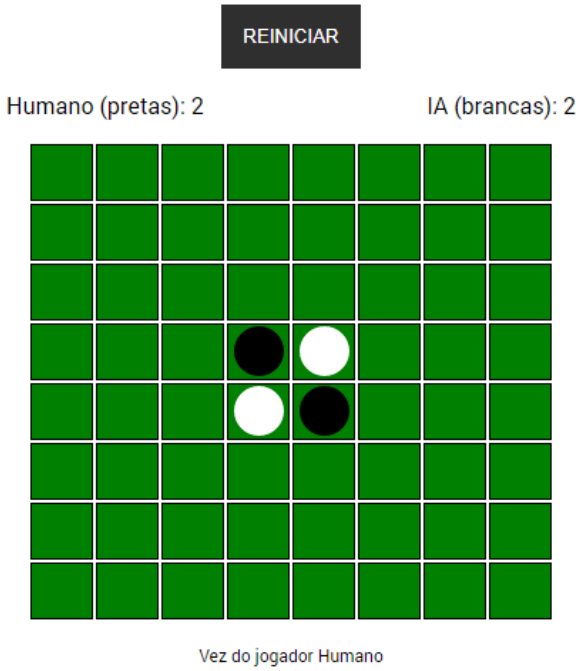

Figura 3.1 – Interface do jogo Othello desenvolvida

O ambiente segue todas as regras do jogo, sendo assim os agentes, humano ou IA, só podem realizar jogadas que obedeçam às regras. Quando o agente realiza uma jogada, o controle verifica quais são as peças do adversário precisam ser transformadas em peças do agente atual. Após fazer as devidas atualizações o controle passa a vez para o próximo agente.

O controle também verifica após cada movimento se algum estado terminal foi atingido. Para o Othelo, estados terminais são aqueles estados onde nenhum dos agentes tem mais jogadas válidas disponíveis. Caso seja a vez de um agente, e ele não tenha mais jogadas válidas disponíveis, mas o outro agente tenha, o controle passa a vez para o próximo agente conforme define a regra do jogo.

3.2 Agentes

A implementação desenvolvida define um agente humano e um agente de IA. O agente humano pode realizar seus movimentos usando o clique do mouse no quadrado desejado quando o ambiente indicar que é a sua vez. Caso o agente humano tente selecionar uma jogada inválida, o controle irá ignorar essa jogada e continuar aguardando uma jogada válida. O agente humano é representado pelas peças pretas e sempre começa o jogo.

O agente de IA é recebe sua vez 3 segundos após o agente humano realizar sua jogada, e assim o jogo segue alternando até um estado terminal. Esse tempo de 3 segundos é adicionado pelo controle para que a animação da troca de peças possa ser acompanhada pelo agente humano. O agente de IA chama o algoritmo minimax para fazer a seleção da jogada.

O trecho de código abaixo mostra como é feita a criação dos agentes, aqui representados pela classe/função Player. Cada agente instanciado recebe um nome, um identificador e um boleano indicando se esse agente deve ser um agente de IA.

```
var players = [
   new Player("Humano", 0, false),
```

```
    new Player("IA", 1, true)
];
```

3.3 Minimax, Poda Alfa - Beta e Heurística

Apesar de que o jogo Othello não apresenta um fator de ramificação tão grande, o fato do algoritmo minimax tem complexidade em $O(b^m)$ torna impraticável analisar toda a árvore do jogo para cada decisão de jogada. Sendo assim, a implementação do algoritmo seguiu duas estratégias de otimização, a poda alfa-beta e a substituição da função de utilidade por uma função de avaliação que usa uma heurística quando chega a determinada profundidade.

O algoritmo minimax funciona de maneira recursiva e, para cada jogada válida, realiza a busca pelo movimento de maior pontuação. O algoritmo gera a árvore do jogo até a profundidade máxima definida, e quando gera os nós dessa profundidade, usa a função de avaliação heurística para determinar a pontuação do nó. Então a recursividade retrocede propagando a pontuação para os nós antecessores. Nesse processo de retrocesso, a poda alfa-beta remove subárvores que não precisam ser geradas devido a pontuação já obtida por um nó estado de um mesmo ancestral. Esse processo de geração segue por meio de busca em profundidade até que a recursão volte ao nó atual trazendo consigo o melhor movimento de acordo com a pontuação. O algoritmo minimax utilizado pode ser encontrado no Apêndice A.

A função de avaliação usa uma heurística já conhecida para o jogo Othello, chamada heurística de mobilidade [8]. Essa heurística representa a quantidade de movimentos válidos disponíveis e parte do princípio de que é interessante minimizar a quantidade de jogadas do seu oponente, enquanto maximiza as suas possibilidades. O algoritmo da heurística implementada pode ser encontrado no Apêndice B.

4. RESULTADOS

O sucesso da melhor escolha pelo agente de IA, está diretamente ligado a heurística utilizada na função de avaliação e na profundidade máxima da árvore de jogo.

Porém quanto maior a profundidade, mais tempo o agente precisará para realizar seu movimento. Como os movimentos devem ser realizados em um período razoável, é necessário achar um balanço para a profundidade.

O gráfico 4.1 exibe a profundidade, o tempo gasto e a quantidade de nós gerados na árvore de jogo para o segundo movimento do jogo, para algumas profundidades. Repare que o tempo e a quantidade de nós gerados crescem exponencialmente.

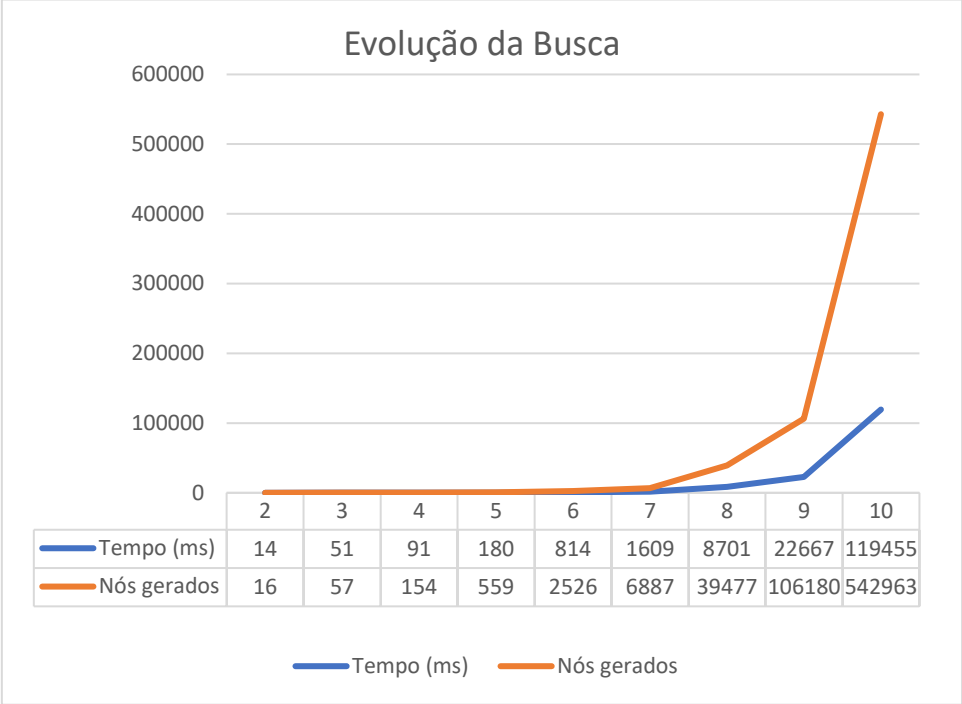

Gráfico 4.1 – Evolução do algoritmo minimax na árvore de jogo.

Todas as simulações foram realizadas no mesmo computador. O projeto foi executado no navegador Chrome Versão 81.0.4044.113, que usa o V8 como interpretador javascript. As configurações do computador usado são: Windows 10 Pro 64 bits, Intel(R) Core(TM) i5-4210U @ 1.7GHz 2.4GHz, 8GB RAM.

5. CONCLUSÃO

Jogos continuam sendo uma área extremamente interessante para a IA. A complexidade dos jogos cresce cada vez mais e junto com ela a necessidade de algoritmos e técnicas de IA que consigam fornecer boas soluções.

Apesar do jogo Othello já ser alvo de pesquisas de IA a muitas décadas, e de que os algoritmos para ele já tenham superado a inteligência humana [6], ele ainda continua sendo um ótimo estudo de caso para irmos mais fundo nas técnicas de busca competitiva em IA.

O algoritmo minimax implementado tem a vantagem de trazer a solução ótima para um espaço de estados finito, mas em geral, seu tempo de execução que cresce exponencialmente torna inviável tal aplicação. Usando as técnicas de poda alfa-beta e a função de avaliação heurística com profundidade máxima conseguimos obter resultados razoáveis para nosso agente inteligente que podem sim desafiar um agente humano.

Como continuação para esse trabalho, podemos considerar a adição de novas heurísticas para comparação de seus resultados, criando assim uma competição de algoritmos.

Referências Bibliográficas

[1] ALMEIDA, Fábio Portela Lopes de. A teoria dos jogos: uma fundamentação teórica dos métodos de resolução de disputa. 2003. Disponível em: <http://www.arcos.org.br/livros/estudos-de-arbitragem-mediacao-e-negociacao-vol2/terceira-parte-artigo-dos-pesquisadores/a-teoria-dos-jogos-uma-fundamentacao-teorica-dos-metodos-de-resolucao-de-disputa>. Acesso em: 19 de abril de 2020.

[2] Gingras, Martin. Javascript Implementation of Othello vs AI player. Disponível em: < https://github.com/mgingras/othello>. Acesso em: 15 de abril de 2020.

[3] MACHADO, Vinicius Ponte. Inteligência Artifical. UECE. Disponível em: <http://www.uece.br/computacaoead/index.php/downloads/doc_download/2177-inteligencia-artificial/>. Acesso em: 16 de abril de 2020.

[4] OTHELLO CLASSIC: Regras do Jogo. Disponível em: <http://othelloclassic.blogspot.com/2010/06/regras-do-jogo.html/>. Acesso em: 16 de abril de 2020.

[5] Othello (Reversi) - Como jogar. Disponível em: <http://www.othellobrasil.com.br/regras.php/>. Acesso em: 16 de abril de 2020.

[6] QUAGLIO, Vinícius Gomes. Técnicas de Inteligência Artificial Aplicadas ao Jogo Othello: Um Estudo Comparativo. 2013. 48f. UEL. 2013.

[7] RUSSELL, S.; NORVIG, P. Artificial Intelligence: A Modern Approach. 3 ed. Prentice Hall, 2010.

[8] Sannidhanam, Vaishnavi; Annamalai, Muthukaruppan. An Analysis of Heuristics in Othello. University of Washington. 2004.

[9] SHANNON, C. E. XXII. Programming a Computer for Playing Chess. Philosophical Magazine, 1950.

Apêndice A – Algoritmo Minimax Implementado

```
IA.prototype.minimax = function(board, depth, currentPlayer,
maxDepth, alpha, beta) {
    this.visits++;
    var newBoard, score, move;
    var bestMove;
    var moves = board.getAllValidMoves(currentPlayer);

    if(depth >= maxDepth || moves.length === 0){
        var he = this.mobility(board, currentPlayer);
        return he;
    }
    if(currentPlayer === this.currentPlayer){
        // Maximize
        for (var i = moves.length - 1; i >= 0; i--) {
            move = moves[i];
            newBoard = board.copy();
            this.doMove(newBoard, move, currentPlayer);
            score = this.minimax(newBoard, (depth + 1),
(currentPlayer ? 0 : 1), maxDepth, alpha, beta);
            move.score = score;
            if(score > alpha){
                alpha = score;
                bestMove = move;

            }
            if(beta <= alpha){
                break;
            }
        }
        if(depth === 0){
            return bestMove;
        } else {
            return alpha;
        }
    } else {
        // Minimize
        var min = 100000;
        for (var i = moves.length - 1; i >= 0; i--) {
            move = moves[i];
            newBoard = board.copy();
            this.doMove(newBoard, move, currentPlayer);
            score = this.minimax(newBoard, (depth + 1),
(currentPlayer ? 0 : 1), maxDepth, alpha, beta);
            if(score < beta){
                beta = score;
            }
            if(beta <= alpha){
                break;
            }
        }
        return beta;
    }
}
```

Apêndice B – Heurística de Mobilidade Implementada

```
IA.prototype.mobility = function(board, currentPlayer) {

    var aiMoves =
board.getAllValidMoves(currentPlayer).length;
    var oppMoves = board.getAllValidMoves(currentPlayer ?
0 : 1).length;

    return Math.ceil((oppMoves + aiMoves) === 0 ? 0 : 100
* ((aiMoves - oppMoves)/(aiMoves + oppMoves)));
}
```