Implementação dos agentes Felipe Franco e Bambam para o jogo de tabuleiro Othello

Igor Carpanese Thais Luca Tiago Montalvão

Resumo

Concluímos nossa participação no curso "Inteligência Artificial" com um trabalho que muito nos agradou. Neste relatório, descreveremos não apenas a lógica por trás dos agentes implementados, mas todo o processo de aprendizado. Esperamos refletir a satisfação que foi brincar com as diferentes estratégias usadas e participar deste projeto como um todo.

Sumário

1 Introdução **3**

1.1 Descrição do jogo 3

1.2 Estratégia inicial 3

2 Agentes **4**

2.1 Felipe Franco 4

2.2 Bambam 5

2.2.1 Minimax 5

2.2.2 Tabela de transposição 6

2.2.3 Posições estáveis 7

2.2.4 Peças de fronteira 8

2.2.5 Outras observações 8

3 Conclusões **9**

1 Introdução

Nunca havíamos jogado Othello. De fato, éramos tão desconhecidos um do outro que nem sabíamos que podíamos chamar Reversi por este nome. Nossa primeira dificuldade foi, portanto, aprender a jogar. Só depois pensamos em como jogar de maneira mais inteligente.

1.1 Descrição do jogo

Reversi, ou Othello como ficou conhecido, é jogado por dois participantes – um assume a cor preta e o outro, a branca – em um tabuleiro quadrado de dimensões 8×8. Cada jogador começa com 2 peças postas em diagonal (ver figura 1) e tem como objetivo, ao final da partida, ocupar o tabuleiro com o maior número possível de peças a seu favor.

Para que uma jogada seja realizada, é preciso que haja pelo menos uma linha reta, seja na horizontal, vertical ou diagonal, sem espaços vazios entre a posição disponível e uma peça da cor escolhida pelo jogador, além de ter uma ou mais peças do oponente entre elas (ver figura 1). Assim que o jogador realiza uma jogada válida, todas as peças do oponente nesta linha reta entre a peça recém-colocada e a outra peça do jogador são viradas e passam a ser da cor das peças do jogador.

O jogador com as peças pretas sempre dá início a partida e o jogo só termina quando não há mais jogadas disponíveis no tabuleiro. Caso ambos tenham a mesma quantidade de peças, o jogo termina em empate.

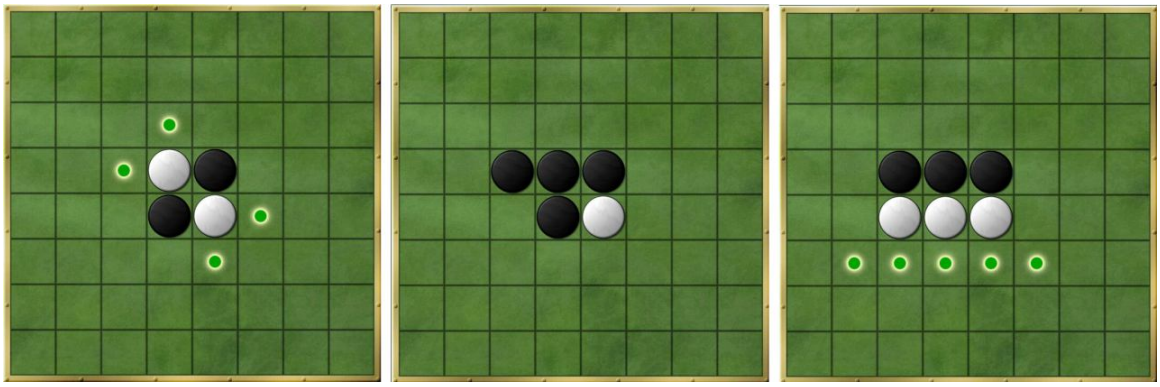

Figura 1: Exemplos de jogadas. A configuração inicial é a do tabuleiro mais à esquerda. Imagens obtida através do aplicativo Reversi Free, para Android.

1.2 Estratégia inicial

Após termos aprendido as regras do jogo e realizado algumas partidas, percebemos quase que imediatamente que as posições mais importantes são os cantos (corners) do tabuleiro, já que uma peça no canto nunca será virada (voltaremos a essa discussão em breve).

Procurando algumas estratégias na Internet [1] , encontramos a definição de **quadrado de segurança**, uma área em que podemos jogar sem darmos oportunidade para o oponente conquistar os cantos.

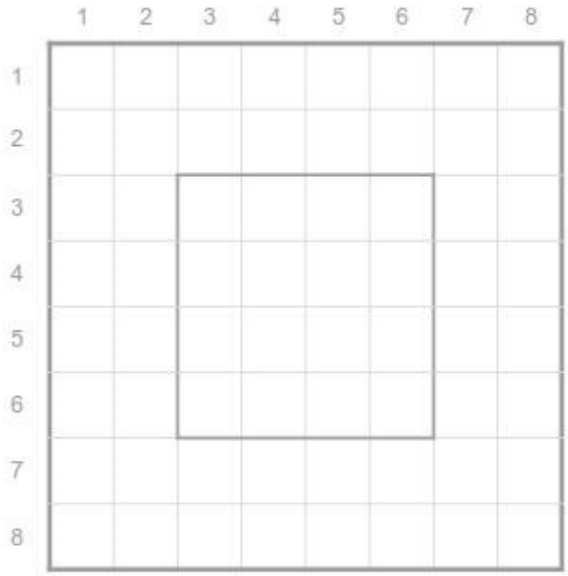

Figura 2: Posições contidas no quadrado de segurança.

Se sempre conseguirmos jogar no quadrado de segurança, iremos fazer com que o adversário, em algum momento, jogue fora dele. Quando isso acontecer, teremos a chance de conquistar uma posição da borda do tabuleiro. A borda é composta pelas linhas 1 e 8 e pelas colunas 1 e 8. Elas são, nesta estratégia inicial, nosso caminho para os cantos.

Mesmo com essa lógica bem simples, conseguimos ganhar algumas poucas partidas no aplicativo mencionado acima. Ficamos satisfeitos com o nosso resultado e partimos para a implementação dos agentes.

2 Agentes

2.1 Felipe Franco

Nosso objetivo era desenvolver agentes cronologicamente. Permita-nos explicar melhor. Queríamos começar de algo muito simples e ir "turbinando" a inteligência do jogador artificial a cada nova versão. Nosso primeiro agente feito, e o primeiro a ser apresentado, chama-se **Felipe Franco**.

Sua estratégia é, a cada rodada, escolher a posição que irá virar o maior número de peças adversárias. Como dito, é uma estratégia muito simples e serve, principalmente, para verificar o desempenho dos agentes futuros em partidas contra uma inteligência artificial

não muito desenvolvida.

Um comentário muito interessante a ser feito (e que por nós não foi observado de imediato) é o fato de que a abordagem gulosa nada mais é do que um algoritmo Minimax com altura igual a um.

2.2 Bambam

Nosso segundo agente, escolhido para participar do torneio, recebe o nome **Bambam**. Ao contrário de Felipe Franco, este não joga somente tentando virar o maior número de peças possível. Bambam leva em consideração uma quantidade bem maior de estratégias e observações, as quais descreveremos a seguir.

Em destaque, informamos de antemão que tanto o corte α–β quanto o Minimax foram implementados na função `alphabeta`. Daqui a pouco falaremos sobre eles.

2.2.1 Minimax

A primeira (e mais óbvia) melhoria a ser feita no Felipe, foi adicionar pesos condizentes à estratégia do quadrado de segurança (ver figura 3), sempre testando o resultado com agentes não-humanos de diferentes dificuldades.

Cada posição possui um certo valor que pode ser positivo ou negativo. Quanto mais próximo de $+\infty$ mais interessante é a posição para o agente. Quanto mais próximo de $-\infty$ mais interessante é a posição para o adversário. O valor da jogada é o valor da soma de todas as posições que contém uma peça do agente.

Como nosso objetivo inicial era conquistar os cantos, nossa primeira ideia foi atribuir um peso $+\infty$ a eles. Funcionou, de certa forma. O Bambam sempre pegava todos os quatro corners. Isso significava para ele, entretanto, que adquirir os cantos era o objetivo do jogo. Péssima modelagem. Redistribuímos os pesos de outra maneira que funcionou melhor (ver figura 3).

Implementamos um caso particular do Minimax, o **Negamax**. Nesta variante, o valor da jogada do seu adversário é o oposto do valor da sua jogada. Em outras palavras, quando o jogador maximiza sua jogada, ele também estará minimizando a jogada do adversário fazendo com que a computação sirva para ambos os jogadores (ver figura 4).

Uma modificação obrigatória foi o corte α–β. Medimos o tempo economizado após sua implementação e descobrimos que, ao longo da partida, o Bambam foi, em média, 6 segundos mais rápido.

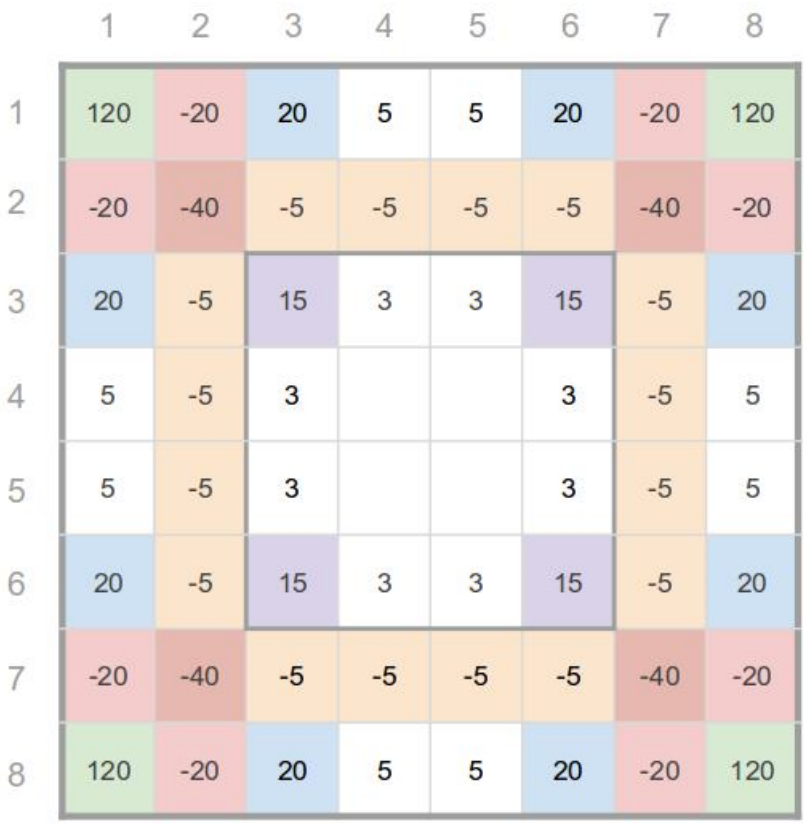

Figura 3: Tabela usada para avaliação de uma jogada com algoritmo Minimax.

Por padrão, o Minimax tem uma profundidade de 5 níveis, o que resulta em jogadas que demoram, em média, menos de 5 segundos. Analisamos uns valores especiais e percebemos que, quando o agente tem poucas jogadas (até 5 opções) para escolher, era possível aumentar a profundidade da busca (para 7 níveis) sem impactar o desempenho de tempo.

Uma outra modificação feita foi de deixar o Minimax buscar a árvore inteira até o final do jogo quando haviam poucas casas sem peças (menos de 12 espaços vazios no código).

Uma última modificação, mais simples e com resultados menos impactantes, foi feita para o caso de quando só há uma jogada disponível. Nessa situação, não faz sentido usar algoritmo algum. Basta fazer tal jogada.

2.2.2 Tabela de transposição

Uma ideia vista em aula, porém não muito explorada, envolve o conceito de **tabelas de transposição**. Imagine que tenhamos gasto um tempo considerável avaliando a qualidade de uma jogada j. Suponha agora que j seja uma jogada recorrente e, portanto, frequentemente precisaremos computar sua qualidade. A tabela de transposição (comumente implementada como uma tabela hash) existe justamente para guardamos os valores das jogadas

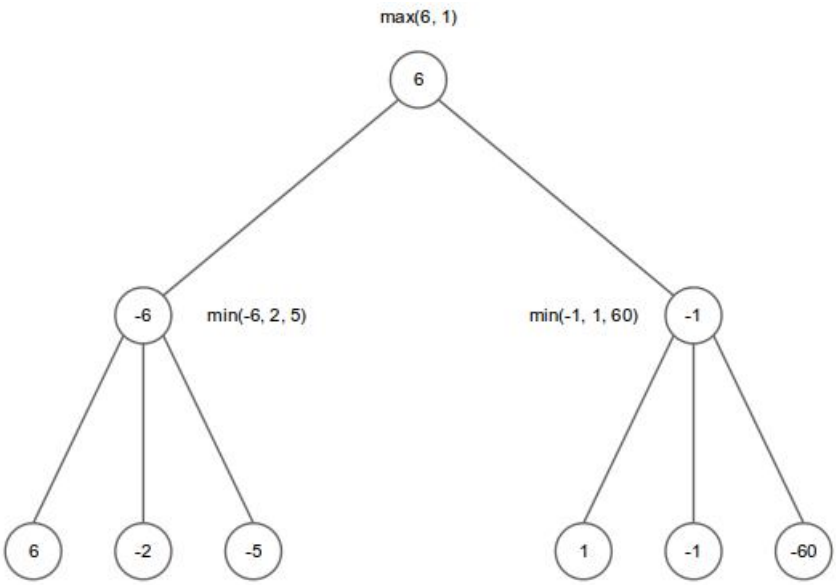

Figura 4: Exemplo de árvore usando o algoritmo Negamax.

já computadas no passado.

Foi bem interessante a discussão se deveríamos usá-las ou não. O grupo estava dividido. Uma parte achava que não havia tantas jogadas repetidas assim e, como se não bastasse, a complexidade de espaço dessa tabela seria absurdamente grande. Outra parte acreditava que ela seria responsável por diminuir o tempo de jogada do nosso agente, que já estava quase no limite do aceitável.

Mesmo com uma implementação não-trivial pela frente, implementamos a tabela de transposição e os resultados não podiam ser melhores. No quesito tempo gasto em média, a implementação com a tabela de transposição é executada em menos da metade do tempo da versão sem a mesma.

2.2.3 Posições estáveis

Além da tabela da figura 3, usamos também um conceito aprendido em [2] chamado **posições estáveis**. Uma posição é dita estável quando é impossível perdê-la para o outro jogador.

Rascunhamos um algoritmo determinístico para contar o número de posições estáveis em um dado tabuleiro. Sua complexidade, porém, era mais alta do que gostaríamos. Optamos por usar um algoritmo aproximativo, que possui um fator linear a menos, mas, ainda assim, oferece uma aproximação muito boa. Obtemos essa vantagem ao desconsiderar certos casos particulares.

É fácil ver que um canto do tabuleiro é a principal origem (porém não única) das posições estáveis, sendo o próprio canto uma posição estável. Nossa estratégia inicial, obter os corners, parece servir de complemento a estratégia de usá-los como fonte de posições estáveis.

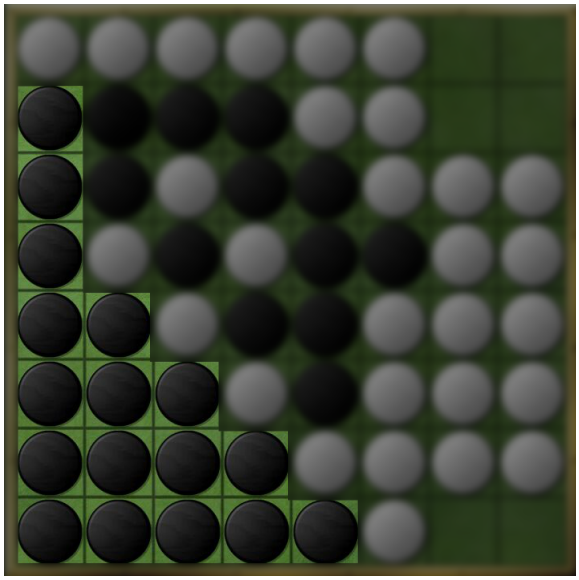

Figura 5: Nosso algoritmo considera as posições estáveis a partir de um corner. A complexidade dessa parte, em cada avaliação de vértice do Minimax, é $O(n^2)$.

Essa, com certeza, foi a otimização que mais impactou nosso agente. Obtivemos resultados muito bons com ela, de modo que ganhamos com uma frequência razoável a dificuldade mais difícil do aplicativo Reversi Free.

2.2.4 Peças de fronteira

Peças de fronteira são peças que limitam um ou mais espaços vazios. Quantos mais peças de fronteira o oponente tiver, menos chances ele terá de ganhar.

Ao reduzir o número de peças na fronteira, damos ao oponente o menor número de jogadas válidas possível. Quanto menos peças colocamos na fronteira, mais empurramos o oponente pra uma jogada ruim (ver figura 7).

Em outras palavras, minimizar o número de peças de fronteira é equivalente a minimizar o número de jogadas que o oponente possui.

2.2.5 Outras observações

Após as estratégias apresentadas terem sido implementadas, começamos a procurar por detalhes que poderiam deixar o agente mais rápido.

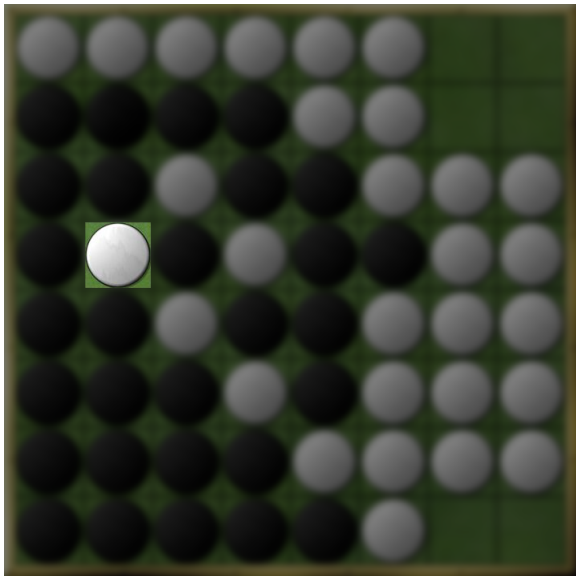

Figura 6: Como dissemos anteriormente, um corner não é a única fonte de posições estáveis. A posição destacada na figura acima não foi considerada pelo nosso algoritmo. Encontrar exatamente todas as posições estáveis requer complexidade $O(n^3)$ para cada avaliação de vértice do Minimax.

Uma modificação feita foi na forma de simular uma jogada. Reimplementamos as funções que fazem o movimento no tabuleiro de forma a retornar uma lista de peças que foram viradas. Assim sendo, apenas uma instância do tabuleiro foi mantida durante todo o tempo de avaliação da melhor jogada, poupando-nos das cópias desnecessárias, o que nos resultou em uma economia de cerca de 20% de tempo.

Outra heurística implementada foi a de ter duas tabelas de pesos para a avaliação de peças do tabuleiro, para partes diferentes do jogo. No início do jogo (até 20 peças no tabuleiro), os cantos e quinas receberam valores maiores do que os que receberam posteriormente. Isto possibilitou que o Bambam ganhasse de algumas IAs com alto nível de dificuldade, coisa que antes desta implementação não acontecia sempre.

Uma última coisa que vale ser mencionada na implementação foi a definição de um tempo limite de **30 segundos** por jogada. Por mais que o torneio não tenha estabelecido tal limite, não queríamos que o Bambam ficasse pensando por muito tempo e atrasasse a partida e, consequentemente, o campeonato. Quando o tempo limite é atingido dentro do Minimax, ele interrompe a execução, abandonando a árvore do Minimax, e retorna o melhor movimento encontrado até o momento.

3 Conclusões

Mais do que importante, este trabalho foi muito divertido de ser feito. Particularmente falando, o grupo se sentia um pouco desmotivado. Frequentemente achávamos a matéria tediosa e desinteressante. Poder brincar um pouco com o Bambam (e com todos os agentes

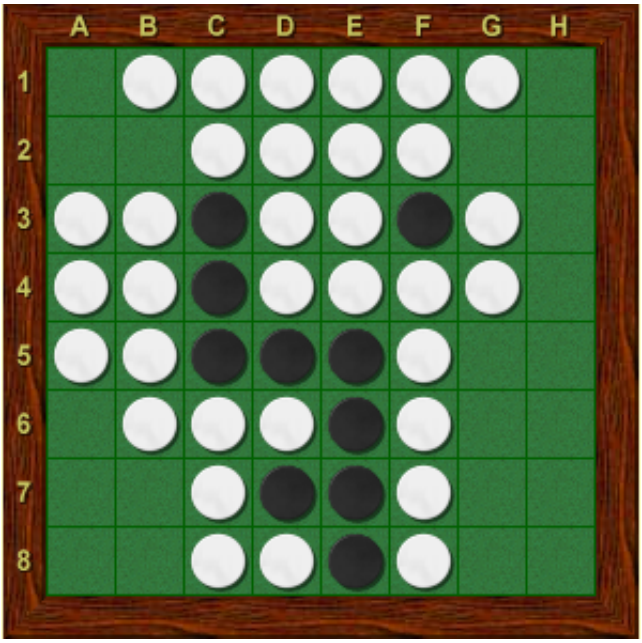

Figura 7: Observe o tabuleiro acima. O branco só tem dois movimentos, e ambos entregam o canto ao preto. O Bambam tenta replicar a mesma estratégia. Imagem retirada de [2].

implementados ao longo do caminho) nos fez observar a importância de certas conceitos (como a da tabela de transposição, como dissemos anteriormente) e, mais do que isso, de ver a importância da Inteligência Artificial na Computação.

Não sabemos como será o resultado da competição, mas já estamos satisfeitos com nosso resultado. Claro que queremos ganhar, mas a maior vitória foi o aprendizado obtido ao longo dessas semanas de projeto.

Referências

[1] OTHELLO – APRENDA A JOGAR. Game Over, YouTube. Disponível em: `https://www.youtube.com/watch?v=qqMFr2GorbA`. Acesso em: 01 de dez. 2016

[2] STRATEGY GUIDE. Gunnar Andersson's. Disponível em: `http://radagast.se/othello/Help/strategy.html`. Acesso em: 07 de dez. 2016

[3] NEGAMAX. Wikipedia. Disponível em: `https://en.wikipedia.org/wiki/Negamax`. Acesso em: 07 de dez. 2016