Projeto de Laboratório de Informática II
Big Two: Jogador Inteligente

Luís Ferreira(78607) Diogo Fernandes (77867) Jorge Cruz (78895)

29 de Maio de 2016

Resumo

O presente relatório descreve de forma sucinta a implementação de um algoritmo de inteligência artificial denominado de MCTS("Monte Carlo Tree Search") ao jogo Big Two, com o objetivo de criar um jogador virtual que opte pelas melhores jogadas possíveis.

Conteúdo

1 Introdução

Durante a nossa investigação para a escolha da estratégia de jogo mais adequada para o nosso jogador, deparamo-nos com um algoritmo de inteligência artificial que teve bastantes bons resultados quando aplicado aos mais variados jogos, sendo alguns destes Go, Pacman, xadrez, etc. Este baseia-se na construção de uma árvore de possíveis jogadas, em que para cada uma dessas se executa várias simulações para determinar qual a melhor, sendo que a jogada que se deve simular é escolhida segundo uma equação(por oposição a ser escolhida de forma aleatória, que é o que acontece noutros algoritmos de monte carlo).

Este método é algo *flexível*, como tal é possível que não tenhamos escolhido a melhor implementação possível, mas foi a que obteve melhores resultados nos nossos testes.

Devemos referir que por ser um algoritmo relativamente recente, a informação existente não é muita, mas encontra-mos duas fontes bastante úteis, que se encontram tanto no rodapé desta página[1][2], como na secção "**Referências**", sendo uma delas um documento publicado pelo IEEE, que no fundo se trata de uma compilação de informações e sugestões de otimização para este método.

Por último, devido a este algoritmo ser de um nível de complexidade um pouco elevado para os nossos conhecimentos, o código fonte final ficou pouco legível(e possivelmente pouco eficiente), sobretudo devido a ajustes de última hora e à falta de tempo para o tornar mais fácil de compreender.

[1] `http://www.doc.ic.ac.uk/~sgc/papers/browne_ieee12.pdf`
[2] `https://en.wikipedia.org/wiki/Monte_Carlo_tree_search`

2 Funcionamento

De seguida iremos descrever o funcionamento do algoritmo MCTS.

O primeiro passo é a construção da raiz da árvore. Trata-se de um nodo, que no nosso caso é igual a todos os nodos da árvore e contém os seguintes campos: estado, que descreve a jogada a ser tomada caso o nodo seja escolhido, que para este primeiro nodo será 0(pois não tem efeito); t, que indica o número de simulações realizadas para aquele nodo; r, que é a pontuação acumulada pelo nodo, a partir dos resultados que cada uma das simulações obteve; prev, que é um apontador para o nodo de onde veio; nextN[4][40], que é um array de apontadores para outras jogadas possíveis a partir do nodo atual, sendo o primeiro campo relativo ao número de cartas na jogada e o segundo à própria jogada.

De seguida, temos que estabelecer uma política de seleção e expansão de nodos, denominada de tree policy(e por esse motivo definida com o mesmo nome no nosso código), que trata de, partindo do nodo atual, escolher qual o nodo a ser simulado(tendo em mente que cada nodo representa uma jogada), ou no caso de se tratar de uma folha da árvore, de apender novos nodos à árvore(com novas jogadas, exeto se chegarmos a um nodo terminal). Para a seleção de nodos, é usado o algoritmo UCT(por extenso, "Upper Confidence Bound Apllied to Trees", que pode ser visto abaixo).

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

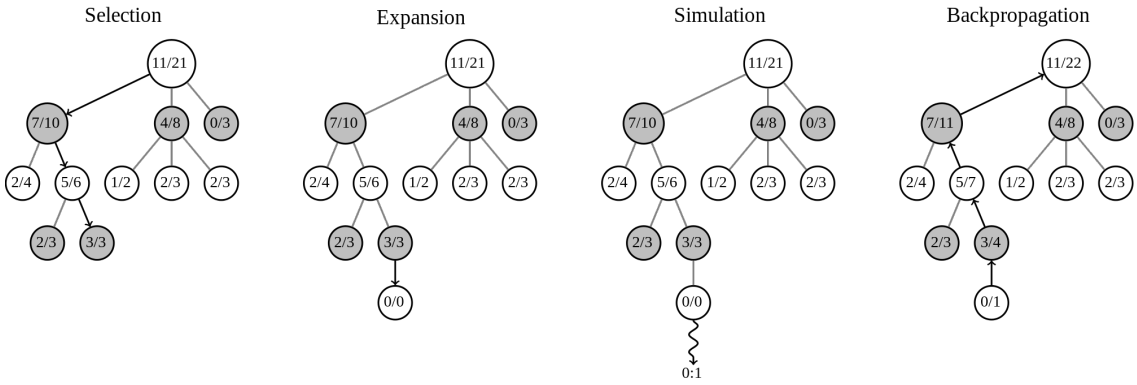
A primeira parte da equação favorece jogadas promissoras, com um *wirate* elevado, e a segunda favorece jogadas pouco exploradas. Isto faz com que se crie uma árvore assimétrica, com maior elaboração nas melhores jogadas. Os parâmetros wi e ni representam, respetivamente, a pontuação e número de simulações de um dado nodo. t representa o número de simulações do nodo parental e c é um valor que deverá rondar raiz de 2, mas que pode ser acertado conforme seja mais conveniente.

Após isso(e no nosso caso, de certa forma, em paralelo), é preciso estabelecer uma política de simulação de jogo denominada de default policy, e realizar a atualização das estatísticas dos nodos, a que se dá o nome de backpropagation. No nosso programa, o papel deste último é desempenhado pela função rewardF. Tal como acontece com o tree policy, também existe uma função no código fonte com o nome default policy, para manter a correspondência.

No processo de backpropagation, todos os nodos que foram percorridos são incrementados de um no número das simulações e é-lhes acrescida a pontuação correspondente ao número de cartas com que o nosso jogador terminou na simulação.

Por último, temos que escolher um critério para, após chegado o fim das simulações, escolher o nodo com a jogada mais promissora. Existem vários critérios para realizar esta escolha. De entre estes destacam-se: selecionar o nodo com maior número de simulações; selecionar o nodo com maior pontuação; selecionar o nodo com maior razão pontuação : num. simulações.

Como resumo, apresentamos um esquema com os 4 passos fundamentais do algoritmo: seleção e expansão(tree policy), simulação(default policy) e atualização de estatísticas(backpropagation).



3 Implementação

Agora que está estabelecido o modo de funcionamento do algoritmo, iremos explicitar a forma como o concretizamos.

A par da estrutura da árvore, existe uma estrutura do jogo, que contém a informação das cartas usadas por cada um, das mãos de cada um(que para os restantes jogadores são determinadas de forma aleatória para cada simulação), de quantas vezes seguidas se passou, do número de cartas a ser jogado na ronda atual(representado por nc, tendo o valor de 0 quando estamos no início de uma ronda), da última jogada realizada(para determinar jogadas possíveis) e de um outro parâmetro(firstplay) que serve para auxiliar a gestão da estrutura, antes de a nossa primeira jogada ser efetuada.

Existem duas destas estruturas de jogo, uma que contém as informações para o jogo real, e outra que começa igual à primeira em cada ronda de simulação, mas é alterada à medida que a simulação decorre.

Assim, começamos por construir o nodo da raiz. De seguida, é aplicado o tree policy que determina as jogadas possíveis, a partir do estado de jogo atual. Se uma jogada ainda não existir na árvore, é adicionado um novo nodo com essa jogada e esse é selecionado. Caso isso não aconteça, é atribuído um valor pela equação UCT a cada um dos nodos, e o nodo com o valor mais alto é selecionado.

O default policy vai receber o nodo selecionado pelo tree policy e a informação de se se trata de um nodo intermédio(primeiro ponto descrito no parágrafo anterior), ou de uma folha. No primeiro caso faz-se a simulação de apenas uma ronda, com o objetivo de atualizar a estrutura do jogo destinada à simulação, e volta-se a correr o tree policy a partir deste nodo, repetindo-se este processo as vezes necessárias para ser selecionado um nodo que corresponda a uma folha. No segundo caso, realiza-se uma simulação até se atingir um estado terminal(fim de jogo, e isto se não se encontrar já num estado como esse) e realiza-se o backpropagation através da função rewardF.

O rewardF determina uma pontuação entre 0 e 1, sendo que a 1 se retira 0.3, por carta, até um total de 9 cartas, após isso retira-se 0.6 por carta e no caso de nenhuma carta ser jogada é atribuída uma pontuação de 0.

Este processo é repetido, sendo que no fim de cada simulação, a estrutura de jogo da simulação é igualada à estrutura de jogo real, durante cerca de 2 segundos. Após isso, é aplicado um critério que analisa qual os nodos com maior número de simulações e pontuação e a partir daí seleciona um nodo para jogar. Do nodo selecionado retira-se a jogada que é imprimida para o standard output, o espaço de memória para todos os outros nodos que não foram selecionados, e para a

atual raiz é libertado, o nodo selecionado torna-se a nova raiz.

4 Referências

https://upload.wikimedia.org/math/f/4/e/f4ee057cfe174882f429f5b9e738d31b.png
https://upload.wikimedia.org/wikipedia/commons/thumb/b/b3/MCTS_(English).svg/2000px-MCTS_(English).svg.png
https://en.wikipedia.org/wiki/Monte_Carlo_tree_search
http://www.doc.ic.ac.uk/~sgc/papers/browne_ieee12.pdf
https://spin.atomicobject.com/2015/12/12/monte-carlo-tree-search-algorithm-game-ai/
http://chrislo.ca/?p=106
https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/