INE5430 - Inteligência Artificial

Trabalho T1 - Parte 1

Caique Rodrigues Marques
c.r.marques@grad.ufsc.br

Fernando Jorge Mota
contato@fjorgemota.com

06 de setembro de 2016

Função Heurística

Para uma máquina que queira jogar o jogo Gomoku é necessário que ela tenha uma boa estratégia a fim de conseguir garantir a vitória. A principal estratégia a se usar é considerar as possibilidades de jogadas em uma determinada partida, no entanto, é inviável uma máquina conseguir computar as várias jogadas e suas consequências possíveis, considerando o tamanho do tabuleiro e a quantidade de jogadas possíveis.

Uma heurística é determinada para que a máquina consiga melhores estratégias a partir de uma estimativa, neste caso, as situações que complementarão para a vitória ou para a derrota.

A máquina pode tanto trabalhar mais na ofensiva ou mais na defensiva, dependendo de como estiver a sua situação no jogo. As seguintes situações serão consideradas pela máquina para uma boa estratégia de jogada ofensiva:

- O número de peças usadas, disto, a possibilidade de formar uma dupla, ou uma tripla, ou uma quádrupla, ou quíntupla;
- A máquina deve considerar uma jogada onde se consiga formar uma tripla ou quádrupla ou quíntupla;
- Uma jogada que forma mais sequências contíguas também favorecem a vitória. Por exemplo, quando as peças estão dispostas onde forma um "L", tem duas sequências que é uma na horizontal e uma na vertical, aumentando as chances de vitória.

As seguintes situações são consideradas para uma jogada mais defensiva:

- Impedir o adversário de formar quíntuplas, quádruplas ou triplas;
- Considerar uma jogada em que evite que o adversário consiga formar sequências contíguas de peças, diminuindo as chances dele de conseguir, por exemplo, de formar duas triplas em forma de "L".

Ainda tem as ações que dê vantagens à máquina:

- Colocar as peças onde há mais possibilidades de formar sequências, por exemplo, colocar um peça no centro dá mais chances de sequência do que colocar nas bordas;
- Verificar quantas sequências já foram montadas (duplas, triplas e quádruplas) para uma possível quíntupla;
- Avaliar as sequências já formadas e ver se é possível na jogada em questão fechar uma sequência maior a partir de duas menores, por exemplo, duas duplas separadas apenas por uma casa, ao colocar uma peça no meio da dupla, forma uma quíntupla;
- A máquina também deve avaliar em quantas casas estão separadas duas fileiras de peças de mesma cor;
- O número de jogadas realizadas até o término no jogo.

Dadas todas as situações listadas, é possível atribuir ranks a cada uma, de forma que seja uma entrada de uma função. Com as possíveis situações à vista como entrada, qual será a saída gerada? A saída define o progresso da máquina no jogo, se está ganhando ou perdendo.

Portanto, define-se uma função v tal que

$$v(p, c, b, s) = (max(p) + c \times b) \times s$$

Onde $max(p)$ corresponde ao maior número de peças contíguas, c corresponde ao número de sequências (duplas, triplas e quádruplas), b corresponde ao número de sequências formadas após a jogada (se não houver sequências

contíguas, então $b = 1$) e s corresponde ao número de peças contíguas formadas após a jogada (se não formar sequência, então $s = 1$)

Define-se uma função l tal que

$$l(p, c, b, s) = (max(p) + c \times b) \times s$$

As variáveis têm o mesmo propósito da função v, mas referente ao adversário.

A seguinte função heurística $h(x)$ recebe as possibilidades de situações listadas anteriormente como entrada, cada uma possuindo um valor inteiro, e saída é a operação entre tais valores, onde quanto maior o valor de saída, maior será a vantagem da máquina.

$$h(v, l, p, j) = \frac{(v - l) \times p}{j}$$

Onde v e l correspondem às saídas das funções $v(p, c, b, s)$ e $l(p, c, b, s)$, respectivamente, p corresponde ao número de sequências de peças contíguas e j corresponde ao número de jogadas realizadas pela máquina até então.

Função Utilidade

A função de utilidade define uma certeza, que está próximo do final do jogo e ela define qual a melhor estratégia tomar para terminar o jogo da melhor maneira possível. É necessário que uma função de utilidade faça um balanço de tudo o que foi feito para, então, atribuir um peso final para a melhor decisão.

Portanto, a seguinte função descreve a utilidade:

$$u(x, seq, qt, jgds, aseq) = \frac{max(seq * qt)}{jgds + max(aseq)} \times x$$

Onde *seq* corresponde ao número de peças contíguas, *qt* corresponde ao número de sequências com mais peças contíguas, *jgds* corresponde ao número de jogadas até então realizadas e *aseq* corresponde ao número de peças contíguas do adversário tudo isso é multiplicado pelo valor de x que define se aconteceu uma vitória ou uma derrota ou um empate. No caso de vitórias, o valor é positivo ($x = +1$), derrota é negativo ($x = -1$) e em empate é neutro ($x = 0$).

Estrutura de Dados

A estrutura de dados usada no programa é imutável. Bastante similar com um grafo - mais especificadamente, uma árvore - cada objeto dessa estrutura possui uma série de atributos que são usados durante a execução do programa:

- Uma matriz representando o tabuleiro vazio - preenchido com "+"internamente, para fins de representação;
- Uma string representando o jogador que está jogando naquele estado - inicialmente, pelas regras do jogo, será o jogador "O", que representa a peça preta;
- Uma string preparada para armazenar uma mensagem a ser exibida para o usuário. Usada principalmente em casos de erro ou avisos;
- Um ponteiro para o estado que gerou originalmente o estado que se está usando;

A partir de um determinado estado, então, essa estrutura cria cópias de si mesma com apenas alguns atributos modificados e um ponteiro para a versão que a criou. Quando é feita uma jogada, por exemplo, um novo estado é criado modificando apenas a matriz e o ponteiro "pai"apontando para o estado no qual foi feito aquela jogada. Da mesma forma, quando é imprimida uma mensagem, é criado um novo estado apenas configurando a nova mensagem e o ponteiro "pai"correspondente.

No final das contas, temos, nessa estrutura, algo bem parecido com uma árvore/grafo: O estado inicial seria a raiz, e nenhum nodo filho seria a principio gerado sem que houvesse extrema necessidade. Dessa forma, é possível evitar uso desnecessário de memória e, pelo fato de cada objeto ser imutável, temos aqui uma vantagem: cada novo objeto só modifica os atributos modificados naquela operação. Todos os atributos que não são efetivamente modificados são referenciados por ponteiros no novo objeto criado, ajudando a economizar memória visto que a estrutura realiza cópias de todo atributo a cada operação feita.

Detecção de vitória

A detecção de vitórias na implementação do jogo Gomoku é feita de uma forma simples. O tabuleiro de Gomoku é representado como uma matriz, com quinze linhas e quinze colunas, e, a cada peça inserida no tabuleiro, é feita uma checagem, partindo dessa peça, avançando 4 peças em cada uma das 8 direções (superior, direita, esquerda, inferior, diagonal superior esquerda, diagonal superior direita, diagonal inferior esquerda, diagonal inferior direita). Se o algoritmo detecta que um dos caminhos não foi completado (como no caso em que há uma peça do adversário na direção percorrido e/ou há um campo vazio) então o algoritmo pula para a próxima direção possível.

Note que, devido a este comportamento, o número de iterações realizado é relativamente baixo: no melhor caso, que é quando a peça inserida somente tem espaço vazio em qualquer uma das oito direções - ou seja, é inserida em um lugar onde não há peças vizinhas da mesma cor, o algoritmo executa apenas 8 iterações, enquanto que no pior caso, que é quando o algoritmo tem 3 peças em cada direção possível, ele executa 32 iterações. E isso apenas quando uma peça é efetivamente marcada no tabuleiro, tornando todo o processamento bastante eficiente.

Além dessa checagem a cada jogada feita, também é disponibilizado um método que checa cada posição no tabuleiro e retorna o primeiro que forma uma sequência de 5 casas. O retorno desse método, assim como no caso do método anterior, é o jogador que formou a sequência de 5 casas.

Por fim, vale destacar que, usando esse algoritmo, é possível fazer a filtragem pelo jogador, de forma que o método retorna positivamente somente se o jogador informado tiver ganhado a partida.

Detecção de Maior Sequência

Além de disponibilizar métodos para detecção de vitória, também foram adicionados métodos para ajudar na detecção da maior sequência disponível tanto a partir de uma determinada peça quanto a partir do tabuleiro como um todo.

Este procedimento é feito usando o mesmo algoritmo apresentado na seção anterior, a única diferença básica e primordial é que ao invés de apenas parar quando são encontradas 5 peças em sequência, ele também vai contando o número de peças encontradas em cada direção e registrando o maior número de peças em sequência encontrado.

Ao final do processamento, o maior número de peças encontrado é retornado. Assim como o algoritmo abordado na seção anterior, este algoritmo também conta com a filtragem pelo jogador, de forma que é possível obter a maior sequência feita por um determinado jogador.