Universidade de São Paulo
Instituto de Matemática e Estatística
MAC 5739 - Inteligência Artificial

Exercício Programa 1: Jogador inteligente de Tetris

Autor:

Walter Perez Urcia

São Paulo

Setembro 2015

Resumo

Neste trabalho o objetivo foi desenvolver um jogador inteligente de Tetris usando heurísticas e buscas informadas e não informadas. Além disso, foi usado um conjunto de peças predefinida para fazer comparações entre as diferentes buscas em termos de ramificação, nós gerados e nós expandidos.

1 Problema

Esta seção descreve o problema a solucionar e a formulação usada nas diferentes soluções.

1.1 Descrição

O jogo Tetris é uma quebra-cabeça eletrônico que consiste em empilhar peças (tetraminós) que "descem"na tela de forma a "limpar"o maior número linhas horizontais. Quando uma linha se completa, ela se desintegra, as camadas superiores descem, e o jogador ganha pontos. A partida se encerra quando a pilha de peças chega ao topo da tela e o jogador não consegue mais limpar linhas.

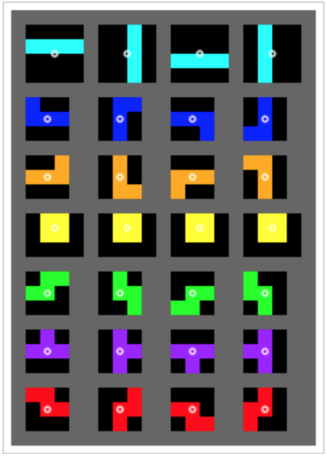

Figura 1: Os 7 tetraminós, respectivamente I, J, L, O, S, T, Z e suas possíveis rotações

Existem muitas variações do jogo Tetris, mas para este exercício serão adotadas as seguintes regras:

1. O tabuleiro é um retângulo de 10 células de largura por 22 células de altura, isto é, 10 colunas e 22 linhas
2. Todos os tetraminós começam no meio do tabuleiro nas linhas do topo
3. Existem 7 tipos de tetraminós I, J, L, O, S, T, Z

4. As ações disponíveis ao jogador são *mover para esquerda* (left), *mover para direita* (right) e *mover para baixo* (down), que têm por efeito alterar a posição da peça em jogo nas linhas e colunas na direção desejada, *despencar* (drop), que move a peça até para baixo o máximo possível, e *rotacionar* (rotate), que rotaciona a peça no sentido horário
5. Uma ação é válida em uma determinada configuração do jogo, se na configuração resultante não existe sobreposição entre a peça e as células ocupadas do tabuleiro
6. O tempo é discreto e a cada certo número de passos (timesteps) a peça atual se desloca para baixo uma unidade automaticamente, mesmo sem ação do jogador
7. As peças são geradas de forma aleatório e o jogador só possui conhecimento da peça atual

A abordagem usada neste exercício ataca o problema em duas etapas: (i) formulação de meta e (ii) busca de solução. Onde (i) consiste em encontrar a melhor posição para colocar a peça atual e (ii) procura encontrar uma sequência de ações que levem a peça atual à posição meta. Só a segunda etapa vai ser explicada em detalhe para este exercício.

1.2 Formulação do problema

O jogo Tetris pode ser formulado da seguinte forma:

- Estado: $< t, b, s >$ onde t é o tetraminó ou peça atual (linha x, coluna y e rotação), b é uma matriz binaria que representa as células ocupadas do tabuleiro e s é o timestep nesse instante (número de ações que o jogador ainda pode tomar antes que a peça caia automaticamente)
- Ações: *drop, noop, left, right, down, rotate* descritas em 1.1
- Estado inicial: a peça atual + tabuleiro vazio + timestep
- Estado final: tabuleiro com a peça atual na posição meta da primeira etapa

A formulação anterior será usada para desenvolver a segunda etapa do problema descrita em 1.1 com diferentes algoritmos de busca.

2 Algoritmos de busca

Todos os algoritmos de busca que foram desenvolvidos usam a seguinte definição do nós de busca:

- *state*: Estado do nó
- *parent*: Nó pai no árvore de busca

- *action*: Ação que gerou o nó
- *depth*: Profundidade do nó na árvore de busca
- g: Custo do caminho até o nó
- h: Heurística de custo até meta

Os dois primeiros algoritmos serão buscas cegas (ou não informadas), enquanto os dois seguintes serão buscas informadas. A principal diferença entre cada algoritmo é a estratégia de busca (ordem na qual nós são expandidos).

2.1 Busca em profundidade (DFS)

Nesta busca é escolhida a folha mais profunda. A borda é uma pilha, os nós são inseridos no começo.

2.2 Busca em largura (BFS)

A estratégia de busca é escolher a folha de menor profundidade não explorada. Borda é uma fila, novos nós são colocados no fim.

2.3 Busca de melhor escolha (BestFS)

Este tipo de busca precisa ter uma função heurística $h(n)$ que prediz o custo do nó n para chegar até a meta. Uma heurística que poderia ser considerada é a distância de Manhattan:

$$dist(p1, p2) = |p1.x - p2.x| + |p1.y - p1.y|$$

Onde x é a coluna e y a fila no tabuleiro. No problema, a heurística poderia ser $h(n) = dist(p_n, p_{meta})$, onde p_n é a posição da peça atual no nó n e p_{meta} a posição meta. Mas tem um problema com aquela heurística porque não é admissível, ou seja, podem ser obtidos valores maiores que o custo ótimo para chegar a meta. Fazendo algumas modificações a aquela heurística temos:

$$h(n) = |p_n.x - p_{meta}.x| + (p_n.y > 0)$$

O termo da diferença das filas das posições foi eliminada porque quando a peça está na mesma coluna da posição meta só precisa fazer uma operação *drop* para chegar. Então a estratégia de busca só é escolher o nó com menor valor de $h(n)$ ainda não explorado.

2.4 Busca A^*

Escolhe o nó com menor valor para $f(n) = h(n) + g(n)$ ainda não explorado, onde $g(n)$ é o custo do caminho até o nó n e $h(n)$ é a heurística descrita em 2.3. Pode ser assumido que cada ação tem custo 1 e portanto a função $g(n)$ aumenta em 1 em cada expansão.

3 Experimentos e resultados

Nesta seção será detalhado o experimento feito usando os diferentes métodos de busca desenvolvidos. Além disso, os resultados serão mostrados para fazer algumas comparações.

3.1 Considerações geráis

Para o experimento foi considerado uma sequência fixa de peças para todos os métodos de busca. A sequência é SZSTI (ver Figura 1). Além disso, as seguintes medidas de comparação foram calculadas:

- Tamanho da solução (d)
- Nós gerados (ng)
- Nós expandidos (ne)
- Fator de ramificação (b)

3.2 Resultados

A tabela 1 mostra os resultados para o conjunto de peças descritos em 3.1 para todos os métodos de busca.

4 Conclusões

Com os dados na tabela 1, pode-se concluir em geral que:

- Usando DFS, o tamanho da solução sempre é maior a outros tipos de busca porque sua estratégia é escolher o nó com maior profundidade sem considerar alguma informação
- DFS tem menor valor do fator médio de ramificação porque, em geral, expande quase todos os nós que foram gerados devido a sua estratégia
- O número de nós gerados e nós expandidos é consideravelmente maior usando BFS que outros tipos de busca. Além disso, usa mais memória porque sempre salva todos os nós na memória
- Sempre é obtida uma solução de tamanho menor usando BFS porque sempre expande os nós ordenados pela profundidade
- A busca de melhor escolha (BestFS) não precisa gerar nem expandir muitos nós para encontrar uma solução, mas o tamanho da solução não é sempre o menor de todos os métodos. Além disso, não precisa expandir muitos nós para encontrar uma solução e por isso tem o maior fator médio de ramificação entre todos os métodos

Peça	Busca	d	ng	ne	b
S	DFS	36	1701	1620	**1.05**
	BFS	**13**	25716	14059	1.83
	BestFS	**13**	**59**	**22**	2.68
	A^*	**13**	1282	753	1.70
Z	DFS	36	490	407	**1.20**
	BFS	**13**	25803	14149	1.82
	BestFS	17	**107**	**38**	2.82
	A^*	**13**	1888	1104	1.71
S	DFS	31	57693	57623	**1.00**
	BFS	**15**	75437	42450	1.78
	BestFS	19	**88**	**35**	2.51
	A^*	**15**	2894	1735	1.67
T	DFS	35	144	58	**2.48**
	BFS	**1**	**6**	**2**	3.00
	BestFS	**1**	**6**	**2**	3.00
	A^*	**1**	**6**	**2**	3.00
I	DFS	27	727255	727203	**1.00**
	BFS	**17**	192651	116067	1.66
	BestFS	25	**165**	**87**	1.90
	A^*	**17**	6753	4096	1.65

Tabela 1: Medidas para cada peça e cada método de busca. Em vermelho os valores maiores e em preto os valores menores para cada medida

- Com a busca A^* sempre é obtida uma solução de tamanho igual a busca BFS, mas a diferença entre os nós gerados por A^* e BFS é muito considerável apesar que o fator médio de ramificação é muito parecido entre ambas buscas. Da mesma forma para os nós expandidos

Por último, uma possível melhoria ao sistema poderia ser considerar a seguinte peça que vai ser jogada. Por outro lado, poderia encontra-se uma melhor heurística para a busca BestFS e A^* dado que aqueles são os métodos mais rápidos e que consomem menos memória. Da mesma forma, para a heurística da primeira etapa (aquela que encontra a melhor posição para a peça) poderiam ser feitos experimentos para obter a melhor combinação de valores dos fatores de importância nos termos da função.