MAC 425/5739 - Inteligência Artificial
Primeiro Exercício-Programa (EP1)

Prazo limite da entrega: 23:59:59 16/9/2014

1 Introdução

Neste exercício-programa estudaremos a abordagem de resolução de problemas através de busca no espaço de estados, implementando um jogador inteligente de Tetris. Os objetivos deste exercício-programa são:

(i) compreender a abordagem de resolução de problemas baseada em busca no espaço de estados;

(ii) estudar uma formulação de problema de busca para criar um jogador automático do jogo Tetris;

(iii) implementar algoritmos de busca informada e não-informada e comparar seus resultados.

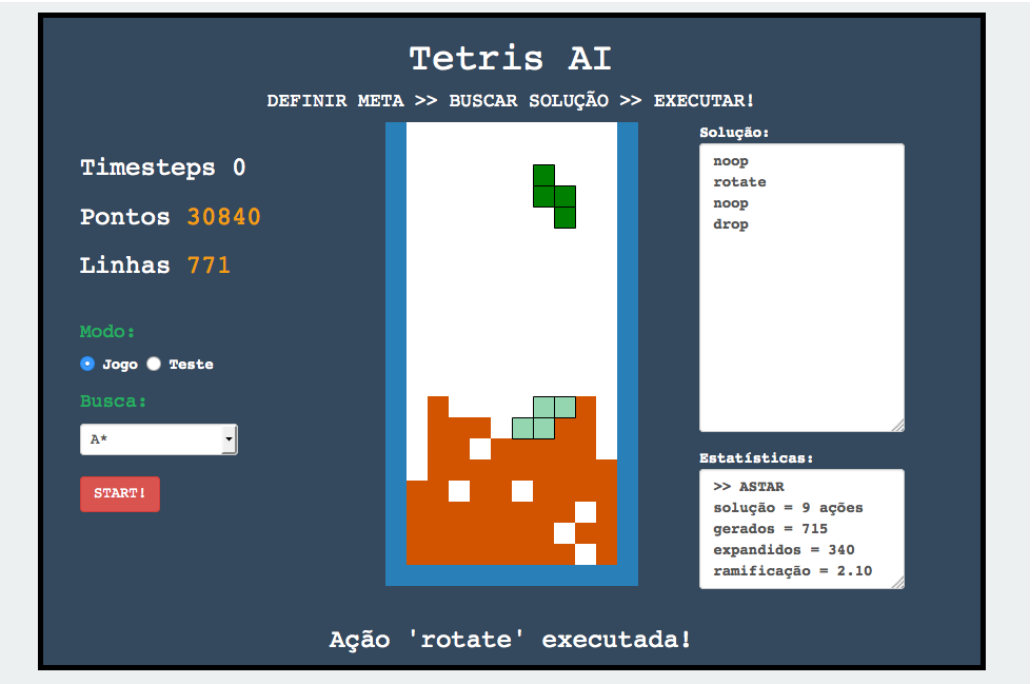

Figura 1: Interface gráfica do jogo.

O jogo Tetris é um 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.

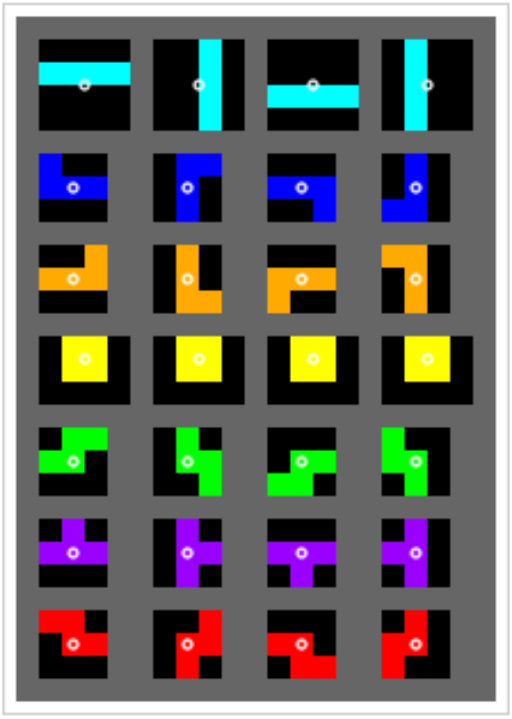

Figura 2: Os 7 tetraminós, respectivamente I, J, L, O, S, T, Z, e suas possíveis rotações

Existem muitas variações do jogo Tetris, para esse exercício adotaremos 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", "O", "J", "L", "S", "Z", "T" (veja a Figura 2);
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 valida em uma determinada configuração do jogo, se na configuração resultante não existe sobreposição entre a peça e as células ocupadas do tabuleriro (considere que a borda é feita de células ocupadas);
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 (p.ex., ação `noop`);
7. As peças são geradas de forma aleatório e o jogador só possui conhecimento da peça atual.

Uma abordagem simples para resolver o jogo de Tetris é considerar apenas um turno por vez (um turno compreende o instante desde que uma nova peça é inserida no tabuleiro até o momento em que ela é consolidada – quando não existem mais movimentos válidos). Em cada turno o agente deve determinar uma sequência de ações que leva a peça atual até uma configuração (posição e rotação) satisfatória. Nossa abordagem ataca este problem em duas etapas: (i) formulação de meta e (ii) busca de solução. A formulação de meta encarrega-se de analisar todas as configurações terminais de turno possíveis e aplicar heurísticas (p.ex., o número de linhas removidas, a altura da linha mais alta etc.) para selecionar a meta. A segunda etapa então procura encontrar uma sequência de ações que leve a peça de sua configuração inicial à configuração meta ou responda que a meta é inalcançável (e uma nova meta deve ser formulada).

Para simplificar, o algoritmo de formulação de meta já está implementado no arquivo `src/js/goal.js`. Sua tarefa neste EP é implementar algoritmos de busca que encontrem uma sequência de ações que levem a peça de sua configuração incial a uma determinada configuração meta.

ATENÇÃO: o arquivo `src/js/goal.js` não deve ser modificado! Se quiser, você pode lê-lo e usá-lo para formulas heurísticas de busca, mas qualquer modificação sua a este arquivo será desconsiderada.

2 Tetris como problema de busca

Esta seção descreve a formulação do espaço de estados utilizado.

2.1 Representação dos estados

A representação do estado da busca é definida em termos de uma tripla $\langle t, b, s \rangle$, onde t representa o tetraminó em jogo (linha x, coluna y e tipo/rotação do tetraminó), b representa as células ocupadas do tabuleiro em uma matriz binária e s representa o *timestep*, isto é, o número de ações que o jogador ainda pode tomar antes que a peça caia automaticamente.

```
var State = function (t, b, s) {
  this.tetromino = t;   // t.xpos, t.ypos, t.type
  this.board = b;       // b.matrix[WIDTH][HEIGHT]
  this.timestep = s;    // 0 <= s <= TIMESTEPS
};
```

2.2 Representação das ações

A representação das ações é o mais simples possível. Cada ação é uma representada por uma string dentro do conjunto {`"drop"`, `"noop"`, `"left"`, `"right"`, `"down"`, `"rotate"`} definido de acordo com os possíveis comandos do jogador. Dessa forma uma solução do problema de busca (i.e. uma sequência de ações) é representado por um array de strings, por exemplo, `["right", "right", "right", "rotate", "rotate", "drop"]`.

ATENÇÃO: para diminuir o espaço de estados e simplificar o problema de busca, assumimos que após cada ação executada do jogador o tetraminó corrente automaticamente se desloca para a linha de baixo (i.e. TIMESTEPS = 1).

2.3 Formulação do problema de busca

A formulação do problema de busca (disponível no arquivo `src/js/problem.js`) define as funções que serão utilizadas pelos algoritmos de busca para explorar o espaço de estados:

```
Problem.prototype = {
  initialState: initialState, // retorna o estado inicial.

  // Actions(s): dado um estado s retorna as ações aplicáveis no estado s.
  Actions: function (s) {
    var applicable = [];
    // ...
    return applicable;
  },

  // Result(s, a): dado um estado s e uma ação a,
  // devolve o estado resultante de aplicar a ação a no estado s.
  Result: function (s, a) {
    // ...
    return new State(...);
  },

  // GoalTest(s): dado um estado s devolve true se s é um estado meta.
  // Caso contrário, devolve false.
  GoalTest: function (s) {
    // ...
    return checkType && checkPosition;
  },

  // StepCost(s, a): dado um estado s e uma ação a,
  // devolve o custo de se aplicar a ação a no estado s.
  StepCost: function (s, a) {
    return 1;
  }
};
```

ATENÇÃO: o arquivo `src/js/problem.js` não deve ser modificado! Você deve apenas entender seu funcionamento para poder implementar os algoritmos de busca.

3 Implementação

Esse exercício-programa deverá ser implementado na linguagem JavaScript[1]. Utilizaremos um navegador Web (preferencialmente Mozilla Firefox) para exibir a interface gráfica do jogo. Após extrair o conteúdo do arquivo `tetris.tar.gz`, você deverá encontrar o arquivo `src/index.html`. Abra este arquivo no navegador para visualizar a interface gráfica. Os arquivos de interesse para esse EP estão disponíveis em `src/js/`. Além disso, no arquivo `src/js/utils.js` você encontrará implementações das estruturas de dados que você utilizará para desenvolvimento dos algoritmos de busca (pilhas, filas, filas de prioridade e conjuntos).

Após finalizada sua implementação, você poderá visualizar seu jogador automático jogando partidas aleatórias de Tetris (**modo jogo**) ou testá-lo em uma sequência de peças pré-determinada (**modo teste**). Essa sequência de peças servirá como "padrão" para comparação dos resultados de cada tipo de busca e confecção do relatório.

ATENÇÃO: não esqueça de recarregar completamente a página após cada alteração do seu código para evitar problemas com a cache do navegador! Se você alterar o seu código e esquecer de recarregar a página, o código JavaScript que continuará sendo executado será o código anterior às suas modificações..

3.1 Parte prática

No arquivo `src/js/search.js` encontra-se disponível um "esqueleto de código"para os seguintes algoritmos de busca:

- **DFS**: busca em profundidade;
- **BFS**: busca em largura;
- **BestFS**: busca de melhor escolha;
- **ASTAR**: busca A*.

Para esse exercício-programa, você deverá realizar as seguintes tarefas:

(a) completar a implementação de todas as buscas nas partes indicadas no arquivo `search.js`;

(b) modificar a heurística da distância de manhattan para torná-la admissível;

(c) verificar o correto funcionamento dos algoritmos através da interface gráfica do jogador automático.

BONUS: implementar 1 ou mais heurísticas e comparar seus resultados com a heurística da distância de manhattan. Atenção: para o desenvolvimento de outras heurísticas pode ser necessário acessar a representação do jogo de Tetris que está implementada no arquivo `src/js/model.js`.

ATENÇÃO: a implementação dos algoritmos de busca deve ser realizada no arquivo `src/js/search.js` nos espaços indicados mantendo a assinatura de cada função. Você não deve alterar os outros arquivos!

[1]Utilizaremos a linguagem JavaScript como linguagem de propósito geral, não havendo necessidade de ter conhecimento em programação Web, tampouco utilizar quaisquer frameworks. No site da disciplina no PACA encontra-se um Tutorial JavaScript. Mesmo que já tenha conhecimento da lingugem, acreditamos que possa ser útil estudar o material antes de começar sua implementação.

3.2 Relatório

Após o desenvolvimento da parte prática, você deverá testar seus algoritmos e redigir um relatório claro e sucinto. Assim, você deverá:

(a) executar a sequência de testes (i.e. rodar seu código no modo teste) para cada busca e recuperar os resultados no console do navegador;

(b) compilar em tabelas os resultados dessas execuções e compará-los no relatório;

(c) discutir os méritos e desvantagens de cada método, no contexto dos dados obtidos;

(d) sugerir possíveis melhorias ao sistema.

ATENÇÃO: Você deve comparar os resultados (tamanho da solução, número de nós gerados, número de nós expandidos e fator médio de ramificação) em função das propriedades de otimalidade e complexidade de tempo e espaço dos algoritmos. Além disso, você deve correlacionar os resultados obtidos com o comportamento do jogador automático visualizado na interface gráfica.

4 Entrega

Você deve entregar um arquivo comprimido através do sistema PACA até o dia 16/9/2015 às 23:59:59 contendo:

(1) o arquivo **search.js** contendo as implementações da parte prática;

(2) um arquivo PDF contendo o **relatório** com a comparação e discussão dos resultados (máximo de 5 páginas).

Não esqueça de identificar cada arquivo com seu nome e número USP! No código coloque um cabeçalho em forma de comentário.