Projeto de Inteligência Artificial 2021

TETRIS

96123 - Lucius Vinicius
97606 - Diogo Monteiro
100055 - Afonso Campos

Agent Loop

O loop do agente consiste no seguinte:

- Enquanto não fechar a ligação com o servidor do jogo:
  - Receber update do servidor
  - Para a primeira mensagem recebida: Obter dimensões do jogo
  - Determinar se a peça atualmente em jogo é nova (ou seja, a anterior já encaixou)
  - Se a peça é nova:
    - Determinar melhor posição possível
    - Determinar sequência de comandos necessários para colocar a peça na posição ideal
  - Senão:
    - Enviar comandos ao servidor

Calcular a melhor posição da peça segue os seguintes passos:

- Achar todas as posições possíveis para a peça
- Classificar todas as posições encontradas
- Usar as N melhores posições para avaliar as peças seguintes recursivamente, onde N é uma constante definida por nós.
- Escolher a posição com melhor classificação

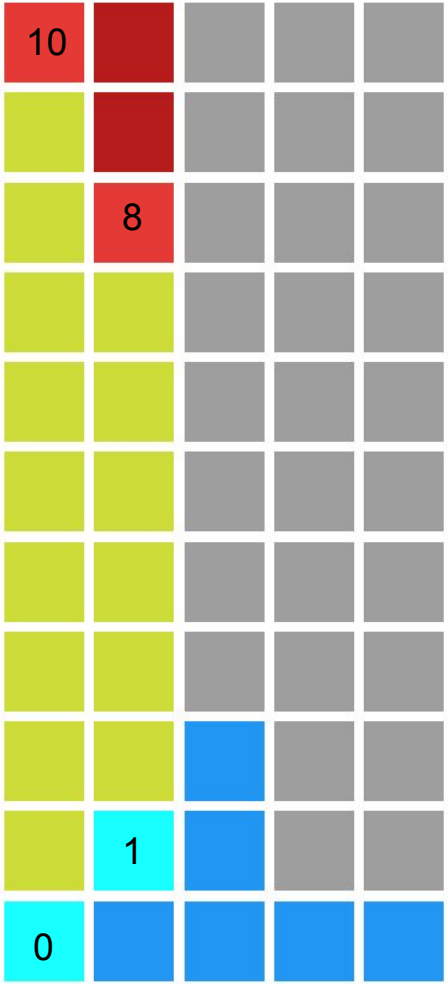
Get Possible Placements

`get_possible_placements`: dada uma peça e o ‘floor’ (lista dos pontos mais elevados de cada coluna) de um game state, queremos todas as possíveis posições para a peça.

Analisando as colunas que a peça vai ocupar, podemos detectar em que coluna vai ocorrer a colisão da peça com o floor através da distância entre o ponto mais baixo da peça e o ponto mais alto do floor:

Exemplo à direita:

Col1: 10-0 = 10; Col2: 8-1 = 7; Logo, a colisão vai se dar na coluna 2. Para obter as coordenadas finais da peça, subtraímos a menor diferença a cada Y da peça.



Get Best Placement

`get_best_placement` é a função recursiva que determina a melhor posição para uma certa peça dado um estado do jogo:

1. São calculadas as X melhores jogadas possíveis para a peça;
2. Caso exista lookahead (>0):
    a. Para cada jogada acima calculada é incrementado ao seu score a melhor posição da seguinte peça, através duma chamada recursiva à função;
    b. Retorna-se a jogada com maior score após a avaliação das peças seguintes.
3. Caso não exista lookahead:
    c. Retorna-se a jogada com maior score.

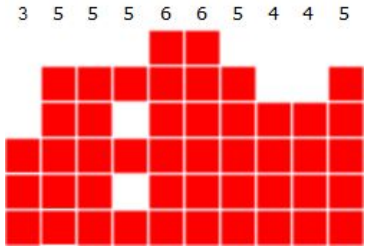
Variáveis globais:

`LOOK_AHEAD: int` -> Número de peças seguintes a considerar na seleção de melhor peça;
`LOOK_AHEAD_WEIGHT: [P0,P1,P2,...]` -> Lista de pesos a atribuir ao score de cada peça sucessiva;
`PLACEMENTS_LIM: [P0,P1,P2,...]` -> Limite de peças seguintes a considerar para lookahead da peça seguinte (pruning).

Heurísticas

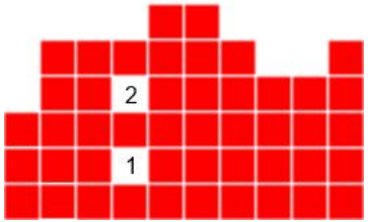
As heurísticas para classificar jogadas são implementadas com a função **`evaluate_placement`**, que recebe a posição a avaliar e o estado do jogo antes da jogada ser realizada.

Quatro heurísticas foram usadas:



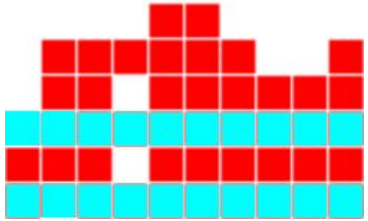
Soma de alturas

Somatório das alturas das colunas. No exemplo da esquerda:
3+5+5+5+6+6+5+4+4+5 = 48
Objetivo: Minimizar



Número de buracos

Um buraco é definido como sendo um espaço aberto numa coluna onde existe um bloco algures acima.
Objetivo: Minimizar



Número de linhas completas

Objetivo: Maximizar

“Bumpiness”

Soma das diferenças de altura entre colunas: No exemplo da esquerda:
1+5+1+0+1+1+6+5+0 = 20
Objetivo: Minimizar

As heurísticas e as constantes usadas foram retiradas do seguinte artigo:
https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/