TETRIS

José Domingues, 80075
Bruno Castro, 80190

LECI - IA 2021/2022

universidade de aveiro
deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

Algoritmo

```
best = {'score': None, 'position': 0, 'rotation': 0}
for rotation = 0 to rotation < len(rotations):
    figure = get the figure from the rotation
    for position = MaxLeft to position < MaxRight:
        newgame = figure + game
        if figure == "I":
            score = calculate the score of the newgame for the "I" piece
        else:
            score = calculate the general score of the newgame
        if best.get('score') == None or score > best.get('score'):
            best['score'] = score
            best['position'] = position
            best['rotation'] = rotation
    rotate()
```

Figura 1 - Pseudocódigo da lógica aplicada para a obtenção da melhor posição e rotação

- Leitura da peça atual e do estado do jogo
- Iterar todas as rotações sobre todas as posições possíveis
    - Simular a queda da peça no estado atual do jogo
    - Calcular o score do novo estado do jogo
- Descobrir a posição/rotação para o qual o score é maior
- Enviar as teclas, que guiam a peça para o objetivo, para o client

Heurísticas

Bumpiness

Variação da altura das colunas.
Este valor deve ser **minimizado.**

Altura Agregada (AG)

Soma das alturas das colunas.
Este valor deve ser **minimizado.**

Buracos

Número de buracos.
Foi considerado como buraco todo o espaço vazio cuja altura é igual ou inferior à altura máxima atual.
Este valor deve ser **minimizado.**

Linhas Completas (LC)

Número de linhas completas.
Este valor deve ser **maximizado.**

$$score = a \times AG + b \times LC + c \times Buracos + d \times Bumpiness$$

$$a = -0.79$$
$$b = 0.82$$
$$c = -0.10$$
$$d = -0.25$$

O objetivo é maximizar o score!

Score para a peça “I”

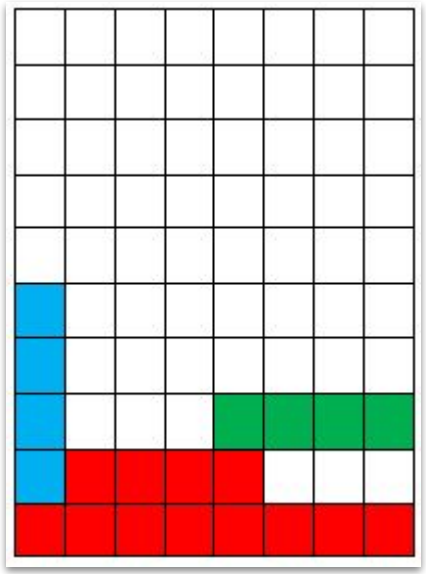

Figura 2 - Exemplo de duas jogadas distintas

Da maneira que o score foi calculado no slide anterior, o nosso agente optaria por colocar a peça “I” na posição e rotação representada a verde, na figura 2, visto que desta forma a quantidade de buracos é muito menor. Porém, nós consideramos que colocar a peça na forma representada a azul seria uma melhor jogada.

Foi implementado um novo método que calcula o score. Este método é usado exclusivamente para a peça “I”.

As constantes que são multiplicadas pelo bumpiness e pelo número de buracos foram aumentadas (diminuídas em módulo).

$$Score_{\text{“I”}} = a \times AG + b \times LC + c \times Buracos + d \times Bumpiness$$

$$a = -0.69$$
$$b = 0.82$$
$$c = -0.04$$
$$d = -0.10$$

Funções implementadas

- **aggregateHeight(game):** Calcula a altura agregada do jogo
- **bumpiness(game):** Calcula as variações das alturas das colunas do jogo
- **holes(game):** Calcula o número de buracos do jogo
- **completeLines(game):** Calcula o número de linhas completas do jogo
- **simulateGravity(game,piece,x):** Adiciona ao jogo, a peça atual que foi movida |x| vezes na horizontal ($x \in [-3,8[$)
- **heuristic(game):** Calcula o score do jogo
- **heuristic2(game):** Calcula o score do jogo. Este método é usado exclusivamente para a peça “I”
- **best_rotation_position(game,piece):** Retorna a posição/rotação que a peça deve obedecer para cumprir o objetivo
- **bestKeys(best_rotation_pos):** Retorna a lista de teclas que guia a peça para o objetivo
- **Outras (intersectsGrid(piece,x,y), originalShape(piece), …):** Funções ‘auxiliares’