

TPG-TETRIS

EQUIPA 2

DETI – Universidade de Aveiro

Inteligência Artificial

Prof. Diogo Gomes

Prof. Luís Lopes

André Clérigo, 98485

Pedro Rocha, 98256

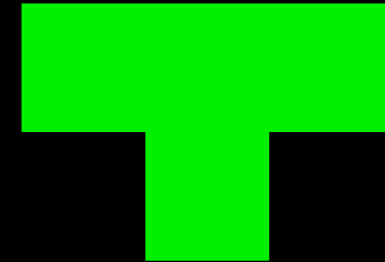


PEÇAS VIRTUAIS

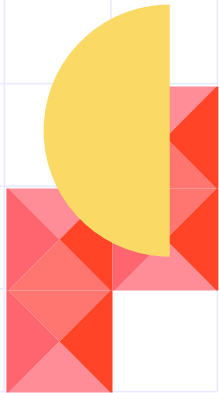
Usamos as peças virtuais quando queremos testar os vários tipos de rotações que uma peça pode ter. Vistos que o nosso agente calcula qual a melhor posição final passando ao servidor a lista de *keys* necessárias, só chegamos a analisar a peça no início do jogo.

Deste modo criámos um dicionário com todas as coordenadas iniciais de qualquer tipo de peça em todos as rotações possíveis. Para reutilizar características das peças, associamos também os valores de **base_min**, **base_width**, **width** e **ovfl** a cada peça virtual.

A **base_min** retorna a menor coordenada de X na **base** da peça (Y de maior valor), a **base_width** retorna o número de blocos que a peça ocupa na sua **base**, **width** retorna o número de blocos **máximo** que a peça ocupa e o **ovfl** representa o número de blocos que estão à esquerda do ponto extremo da base (**base_min**).



base_min: 3
base_width: 1
width: 3
ovfl: 1



FUNCIONAMENTO DO AGENTE

Dadas as coordenadas da peça reconhecer qual é o seu formato e associar N estados de rotação

Reconhecimento de peça

Dado um determinado *target* simula-se a possível interseção da peça com o *game atual* na posição em análise

Simular rotações

Interseção virtuais

Geração de *targets*

Dada uma lista de estados de **rotações** é simulado todas as coordenadas virtuais da peça

Dada a interseção e as suas possíveis alterações é adicionado um *target* à **lista de targets** possíveis para a dada peça

FUNCIONAMENTO DO AGENTE

Para cada estado do *game* esperado calculamos o custo associado, isto é, nº de *buracos*, *linhas* completas, colunas *vazias*, *altura* agregada e *irregularidade*

Por fim gera-se um mapa onde se tem acesso à lista de *keys*, custo, coordenadas do *target* e coordenadas da peça para cada jogada possível

Calcular custos

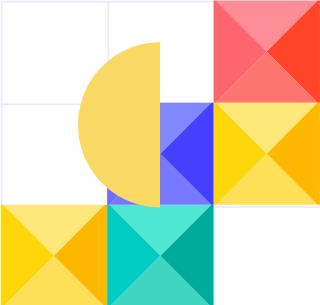
Criar lista de *keys*

Geração de mapas

Decisão da melhor jogada

De seguida é **associado** a cada *target* a lista de teclas necessárias para que a peça atinja o mesmo

Dada a geração de todos os mapas para cada jogada de cada rotação, ordenamos por menor custo e passamos a lista de *keys* **respetiva**



FUNÇÕES ADICIONAIS

GET_FREE_GRID

Dado um estado de jogo, isto é, lista de coordenadas ocupadas na *grid* do Tetris.

É gerado um dicionário de posições complementares, ou seja, coordenadas livres na *grid*. Para facilitar o uso futuro deste dicionário criamos uma chave **Y da *grid*** e associando-lhe a lista de posições livres.

GET_NEIGHBORS

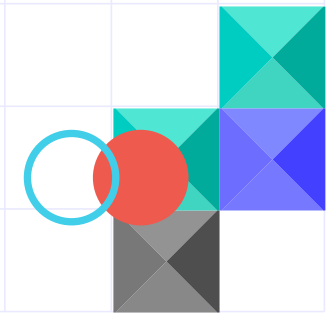
Dado um *target* e uma *free_grid* é retornado uma lista de coordenadas livres e que estão imediatamente à direita do *target*.

Esta lista tem um comprimento compreendido entre 0 e 3, isto porque, no máximo cada peça só ocupa 4 coordenadas.

COL_IS_EMPTY

Dado um *target*, uma *free_grid* e as coordenadas da peça, verificamos se todas as coordenadas acima do *target* estão livres.

Só são verificadas as coordenadas até a um Y igual à altura máxima atingida pela peça. A função é booleana, e por isso, apenas verificamos se é True ou False.



ANÁLISE DO ESTADO DO JOGO

- **Linhas completas**

Como o nome descreve, linhas completas devolve o número de linhas que vão ser completadas.

- **Altura agregada**

Calcula a soma de todas as alturas, usado facilmente usando uma *sorted_cols*, dicionário organizado pelas coordenadas de X com a lista de posições ocupadas.

- **Buracos**

Analisar todas as linhas do jogo (usando o dicionário *free_grid*) e verificar se todos as coordenadas têm a coluna vazia, caso não tenha, existe um buraco.

- **Irregularidade**

Calcula a soma da diferença (em módulo) de altura entre uma coluna e a próxima coluna à sua direita. É de notar, que a última coluna é ignorada.

- **Colunas vazias**

Devolve a diferença da altura entre duas colunas adjacentes caso esta diferença seja maior ou igual a 3.

