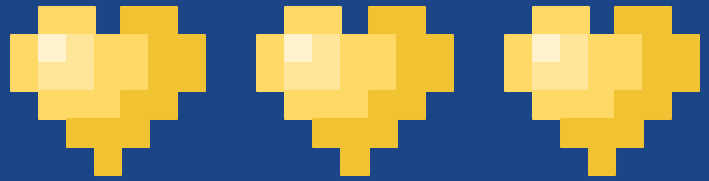
IA 2021/2022



DESENVOLVIMENTO DE UM AGENTE AUTÓNOMO PARA O TETRIS

Edgar Sousa (98757)
João Castanheira (97512)

Estrutura do Código

O código está dividido em 2 ficheiros:

1. **student.py**

   Ficheiro baseado no client.py em que a função **state_handler** faz a a tradução do **state** para um Engine de bytes fazendo uso da função **from_json()**, de maneira a ser computacionalmente mais leve o cálculo da melhor jogada. Neste ficheiro é também chamada a função **calc_states** que vai proceder à simulação das jogadas. Por fim, é neste ficheiro que é retornada a melhor jogada para o servidor.

2. **engine.py**:

   Este ficheiro contém a classe **Engine**, onde se cria um engine para o Tetris, muito semelhante ao fornecido pelo professor, instanciando todos os elementos necessários para simular as jogadas, tendo a particularidade de que estes foram todas traduzidos para bytes, de maneira a ser computacionalmente mais leve calcular todas os estados possíveis de um jogo, a partir de uma jogada.

Heurísticas

As heurísticas usadas para conseguir calcular a melhor jogada foram a **agg_height**, **bumpiness**, **lines** e **holes**.

agg_height: Esta variável guarda a soma das alturas de cada coluna, através de um sum() do tuplo height.

bumpiness: Esta variável guarda um valor que é obtido a partir da soma de diferenças entre duas colunas adjacentes de um determinado estado.

lines: Esta variável guarda o número de linhas limpas que uma determinada jogada consegue obter.

holes: Esta variável guarda o número de buracos que uma determinada jogada deixa num estado.

Estas heuristícas são depois usadas para calcular o custo de uma determinada jogada, ao serem associados pesos a cada heurística, permitindo depois avaliar a melhor jogada, sendo essa a que tiver maior custo.
Os pesos foram inspirados nos usados neste projeto do github: https://github.com/takado8/Tetris

Função calc_states

Esta é a funcão principal do nosso código, a função que simula todos os estados possíveis após uma jogada. É uma função da classe **Engine** pelo qual, é invocada sem argumentos, sendo que usa os atributos do objeto para obter os resultados.

A função analisa as jogadas possíveis para uma determinada peça, obtendo-as através de um dicionário de jogadas possíveis.
O uso do dicionário permite facilitar a implementação de mover e rodar as peças. Depois, para cada jogada testa esse input no **Engine** e guarda o custo dessa jogada como atributo. Todas os estados obtidos de um determinado estado são depois guardados numa lista, que é a depois retornada.

Obtenção da melhor jogada

Para a obtenção da melhor jogada, o que o nosso algoritmo faz é, como está em **student.py**, simplesmente ordenar a lista de estados obtidos a partir da função **calc_states** por custo e retornar a última jogada associada ao primeiro estado dessa lista, sendo depois esta a enviada para o servidor.

Resultados

Em comparação à primeira entrega, os resultados foram muito melhores, devido a erros corrigidos. Também, a partir do momento em que o engine foi mudado para a implementação com bytes, notou-se uma melhoria.

Sabemos, no entanto, que com lookahead seria possível obter resultados melhores..