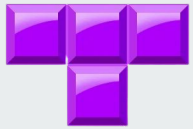
Tetris: AI agent



Licenciatura em Engenharia Informática

Inteligência Artificial
2021/2022

Diogo Cruz, 98595
João Reis, 98474
Ricardo Rodriguez, 98388
P6

universidade de aveiro

Classes utilizadas

No s*tudent.py* foram criadas 3 funções:

- **shapesKeys:** Esta leva como argumento todas as peças do jogo (alcançado através de *import de **SHAPES*** no *shapes.py)* e a informação inicial do jogo. Com estas, ele para cada shape e cada possível rotação da mesma vai guardar num dicionário todas as possíveis combinações de teclas premidas e válidas no contexto do jogo
- **findShape:** Leva como argumento a peça que é lida inicialmente que se encontra na forma de coordenadas individuais de cada bloco existente nela e encontra a forma partir dessas coordenadas.
- **agent_loop**: recebe e envia informação para o server de acordo com as informações recolhidas da função search da classe Search

No *search.py* foram criadas 2 classes:

- **Solution**: classe que define uma solução possível, é representada apenas pela sua forma, shape.
- **Search**: classe com várias funções que ajudam na descoberta da melhor jogada para a peça atual e com o lookahead implementado não só dessa mas das peças seguintes também

Algoritmo de pesquisa (1)

Para a resolução deste projeto, adotamos uma perspetiva **breadth-first** com uma verificação das próximas peças (*lookahead*). Inicialmente, quando a velocidade do jogo é lenta, o algoritmo procura pelas três próximas peças além da atual. No entanto, há medida que a velocidade do jogo aumenta e é necessário que o algoritmo seja mais rápido e eficiente, o lookahead vai sendo reduzido até ser igual a um.

O algoritmo funciona da seguinte maneira:

- Procura todas as soluções possíveis para a peça atual (verifica cada posição possível para cada rotação da peça, bem como as teclas correspondentes).
- Simula a colocação da peça com as teclas calculadas anteriormente, tendo em conta a disposição do jogo atual.

Algoritmo de pesquisa (2)

- As coordenadas da peça repousada são adicionadas à disposição do jogo anterior, guardamos a solução atual à lista de combinações, calculamos a heurística da solução atual (ver slide 6) e adicionamos a solução à lista *best_nodes*, que vai conter todos nós que foram expandidos a partir do nó pai (peça/solução anterior).
- Se a lista de soluções (que guarda um estado final específico) tem tamanho diferente que o número de peças que estamos a verificar (*lookahead*), expandimos apenas um número limitado (*max_nodes*) de soluções, tornando o algoritmo mais eficiente.
- Caso contrário, significa que esse nó não pode ser expandido para mais nenhuma solução. Calculamos a heurística final (soma das heurísticas de cada solução dessa combinação) e adicionamos a solução final à lista *self.best_nodes* (que contém todas as soluções finais).
- No final, a melhor solução é aquela que tem maior heurística. As teclas de cada solução desta são depois usadas no *student.py*.

Heurísticas

Para descobrir qual a melhor posição possível da peça.

- altura de cada coluna (*checkHeight*)
- os pontos obtidos (*checkScore*)
- o número de buracos (*checkHoles*)
- bumpiness (*checkBumpiness*)

A heurística de cada solução é dada pela seguinte fórmula:

checkHeight * **-0.510066 +** ***checkBumpiness*** * **-0.184483 +** ***checkHoles*** * **-0.35663 +** ***checkScore*** * **0.760666**

Conclusão

Os resultados obtidos não coincidiram com as nossas expetativas e ambições, dado que, implementado o lookahead, esperávamos obter uma média de pontuações mais elevada.

Isto deve-se ao facto da fórmula da heurística não se adequar, ou seja, adicionar mais heurísticas com diferentes pesos. Também pode dever-se a algum erro de implementação do lookahead.

Discutimos alguns problemas que nos foram aparecendo ao longo da implementação da nossa solução com o seguinte grupo:

- Alexandre Serras, 97505, P6
- Gonçalo Leal, 98008, P6

Referências

Tetris AI – The (Near) Perfect Bot | Code My Road

Tetris Agent Optimization Using Harmony Search Algorithm

Tetris Strategies in a Multiplayer Environment

Python Performance Tuning: 20 Simple Tips – Stackify