Heurísticas

De forma a que o agente escolha a melhor ação para uma determinada peça de acordo com o estado do jogo, é necessário ter uma função heurística que calcule a melhor posição para as peças.

Decidimos que a função heurística seria composta por 4 funções elementares:

- **aggregate_height()** -> Calcula a altura de cada coluna do estado do jogo e retorna a soma de todas as colunas.
- **holes()** -> Calcula o número de buracos que o estado do jogo apresenta.
- **bumpiness()** -> Calcula a diferença entre a altura da coluna atual e a próxima coluna em módulo.
- **completed_lines()** -> Calcula o número de linhas completas presentes no estado do jogo.

A junção destas 4 funções resulta da função **heuristic()**, nesta função cada valor resultante das funções anteriores é multiplicado por valores específicos.

Implementação

- Primeiro começamos por identificar a peça e obtemos todas as ações possíveis da mesma, após isso, vamos calcular a heurística de cada ação, e escolher aquela que terá a heurística maior.

- Numa segunda implementação, inserimos a pesquisa de ações da peça atual com a próxima peça, recursivamente.

- De modo a resolver o problema de o agente “esperar” pela letra “I” e assim o jogo , decidimos utilizar a próxima peça e verificar se essa não fosse um “I”, jogar uma outra letra de modo a parar o aumento do fosso que estava a ser criado.

Peça Atual vs Próxima Peça

- Durante a implementação do agente, a utilizar a próxima peça, apercebemo-nos que por vezes a pesquisa era muito demorada, e assim o servidor não iria conseguir executar todas as ações na peça que deveria. Sendo assim, criamos uma condição para quando essa pesquisa é muito demorada, e assim em vez de utilizar a próxima peça, a pesquisa utilizará somente a peça atual, diminuindo em larga escala o tempo de execução da pesquisa, e assegurando que as ações não serão perdidas nem utilizadas em peças posteriores.

Conclusão

- Se estivermos somente a analisar a peça atual, a pesquisa por todas as ações dessa peça tem um tempo de execução minimamente aceitável, visto que não são perdidas ações, no entanto quando começamos a aumentar o número de ações possíveis (com a adição da próxima peça), os tempos de execução aumentam relativamente e prejudicam as ações a executar se estes forem demasiado altos em comparação com a taxa de refresh do jogo.

Fontes

https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/

https://medium.com/python-pandemonium/building-a-tetris-bot-part-1-the-stupid-bot-2cbc38d6e32b

https://levelup.gitconnected.com/tetris-ai-in-python-bd194d6326ae