Agente Autónomo - Tetris

Inteligência Artificial

Alexandre Serras - 97505
Gonçalo Leal - 98008

Classes e Ficheiros

Para resolver o problema proposto desenvolvemos várias classes:

- Solver - entidade responsável por encontrar e avaliar todas as jogadas de uma peça e devolver a melhor (1.ª entrega)
- Solver2 - entidade responsável por encontrar e avaliar todas as jogadas de uma peça e devolver a melhor (2.ª entrega)
- MyShape - Classe que representa as formas e nos dá alguma facilidade para as manipularmos
- NextSolution - entidade responsável por encontrar todas as jogadas possíveis da próxima peça a ser jogada

Para além das classes, existem alguns ficheiros importantes:

- student.py - ficheiro a ser executado
- runner.sh - script bash que corre 10 tetris em simultâneo (10 servidores, 10 viewers e 10 students)
- runner2.sh - script que corre 10 vezes o mesmo tetris (um servidor e um viewer para 10 students)

Algoritmo de Pesquisa

O nosso algoritmo de pesquisa não mudou muito de conceito entre a primeira e a segunda iterações. Recebemos a peça e tentamos colocá-la o máximo à esquerda. Calculamos todas as jogadas da peça e avaliamos o valor das heurísticas para cada uma delas. Este processo repete-se para cada uma das rotações da peça. No fim o algoritmo devolve o conjunto de teclas que representam a jogada com melhor score.

Entre a primeira e a segunda iteração o código foi bastante trabalhado e melhorado em termos de eficiência reduzindo o número de ciclos, eliminando jogadas repetidas e reduzindo o número de ciclos nas funções das heurísticas através do uso de list comprehensions, sets e strings.

A maior mudança da primeira para a segunda entrega é que agora calculamos as jogadas da próxima peça rotação a rotação previamente e só depois avaliamos o seu score. Cada vez que uma key é enviada para o servidor são calculadas as jogadas referentes a uma rotação específica da peça. No fim, quando a peça atual é pousada, calculamos o score de todas as jogadas e escolhemos a melhor.

Heurísticas

- Holes - Conta todos os buracos que existem no jogo, têm um peso de -0,32663 quando o current height é inferior a 20, quando é maior ou igual a 20 o peso passa para -0,35663
- Aggregate height - Faz o somatório da altura de todas as colunas , têm um peso de -0,550066 quando o current height é inferior a 20, quando é maior ou igual a 20 o peso passa para -0,710066
- Current height - Indica qual das colunas do jogo tem a maior altura
- Completed lines - Indica quantas linhas vão ser feitas com a jogada que está a ser calculada, têm um peso de 0,660666 quando o current height é inferior a 20, quando é maior ou igual a 20 o peso passa para 4
- Bumpiness - É um somatório que faz a diferença entre a coluna atual e a coluna anterior, logo começa a contar-se na segunda coluna, têm um peso de -0,204483 quando o current height é inferior a 20, quando é maior ou igual a 20 o peso passa para -0,284483

Com a fórmula abaixo calcula-se a pontuação da jogada que o agente está a calcular, para cada peça o agente escolhe a jogada com o maior score.

```
score = (aggregate_height * aggregate_height_weight + current_height * current_height_weight +
       holes * holes_weight + completed_lines * completed_lines_weight + bumpiness * bumpiness_weight)
```

Resultados:

Os resultados obtidos neste versão final do trabalho foram muito melhores do que aqueles que tínhamos na primeira entrega, onde passamos de uma média de 150 pontos para uma média de 350 pontos. Para além disso, conseguimos obter várias vezes pontuações acima dos 900 pontos.

Contudo, consideramos que os resultados podiam ser bastante melhores. Durante a primeira entrega, que foi onde tivemos mais tempo para desenvolver o projeto, perdemos bastante tempo a tentar perceber que erro existia e apercebemo-nos demasiado tarde que era uma questão de eficiência que estava a prejudicar bastante a performance do nosso agente e tivemos que fazer uma re-estruturação do problema.

Outro aspeto que pensávamos que iria melhorar bastante a performance do nosso agente era colocar o mesmo a fazer logo os cálculos da próxima peça quando estava a enviar uma key para o servidor contudo não notamos uma melhoria muito significativa.

Por final, a implementação do lookahead de 1 peça fez com que o agente ficasse muito lento e apenas prejudicou a performance do agente.

Conclusão

O resultado do nosso projeto acabou com um sabor agridoce, onde conseguimos ter melhorias significativas entre as entregas , contudo temos noção que o resultado final não traduz o esforço que foi efetuado.

O grupo teve alguns problemas a conseguir perceber que muitos dos problemas que o agente tinha não era relacionado com a inteligência do mesmo mas sim com a eficiência deste, visto que até esta altura do curso ainda não tínhamos sido penalizados por questões de eficiência e daqui em diante é mais um fator que vamos ter que ter em conta nos nossos trabalhos.

Alunos com os quais debatemos o projeto:

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

Todos estes alunos pertencem ao mesmo grupo