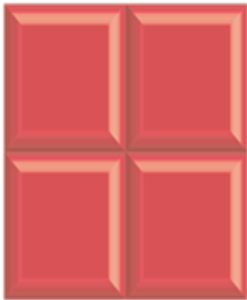


# TETRIS

Inteligência Artificial

DETI, Universidade de Aveiro

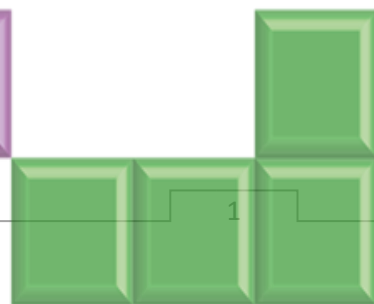
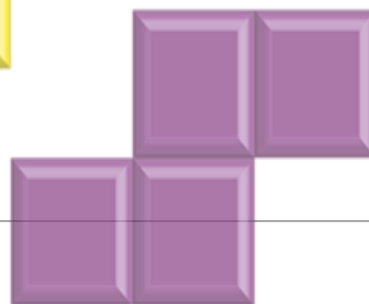
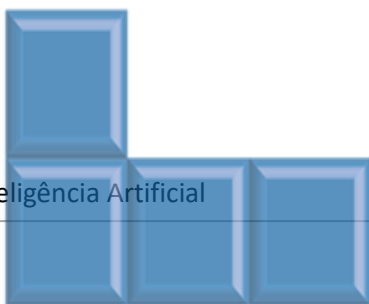
Dezembro, 2021



Frederico Vieira, nmec: 98518

Ricardo Antunes, nmec:98275

Tiago Coelho, nmec:98385



## Introdução

O trabalho prático de grupo proposto na cadeira de Inteligência Artificial (IA) tem como objetivo desenvolver um agente capaz de jogar de forma inteligente o jogo do Tetris.

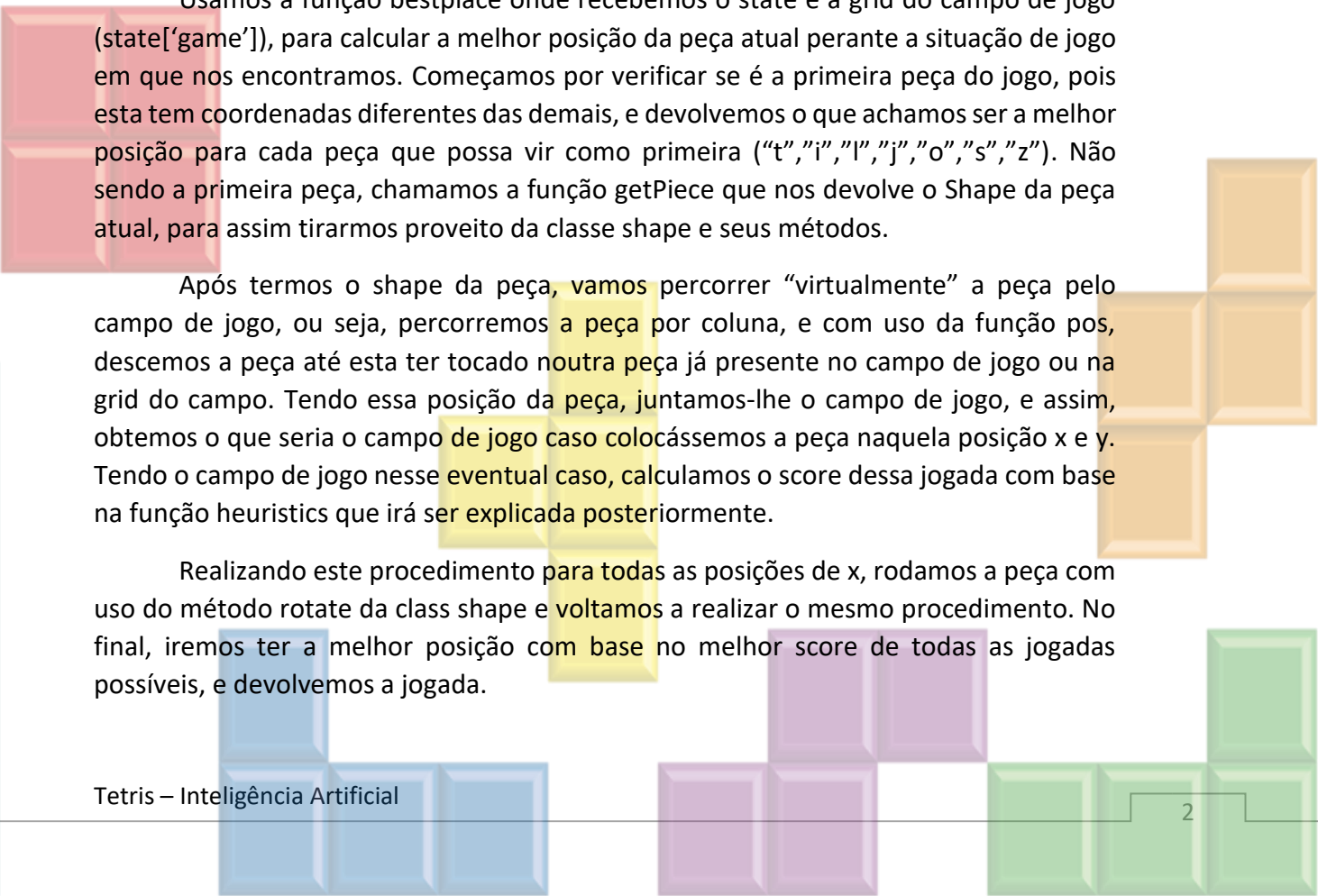
Tetris consiste num jogo de completar linhas movendo peças de formas diferentes que descem para o campo de jogo. Quando completamos as linhas, as mesmas desaparecem e recebemos pontos. O jogo prossegue assim até uma peça no campo atingir o topo. Quanto mais tempo passarmos sem atingirmos esse top, maior será a pontuação. Conforme o nível do jogo aumenta, a velocidade consequentemente também aumenta, aumentando a dificuldade, diminuindo o tempo de resposta por parte do jogador.

A forma que encontrámos para criar o agente foi basearmo-nos no jogo, criando diversas funções de maneira que o agente consiga assim escolher a melhor posição para colocar cada peça. Para tal criamos uma heurística, onde para cada jogada damos um score, que ajuda assim o agente a poder decidir qual a melhor jogada para cada peça.

## Descrição do algoritmo e sua implementação

Todo nosso código foi desenvolvido unicamente no `student.py`, desde movermos a peça, escolhermos a melhor posição, calcularmos a heurística e suas devidas funções para tal.

### 1. Melhor posição



Usamos a função `bestplace` onde recebemos o `state` e a `grid` do campo de jogo (`state['game']`), para calcular a melhor posição da peça atual perante a situação de jogo em que nos encontramos. Começamos por verificar se é a primeira peça do jogo, pois esta tem coordenadas diferentes das demais, e devolvemos o que achamos ser a melhor posição para cada peça que possa vir como primeira ("t", "i", "l", "j", "o", "s", "z"). Não sendo a primeira peça, chamamos a função `getPiece` que nos devolve o `Shape` da peça atual, para assim tirarmos proveito da classe `shape` e seus métodos.

Após termos o `shape` da peça, vamos percorrer "virtualmente" a peça pelo campo de jogo, ou seja, percorremos a peça por coluna, e com uso da função `pos`, descemos a peça até esta ter tocado noutra peça já presente no campo de jogo ou na `grid` do campo. Tendo essa posição da peça, juntamos-lhe o campo de jogo, e assim, obtemos o que seria o campo de jogo caso colocássemos a peça naquela posição `x` e `y`. Tendo o campo de jogo nesse eventual caso, calculamos o `score` dessa jogada com base na função `heuristics` que irá ser explicada posteriormente.

Realizando este procedimento para todas as posições de `x`, rodamos a peça com uso do método `rotate` da `class shape` e voltamos a realizar o mesmo procedimento. No final, iremos ter a melhor posição com base no melhor `score` de todas as jogadas possíveis, e devolvemos a jogada.

## 2. Heurística

**AggregateHeight:** Esta função diz-nos o quão alto está o campo de jogo, calculando a altura do bloco mais alto de cada coluna da grelha. O objetivo é minimizar este valor, uma vez que isso significa que temos o jogo com uma altura baixa, podendo assim colocar mais peças e fazer mais pontos antes de chegar ao topo.

**CompleteLines:** Devolve-nos o número de linhas completas que a jogada irá completar. Como este é o aspeto mais importante, pois é que nos dá pontuação, queremos maximizar este valor.

**Bumpiness:** Função que devolve as variações de altura entre colunas, somando a diferença entre colunas adjacentes em valor absoluto. O propósito disto é manter as colunas no mesmo nível, evitando “poços”, ou seja, que uma ou mais colunas estejam muito mais baixas que as restantes. Sendo isto importante para manter um bom campo de jogo, iremos minimizar este valor, por forma a evitar “poços”.

**TotalHoles:** Buracos no campo de jogo é mau, mas sempre irá haver conforme o jogo desenvolve. Assim, evitá-los é o nosso objetivo. Portanto, esta função conta o número de buracos por coluna, considerando buraco cada bloco vazio que contenha um bloco de peça por cima, não sendo possível de colocar peça. Como meta queremos minimizar este valor.

Depois das 4 heurísticas calculadas, criamos uma combinação linear com 4 constantes, testadas e calculadas por nós, conforme o projeto avançava, ou seja, descobríamos novas constantes e adaptávamos conforme a prestação do agente com a finalidade de o aprimorar. Baseámos também as constantes na altura do campo de jogo, variando as mesmas para alterar a escolha de melhor posição para o agente, focando-se menos nas linhas com um jogo baixo, e mais em não fazer buracos, de maneira a fazermos mais linhas de uma só vez, e focando nas linhas conforme a altura do campo de jogo subia.

## 3. Mover a peça

Para criarmos os movimentos da peça, usamos a função `move_piece`, que com foco na função `bestplace`, onde nos é devolvida a melhor posição, criamos uma string com as teclas devidas para a peça poder ser movida para o local correto.

## 4. Agente

Assim o agente chama a função `move_piece`, que consequentemente chama a função `bestplace`, onde irá realizar todos os cálculos e devolver a melhor posição. De seguida, a função `move_piece` devolve a string com as teclas, o agente recebe e envia peça a peça para o servidor, e vai-se atualizando usando o `loads`.

## Resultados

Como se pode ver na imagem, o nosso agente faz em média, no nosso dispositivo, 300 pontos. No entanto, podemos realçar o facto de o nosso agente ter sido testado numa máquina virtual com 2 Gb Ram e com 500 Mb espaço livre, portanto é possível que noutros computadores possa ter uma melhor performance devido a um maior poder computacional. Este foi o melhor agente que conseguimos criar utilizando uma heurística onde é usada uma estratégia dependente da peça atual, de maneira a tentar ao máximo que o agente permaneça na parte inferior do jogo. Testámos o agente com lookahead, mas o mesmo tinha uma grande perda de performance e não alcançava grandes pontuações.



## Conclusão

A realização deste trabalho prático permitiu aprofundar os conhecimentos sobre como se deve comportar um agente inteligente, que é aquele que adota a melhor ação possível diante de uma situação. Ao longo da sua realização foram discutidas diversas linhas de pensamento sobre como o agente deveria ou não se comportar, funções a implementar, métodos de realização de certas partes do código, de maneira a podermos melhorar o mesmo diminuindo assim a sua complexidade.

Durante o trabalho foram feitas diversas pesquisas, tanto para código como para sabermos mais sobre o jogo, de forma a podermos compreender melhor e descobrir as melhores jogadas, para assim criarmos a nossa heurística e seus valores. Também houve bastante pesquisa no âmbito do agente por forma a sabermos mais e chegarmos a conclusão de como implementar o nosso código para criarmos um agente “bom” no jogo do Tetris.

Em termos de resultados, podemos referir que atingimos uma boa pontuação, mas que poderíamos ter melhorado, adicionando as próximas peças, fazendo com que o agente jogasse de forma a pensar nas próximas, melhorando assim cada jogada do mesmo.

Em suma, concluímos que o presente trabalho enriqueceu de forma positiva o nosso leque de conhecimentos sobre como implementar um agente inteligente, tornando-se assim um trabalho interessante pois não tínhamos muito conhecimento neste tema.

