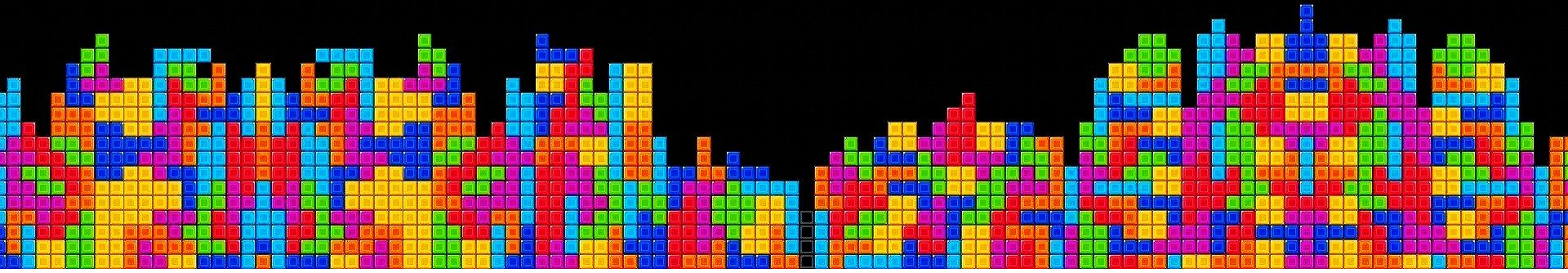
Tetris AI

Alexandre Pinto 98401
Pedro Jorge 98418
Luís Martins 98521



Funcionamento do Programa

Ao longo do projeto foram implementadas várias funções, todas desempenhando papéis diferentes e importantes no programa:

- ***identifyPiece() -*** Esta função permite que o programa seja capaz de identificar que peça vai ser usada. Para tal, criou-se várias condições “if”, sendo que em cada uma delas definiu-se as coordenadas *x* e *y* de cada peça.
- ***numberRotations() -*** Esta função permite saber quantas rotações cada peça pode fazer. A implementação passa por, novamente, criar condições “if”, onde em cada uma delas definimos o número de rotações que cada peça pode fazer. De notar que definimos o número de rotações como sendo a quantidade de vezes que uma peça roda até atingir o seu estado “inicial”, e a esse valor subtraímos 1.
- ***pos() -*** Esta função retorna a adição das coordenadas da peça no jogo ao jogo. Permite saber se está alguma peça presente no final da grelha ou não. Caso esteja, as coordenadas da nova peça vão ficar por cima das que já lá estão no jogo, caso contrário, a nova peça assume as coordenadas mais próximas da base da grelha.
- ***aggregateHeight() -*** Esta função permite saber a soma total das alturas. Para tal, criou-se um ciclo “for” que percorre o número de colunas e outro ciclo for que vai percorrer o “virtualgame”. Caso a coordenada “x” esteja dentro dos valores do primeiro ciclo for (ou seja, é coluna) e a coordenada “y” seja menor que 30 (ou seja, é linha), então é porque é o valor máximo da coluna.
- ***completeLines() -*** Esta função permite saber quantas linhas é que estão preenchidas. Para tal, primeiro vamos verificar se existem colunas para cada linha e somá-las, e, caso a soma das colunas seja 8 então é porque existe linha.
- ***countHoles() -*** Esta função permite saber quantos buracos é que estão disponíveis. Um buraco é definido como um espaço vazio de forma a que haja pelo menos um quadradinho na mesma coluna acima dele.
- ***getBumpiness() -*** Esta função permite saber a “bumpiness” da grelha. A “bumpiness” está relacionada com a variação da altura das colunas e é calculada através da soma da diferença absoluta de todas as colunas adjacentes. Para tal, inicializou-se 2 variáveis que vão funcionar como sendo 2 colunas adjacentes. Se seguida iniciamos 2 condições “if” que vão verificar se são adjacentes e obter a coordenada y de cada coluna, que vai funcionar como a altura de cada coluna. De seguida fazemos a diferença absoluta entre esses 2 valores e obtemos a “bumpiness”.

Funcionamento do Programa

- ***solve() -*** Esta é das funções mais importantes do programa visto que vai retornar todas as pontuações de todas as posições e rotações da peça na grelha. Nesta função, vamos identificar a peça e o número de rotação que ela pode fazer. Para cada rotação da peça, vamos criar um jogo virtual e tentar encaixar a peça em todas as posições possíveis da esquerda para a direita da grelha. Obtemos o score de todos os jogos virtuais e adicionamos um tuplo com esta pontuação e um dicionário com a posição e rotação a uma lista “totalScores”. Esta função retorna a lista com esses tuplos.
- ***bestPosition() -*** Esta função permite descobrir a melhor posição para a peça ir. Para tal, iniciou-se um ciclo “for” que vai percorer a lista dos “totalScores”, onde estão guardados todos os scores, e guardar na variável “bestScore” o melhor score (mais alto). De seguida obteve-se a melhor posição através do valor “x”, que está presente na segunda posição de cada lista guardada na lista “totalScores”.
- ***bestRotation() -*** Esta função permite descobrir qual a melhor rotação a realizar. A implementação é quase igual à função anterior sendo que a única diferença é que em vez de irmos buscar o valor de “x” vamos buscar o valor do “rotate”.
- ***compareX() -*** Esta função vai comparar a menor coordenada “x” da peça com a “bestPosition”, ou seja, permite-nos saber quanto é que temos de andar para cada lado para encaixar a peça (“shift”).

Heurística

A heurística utilizada teve por base os seguintes parâmetros:

- Soma das Alturas (aggregateHeight que retorna aggregate) - Diz-nos o quão alta é a grelha;
- Linhas Completas (completeLines que retorna lines) - diz-nos o número de linhas que ficam preenchidas;
- Nº total de buracos (countHoles que retorna numberHoles) - diz-nos o número de espaços vazios entre peças ao longo da grelha;
- Diferença entre colunas (bumpiness que retorna bumpiness) - diz-nos o agregado das diferenças entre todas as colunas da grelha;

Para cada um dos parâmetros atribuímos diferentes pesos. Como o objetivo do Tetris é fazer linhas para que a coluna de peças não toque no topo da grelha, atribuímos um peso maior a esse parâmetro, sendo o único com peso positivo.
Os pesos são -0.510066, 0.760666, -0.35663 e -0.184483, respetivamente.

Assim, a fórmula utilizada foi `score = a * aggregate + b * lines + c * numberHoles + d * bumpiness`

O score é calculado para todas as posições que uma peça poderá ter no jogo tanto para a orientação com que chega ao jogo como para as possíveis rotações que faz. É verificado qual o melhor score e utilizados os valores de posição e rotação correspondentes a essa pontuação. A melhor pontuação significa que é o melhor lugar para colocar a peça.

Enviar dados (teclas)

Para enviarmos as teclas corretas para o servidor temos de saber qual a posição correta e a rotação que a peça terá de fazer. Para isto necessitamos da função solve que vai retornar todas as pontuações de todas as posições e rotações da peça na grelha, assim como as suas pontuações.

De seguida, temos de ver qual a melhor posição e o número de rotações através das funções bestRotation e bestPosition que irá procurar pelo melhor score nos tuplos da lista e retornar as entradas (‘x’ e ‘rotation’) do dicionário correspondente.

Com isto, temos todas as informações para enviar as teclas.

Primeiramente, para efetuar a rotação, enviamos para o servidor a tecla ‘w’ o número de vezes da bestRotation. A peça fica com a rotação ideal para a podermos posicionar horizontalmente.

Após a rotação estar finalizada, utilizámos uma função compareX que vai comparar a menor coordenada x da peça com a bestPosition. Esta função retorna o valor shift que é o número de casas que temos de andar horizontalmente. Caso o shift seja negativo, teremos de carregar o número absoluto de shift vezes na tecla ‘a’ para que a peça vá para a esquerda. Em caso contrário, teremos de carregar shift vezes na tecla ‘d’ para que a peça vá para a direita. Utilizámos ciclos for para repetir o processo de envio. Por último, se shift for nulo, basta carregar em ‘s’ para que a peça caia nesse lugar.

Para enviar as teclas utilizamos o seguinte comando:

```
await websocket.send(
    json.dumps({"cmd": "key", "key": "w"})
)
```

Conclusão

Concluindo, do nosso ponto de vista o objetivo geral do trabalho foi conseguido. O programa funciona de forma autónoma, coloca as peças na melhor posição e é capaz de as rodar de forma a obter melhor posição para encaixar.

Claro que existem ainda algumas arestas por limar, nomeadamente, conseguir fazer mais pontos à medida que a velocidade aumenta, contudo, consideramos que fizemos um bom trabalho e que este projeto nos ajudou a desenvolver as nossas capacidades de programação de inteligência artificial e sem dúvida que nos deu mais bases para o futuro.

Percentagem de trabalho de cada elemento:

Alexandre Pinto:45%;

Pedro Jorge:45%;

Luís Martins:10%.