Jake stevens

IS53049A

33703705

Tetris AI Assignment

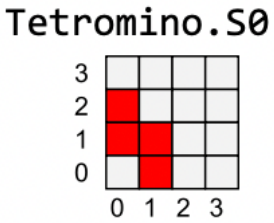
Inspiration and Implementation

To design an effective AI for this project, I attempted to express through code the thought processes that I would make myself playing Tetris. Basing my implementation off of utility theory, I created one long function that takes the move as an input and judges the decision in a variety of different ways. I am then using weights to prioritise certain strategies over another in a way that is effective and readable.

Sectors of the function

While working on the project, I encapsulated each sector of this function into separate functions. This allowed me to debug and address each segment as its own. Each sector will output a high score for a wrong decision. The first sector of the function outputs the move that will have the lowest column after the block is placed. This forces pieces to be packed as tightly to each other as they can while also staying away from the “danger zone” (the top portion of the board). This is an important aspect to the algorithm as it is both an offensive and defensive technique. The second sector assists this by preferring a piece that has itself been placed lower on the board, as opposed to the overall height of the board. For instance, if the piece will have no effect on the highest column as a previous piece has been placed at the top of the board.

The third sector of the function will output a higher number if more spaces on the pieces’ row are free. This attempts to prevent lines that cannot be cleared due to pockets of space as a result of poor placement. Surprisingly, this technique was not as successful as I had initially thought and is simply used to assist the first two sectors which take priority within the score system (the weights shown at the bottom of the writeup reflect the priority of each sector).



Poor shape to place on the left

The final part of the system individually ranks inappropriate moves for certain pieces. There are a number of pieces that should not be placed to the side of the grid as they will always live unwanted space to the side of the board. This is a small aspect to the algorithm as it does not reflect a game plan or general goal that the others do.

Weights

After creating 5 separate functions, I grouped them together into one long process and created 5 variables, calling them weights. These are assigned to the final result of each section and essentially means that certain ways of acting are measured more critically. This is reflective to real life, where players adapt to situations with a fixed goal always in mind. This is why techniques such as forbidding certain moves to pieces is not as effective as I had previously thought.

Shaping and Testing

To test and shape my AI, I measured the effectiveness of each process 10 times at standard speed. This allowed me to analyse each move and determine its effectiveness

depending on the height of the board. I did this by multiplying the processes weight and reporting my findings

I found this more effective than using the automation for testing as it allowed me to see the mistakes in real-time, so I can attempt to rectify them.. When testing the results of each formula, I am including the other processes, I am greatly increasing the weight of that technique. This can be shown as the columnWeight formula is given in the brief, yet combined with the other techniques, has yielded far greater scores.

ColumnWeight: 114, 150, 87, 137, 111, 177, 126, 147, 59, 217, 197
Average Score: 122
By increasing this weight to 50, I was met with much higher scores. All other weights were set to 1. This technique is placing the block at the lowest height. Very effective at clearing blocks.

Low pos weight: 151, 95, 78, 128, 78, 120, 128, 95, 81, 82.
Average Score: 104
This technique seems to be more effective in early game / mid game scenarios but is very ineffective when the pieces reach the top of the board

Cell Weight: 121, 72, 63, 87, 62, 53, 72, 71, 77, 57.
Average Score: 73
Poor effectiveness, seems to place blocks at random opportunities. Often missing one or two block per row but filling the row almost entirely. Here is an example. This move is choosing a move that gives the least empty cells on the given row

Corner: 66, 54, 66, 52, 62, 71, 62, 53, 67, 66
Average score: 62
Places boxes and straights on the side. Resulting on full sides yet building up the height too quickly which ultimately results in a quicker failure.

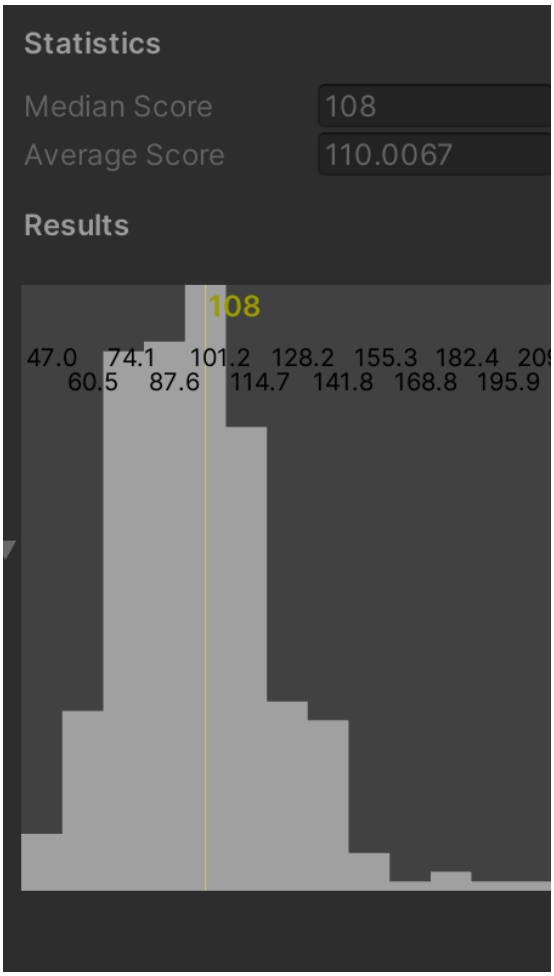
Bad Blocks: 53, 71, 89, 64, 76, 59, 71, 71, 104, 73
Average Score: 73
Playing defensively, poor at clearing blocks but doesn't seem to place boxes in wrong places.

Comparison:
This is the script included in the brief that we need to improve upon
Average Score: 86
75, 95, 126, 97, 56, 109, 87, 57, 89, 65.

After this I have implemented a small if statements that will adjust weights throughout the game, For instance I feel that certain strategies that work better in the mid game should have more weight when the game is being played in the middle of the board.

I have then used the average scores as a starting point to set the weights to. This has generated these results:
The results I have for this are: 98, 68, 87 ,81, 73, 97, 111, 109, 79, 138
Average Score: 94

This is an improvement from the comparison script but is far from the aim of 122 which is where the columnWeight variable at a far larger sum.



After changing weights and testing I have come up with this as the most suitable weight distribution

ColumnWeight = 50
LowPosWeight = 30 (depending on location on board)
cellWeight = 4
cornerWeight = 2
badShapeWeight = 2

Conclusion

I feel that my script is reflective of my own personal Tetris tactics, by placing blocks as close together as possible while attempting to remain low on the board and avoid stupid errors. All aspects of these techniques can be seen in the code and I feel that the code is structured in a way that makes it easy to visualise and is easy to modify.

In the future I would like to create a way to use machine learning techniques such as neural networks to experiment with the best combinations of weight distributions.