CS170 Project : Introduction + Methodology + Final Report
Tetris AI

Angel Gonzalez(agonz063@ucr.edu)
Marco Rubio(mrubi005@ucr.edu)
Isaac Lino(ilino001@ucr.edu)
Kevin Ortega(korte004@ucr.edu)

1. INTRODUCTION

1.1.1 Problem Space

Our project deals with a specific branch in game theory. This is a very big space that has produced much data and analysis. Specifically, our problem space will be on using search based algorithms along with heuristics to solve a specific gaming problem. This problem is the popular game of Tetris. The point of this game is to stack different shaped blocks, called Tetrominoes, on a 10 x 20 (size may vary) board while filling horizontal lines on the board with blocks thus deleting the line from the board and increasing the player's score. Our goal is to use heuristics to define the best place to set each block in order to maximize the amount of lines filled. These heuristics will take into account different factors of the game and combine them to come up with the best position for a new given block.

1.1.2 What is the problem? Why should anyone care?

The problem being addressed is the need for more data to a fundamental field that is growing rapidly. This project will hopefully go beyond the classroom in, not only teaching us the fundamentals of Artificial Intelligence, but perhaps also adding more data to a field that is ever growing and could be of much use in our lives moving forward. Thus, this project is important because of the impact the field it encompasses can have in this world; and adding more data to this field can only help it expand.

The A.I. field believes, it is important to think about *values,* how to introduce them into the machine and negotiate or analyze what has a greater importance above other things. Human values can be implemented by using a Artificial Intelligence Neural Network in reinforcement learning which is the strategy based on observation, in case there is a negative observation, it can be adjusted accordingly.

1.1.3 Expected deliverables

We analyzed a heuristics that can maximize the score in a Tetris game by maximizing the amount of lines filled within the board. We collected data and analyzed how effective the algorithm is in terms of its heuristics and time. We hope our data reflects the fact that the algorithm has minimal errors and is able to maximize the number of lines completed on the Tetris board.

1.2 RESOURCE PAPER#1

1.2.1 What was done

Many have studied the game of Tetris using search based algorithms. Our first resource comes from a Wordpress post written by Yiyuan Lee named “Tetris AI – The (Near) Perfect Bot”. In it he describes a program that can play and maximize the score in a Tetris game using heuristics. His approach is to use four different heuristics, each accounting for a different factor of the game. He then uses the results of each of these heuristics in order to calculate the best possible position for his next piece.

1.2.2 What we will do

This is the algorithm that we analyzed when running our tests. This paper includes a lot of information about the heuristics used and how they are combined. However, there is very little data about the algorithm besides the fact that it was able to complete 2,183,277 lines after running for about two weeks. Our analysis of this algorithm consists of running tests on these heuristics to find out how well each one performs. We wrote functions that will keep track of the average holes on a board, average height of the board, etc. This gives us a clearer view of how this algorithm performs and how efficient it is.

1.3 RESOURCE PAPER #2

1.3.1 What was done

Another paper that focused on using search based algorithms and heuristics to solve the game of Tetris comes from Max Bergmark in a paper titled, “Tetris: A Heuristic Study” http://www.diva-portal.org/smash/get/diva2:815662/FULLTEXT01.pdf. In this paper Bergmark studies another heuristic algorithm used to create the most amount of filled lines possible. This algorithm uses a breadth first search in order to choose the next best move to make. All moves are categorized by looking at the current piece and analyzing the different positions for that piece then giving the resulting board a score. This score is based on the number of holes on the board and the current height of the board.

1.3.2 What we will do

Since there is a lot of data on this paper, we use this data as a reference when conducting our own tests. In our project we take a similar approach to Bergmark in that we collect data on the various heuristics used. We then use the data gathered and analyze it to see how well the heuristics work and whether or not there is room for improvement. This paper was helpful to our project in that it gave us data and analysis technique that we used to conduct our own tests.

2 METHODOLOGY

The algorithm that we will be using for this project uses heuristics in order to fit each incoming Tetromino onto a Tetris board such that the amount of blocks that fill complete horizontal lines along the board are maximized.

1.1 First Heuristic

The first heuristic in the algorithm measures the height of each of the columns of the resulting board and adds them together. The algorithm takes into account all the possible positions of the given Tetromino piece and with that, uses the heuristic to measure the heights of each of the resulting boards and outputs each of these heights.

2.2 Second Heuristic

The second heuristic looks at all the vertical lines filled for each of the resulting boards and outputs this number for all boards. Since our goal is to maximize the amount of vertical lines filled, We will use this heuristic to accomplish that specific goal.

2.3 Third Heuristic

The third heuristic looks at the number of holes in each resulting board. If there is a hole in the board (meaning there is an empty space in the board with at least one block above it), the row which contains that hole can no longer be filled. Therefore, the number of holes must be minimized.

2.4 Fourth Heuristic

This heuristic calculates how each column of the resulting board differs in height. This is calculated by taking the absolute difference between one column and the column directly next to it and then adding these values. We want this value to be at a minimum to keep our board as level as possible.

2.5 Combining the four heuristics

The results of these four heuristics are then combined by adding each of these results together after each result is multiplied by a constant. After adding them, the board which results in the highest score is chosen. This resulting score is the cost function of each board. The constant that is multiplied by each of the results is calculated using an equation that weighs each constant by the value that it will be multiplied by. For example, since we want the number of holes to be minimized, its constant will be negative but since we want filled vertical lines to be maximized, this constant will be positive.

2.6 Improving the algorithm

Five functions are implemented in the algorithm in order to help it improve as it is running. The Fitness Function tests a specific board to see how many lines it has cleared to get an indication of how well the algorithm is running. The Tournament Selection selects some of the best performing boards and sends them off to be combined with other boards. The Weighted Average Crossover combines the constants for two boards to produce a set of combined constants. The Mutation Operator selects some of the offsprings produced by the Weighted Average Crossover (each offspring has a 5% chance of mutating) and changes their constants by ± 0.2. Finally, the Delete-N-Last Replacement deletes some of the offsprings that are performing the worst (have the lowest scores) and replaces them with new offsprings that are created.

2.7 Testing and Collecting Data

As discussed earlier, this paper includes very little to no testing on the algorithm, so our job was to conduct these tests ourselves. We wrote functions that test the effectiveness of the algorithms and heuristics in many ways. We test each of the heuristics individually, that is, we only let the program see one of the heuristics and use it as the cost function. We then graph the effect that this has on all the aspects of the game versus time. These functions gave us a more accurate picture of how this algorithm runs and where, if anywhere, there can be improvements.

3 FORMULAS AND DATA

3.1 Vector Formula

Using a parameter set with (A, B, C, D) can be represented as a vector in $\mathbb{R}^4$, A standard Genetic Algorithm for real-valued genes would involve searching the entire solution space ($\mathbb{R}^4$) for an optimal set of parameters. In this project, however, we need only consider points on the unit 3-sphere (i.e. 4-dimensional unit sphere). This is because the score function defined above is a linear functional and the comparison outcomes are invariant under scaling (of the score function).
To understand this from a more mathematical point of view, first rewrite the score function in the context of vectors as $f(\vec{x}) = \vec{p} \cdot \vec{x}$.

Let:

$$\vec{p} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}, \vec{x} = \begin{pmatrix} \text{Aggregate Height} \\ \text{Complete Lines} \\ \text{Holes} \\ \text{Bumpiness} \end{pmatrix}$$

Suppose we want to compare between two moves, move 1 and move 2, to decide which move is “better” (i.e. gives a better score). Suppose also that move 1 and 2 produces values $\vec{x_1}$ and $\vec{x_2}$ respectively. To check if move 1 is better than move 2, we need to check if $f(\vec{x_1}) > f(\vec{x_2}) \iff f(\vec{x_1}) - f(\vec{x_2}) > 0$. If this condition fails then it is trivial to conclude that move 2 is better or equally good as compared to move 1.

Now,

$$f(\vec{x_1}) - f(\vec{x_2}) = \vec{p} \cdot \vec{x_1} - \vec{p} \cdot \vec{x_2} = \vec{p} \cdot (\vec{x_1} - \vec{x_2})$$

Suppose $\vec{p} \cdot (\vec{x_1} - \vec{x_2})$ is either positive or negative. The better move would then be move 1 and move 2 respectively. Here, we can see how the comparison outcome is invariant under scaling of the parameters – we can multiply $\vec{p}$ by any positive real (scalar) constant and the sign of the above difference will remain the same (although the magnitude differs – but that is irrelevant), in which case the decision does not differ!

All parameter vectors in the same direction produce equivalent results. As such, it is sufficient to only consider a single vector for each direction. This justifies the restriction of the search to only points on the surface of the unit 3-sphere, as the surface of the sphere covers all directions possible.

We'll now move on to define the fitness function, selection procedure, crossover operator, mutation operator and recombination procedure used for tuning the AI.

3.2 Weighted Average

For the average, the two parent vectors $\vec{p_1}, \vec{p_2}$ has to be combined in vector form to get the average. The unit result vector $\hat{\vec{p'}}$ can then be produced where $\vec{p'} = \vec{p_1} \cdot fitness(\vec{p_1}) + \vec{p_2} \cdot fitness(\vec{p_2})$

Resulting vector is in the two main vectors on the unit in $\mathbb{R}^4$, it has more information towards main vector by the weight amount that is increased by both reflecting an uneven weight.

4 RESULTS AND ANALYSIS

4.1. How the Space is Scanned

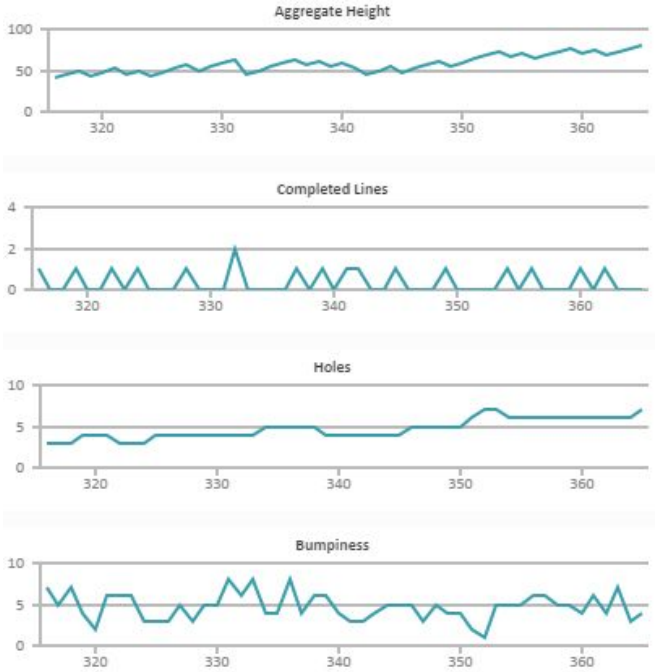
The code has a function that when given a piece, which includes its position and orientation, it places the piece in every possible orientation and every possible position and calculates the score of each. Based on the score the algorithm then places the piece in the location with the best score.

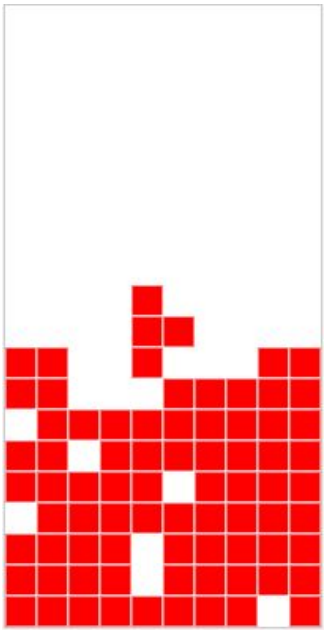
4.2 How the gain values were calculated

Given the four heuristics aggregate height, complete lines, holes, and bumpiness you will not get the best results as some of these heuristics are more important than others. The original programmer used genetic programming to find the best gains which resulted to be 0.51, 0.76, 0.35, 0.18 respectively. Below we show what would happen if we simply had one heuristic running at a time and what kind of results they would give.

4.3 Using all heuristics

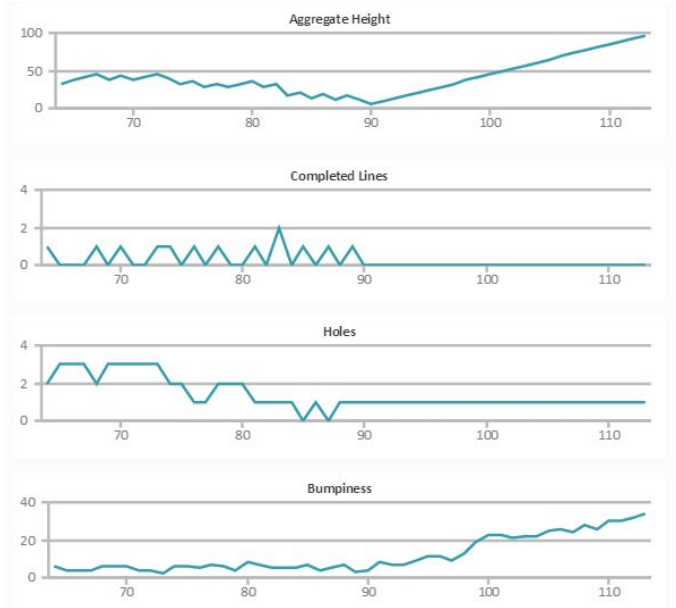
These graphs plot each heuristic by time. In these graphs all heuristics are activated, the image showing the board shows the current state of the Tetris board. In the following subsections, only one heuristic will be activated at a time so we can analyze each one.

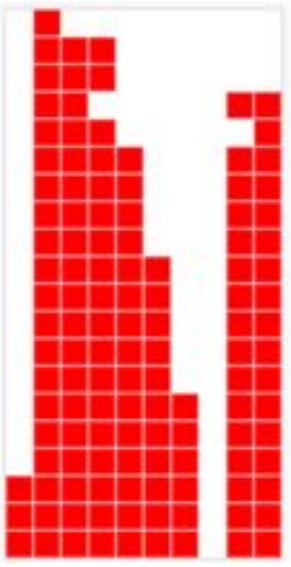




4.4 Aggregate Height Only

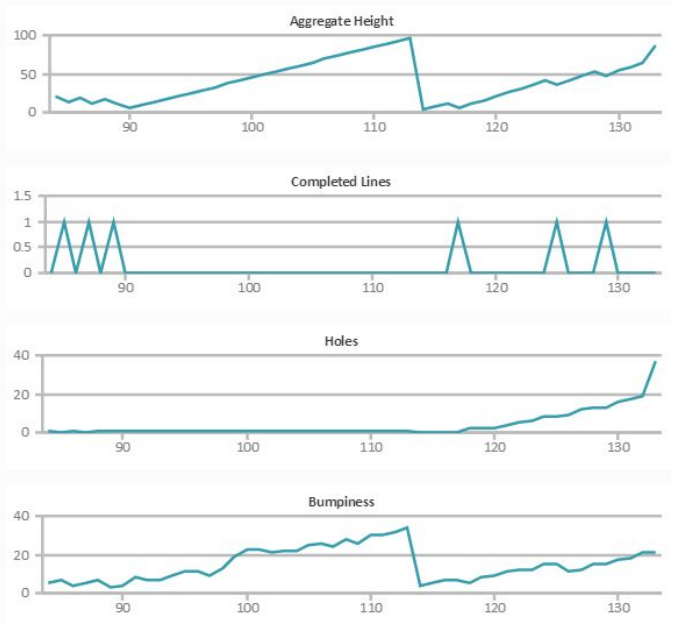
Aggregate Height Only Results in the aggregate Height slowly increasing as the algorithm tries to place the piece in the best location. Without the other heuristics taken into consideration multiple positions will return the same aggregate height so the algorithm picks the first choice. This is not always the most optimal choice. As a result of using only the Aggregate Height heuristic the program takes no other aspect of the game into account therefore resulting in a lost game.

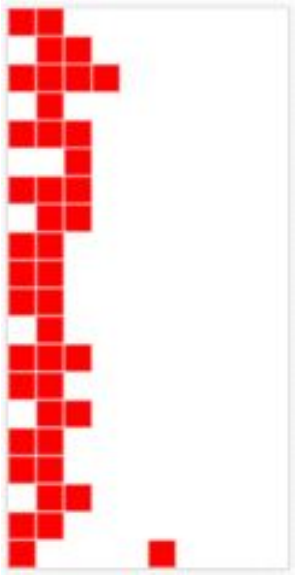




4.5 Complete Lines Only

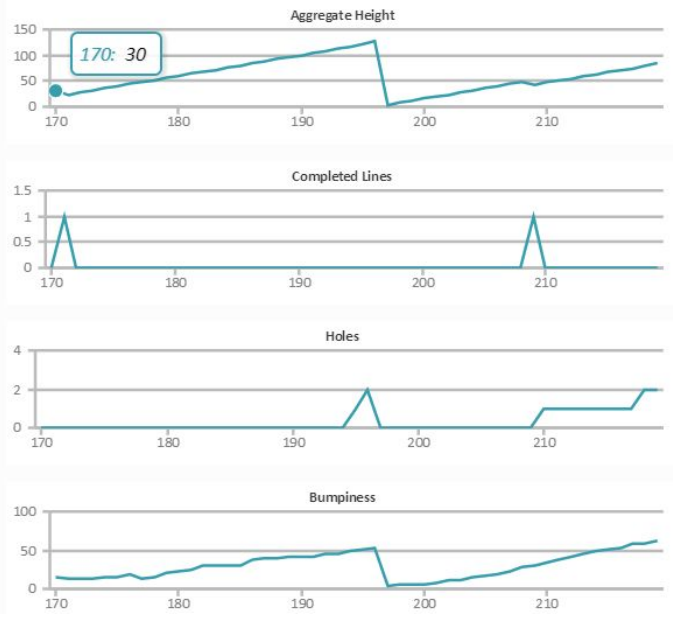
In an attempt to maximize the number of complete lines the algorithm would opt to simply picking the first choice that would give it a complete line. If you've played Tetris before you would know that just trying to complete a line at any height is not the best choice. Usually after completing a line it's sometimes impossible to complete a second line. The algorithm does not rotate the incoming piece and places it in the first position it sees creating a large stack and ending the game rapidly.

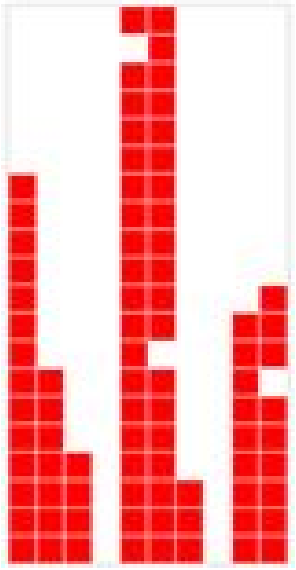




4.6 Holes Only

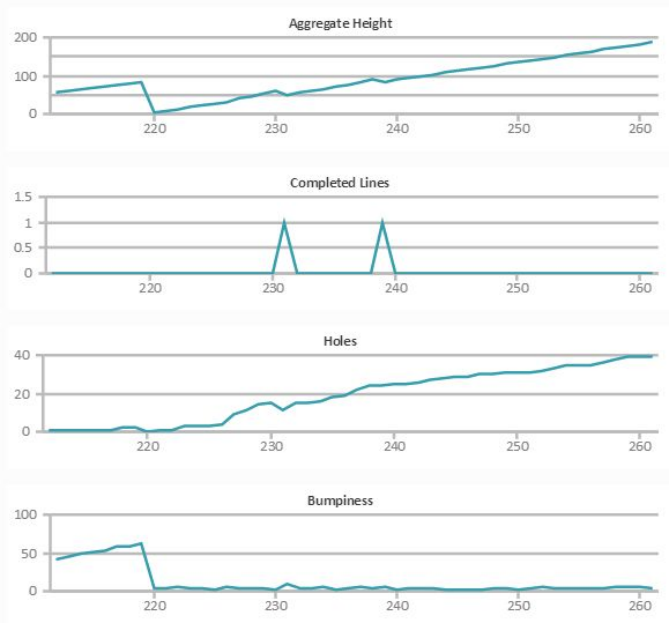
The algorithm that is optimized to avoid having holes, or empty spaces on the board, simply created towers to try and avoid creating holes. Since the Holes heuristic only worries about empty spaces that are *surrounded by other blocks*, it ignores the empty space that is not surrounded by blocks. This results in the creation of "wells" which causes the *bumpiness* and height to reach the top and end the game.

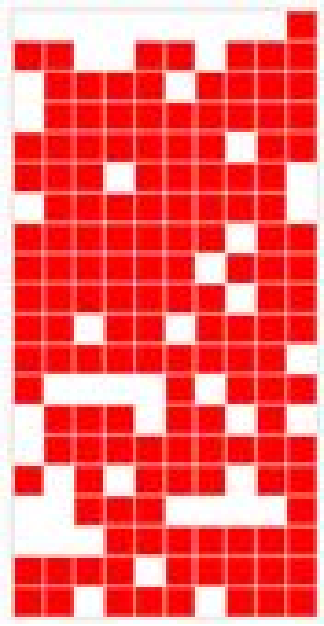




4.7 Bumpiness Only

The bumpiness heuristic lasted the longest of the others and filled the space best. This is because the program made its choice on where to place the next piece by looking at which position would allow the pieces on the board to remain as flat as possible. Without considering holes, it was unable to create a lot of successful lines so it simply filled the space available.
This makes you wonder how the algorithm would do with only the bumpiness and hole heuristics. Which we did in fact test, and giving the Bumpiness and Holes arbitrary weights of 0 and 5, respectively, we were able to get the program to run without fail for as long as we tested it (2 hours).





5 CONCLUSION

In conclusion, this algorithm never failed using the heuristics and gains provided by the original programmer no matter how much we tested it. The reason these heuristics work so well *together* is because each heuristic takes into account a specific aspect of the game, and when combined, they are able to combine each of these aspects resulting in a near perfect placement of the next piece. The tests that we ran hopefully accentuate this point, showing that no one heuristic is good enough to have the program run for a long time without failure. The next level of development would be to create an algorithm that outperforms this one. A possible improvement would be to decrease the number of holes even further to where holes are practically eliminated from the board entirely. We encourage those who are interested in this problem space to further test this algorithm and possibly improve upon it.

Source Code Link:
https://github.com/Tetris-Artificial-Intelligence/Tetris-Artificial-Intelligence.github.io

Report By: Angel Gonzalez, Isaac Lino, Marco Rubio
Functions written By: Marco Rubio
HTML and GitHub By: Isaac Lino