**Tetris Report**

The following report details the implementation of a Tetris AI and describes the various heuristics utilized to optimize its performance. It will include the working process and techniques implemented to create the Tetris AI. As well as this will be a critical reflection on what worked well and what did not, and discuss the limitations of the approach

It was done by using a series of heuristics. Each of these heuristics outputted a normalized value between 1 and -1, and were then given a weight. These weights were totaled for each simulated move and the one that provided the lowest total score was used.

The various heuristics that were used, were based off different ways that could quantify the game going well. These were sourced from research (Bergmark, M. 2015), as well as intuition. The key was to search ways that could quantifiably represent the Tetris blocks as data in some form to represent the status of the game for each move. Below are the heuristics that were used.

**Description of Solution**

The Tetris AI utilizes 11 heuristics to evaluate potential moves:

1. Max Height: This heuristic measures the maximum height of the Tetris board after a potential move is made.
2. Bumpiness: This heuristic measures the sum of the absolute differences between the heights of the columns on the Tetris board.
3. Complete Lines: This heuristic counts the number of complete lines on the Tetris board after a potential move is made.
4. Complete 4 Lines in a Row: This heuristic counts the number of rows of 4 or more lines that are completed after a potential move is made.

5. Closed Hole Number: This heuristic counts the number of "holes" on the Tetris board, which are defined as ‘An empty cell under the topmost filled cell in the same column’. (Bergmark, 2015)

6. Open Hole Number: This heuristic counts the number of "holes" on the Tetris board, which are defined as an empty cell in between 2 columns where the topmost cell of adjacent columns is at the same height or above. (Bergmark, 2015)

1. Well Number: This heuristic counts the number of "wells" on the Tetris board, which are defined as empty spaces that are surrounded by blocks on three sides.
2. Well Number (Deep): This heuristic is similar to the well number heuristic, but it also takes into account the depth of the wells.
3. Hole Number (Deep): This heuristic is similar to the hole number heuristic, but it also takes into account the depth of the holes.
4. Height Difference: This heuristic measures the difference between the highest and lowest columns on the Tetris board.
5. Top Heaviness: This heuristic measures the sum of the heights of all of the columns on the Tetris board.
6. Standard Deviation of Heights: This heuristic measures the dispersion of the heights of the columns on the Tetris board.

To combine these heuristics and choose the best moves, there was a function called combineScore that takes in every available move and calls each heuristic on that move, using the simulateMove method to simulate the effect of the move on the Tetris board. Each heuristic returns a normalized value between 0 and 1, which is then multiplied by a weight and summed with the values from the other heuristics. The move with the lowest resulting sum is chosen as the best move.

Text

Description automatically generated

I then implemented an evolutionary algorithm that runs many simulations of the Tetris game in parallel, using a library to mutate the weights of the heuristics randomly and selecting the weights that performed the best in each simulation. These weights are then evolved through successive generations of Tetris plays, with the goal of finding the optimal combination of weights that leads to the highest score.

Initially, only 4 heuristics were used, which was bumpiness (2), a basic closed hole number method (5), max height (1) and complete lines (3) (Lee, 2013). This obtained a score of around 30,000.

Due to the nature of the evolutionary script, the more generations in principle equated to better scores, until a threshold where the values start to plateau. Due to the unpredictability of which generation this would be, and time constraints, I generally overestimated and stopped the evolution when it was clear the best values stopped increasing. This meant however there weren’t any graphs outputted as the evolution script never got through all the generations that it was initially set to. Instead below are some screen shots of when the evolutionary algorithm was used, showing the best score that was obtained.

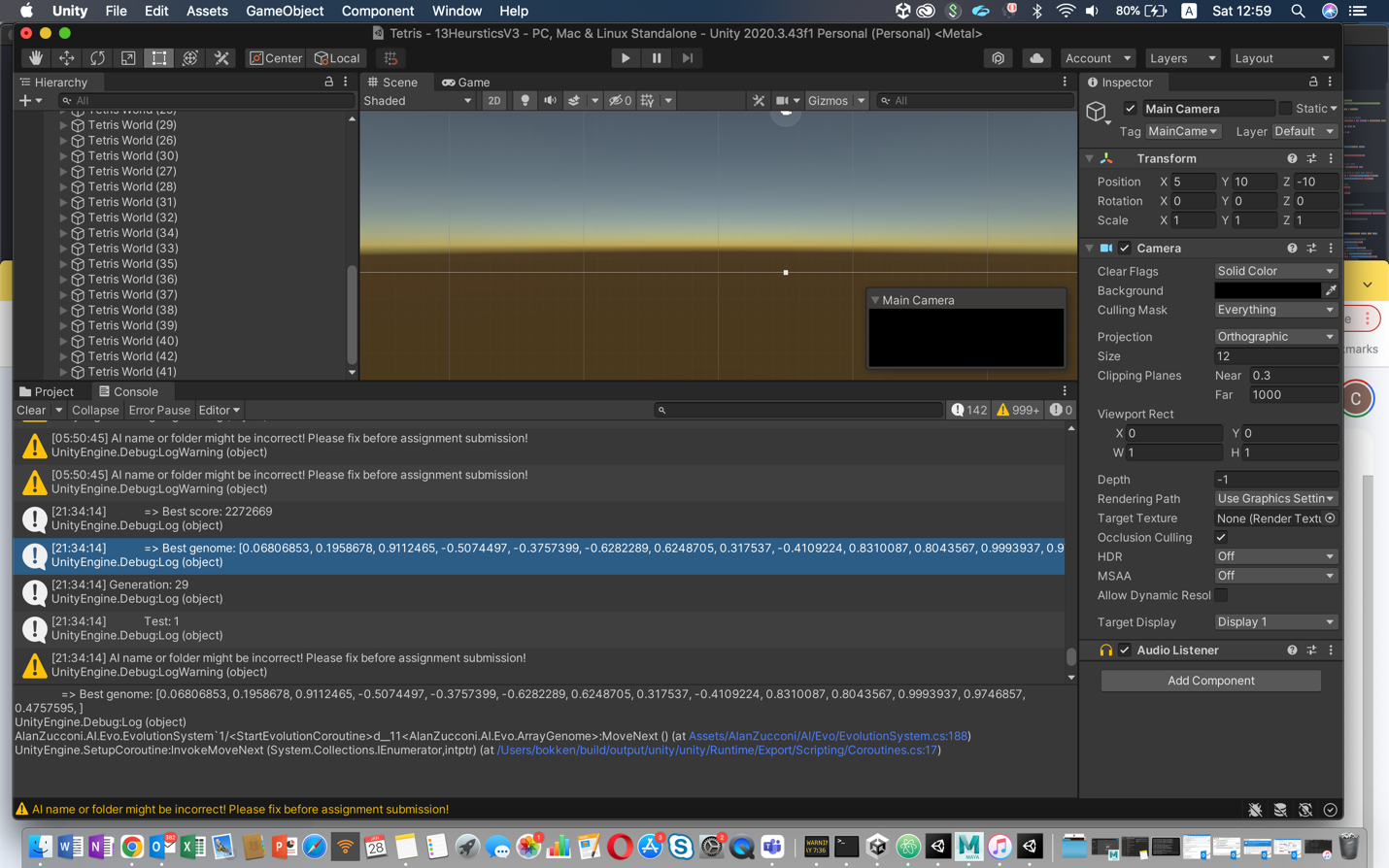
A screenshot of a computer

Description automatically generated with medium confidenceGraphical user interface, application

Description automatically generated

Best score: 436, 4 heuristics used, generation 4

Best score: 59230, 4 heuristics, generation 14



Best score: 2,272,669. This was from generation 28 using all 11 heuristics.

**Optimizing holes heuristic**

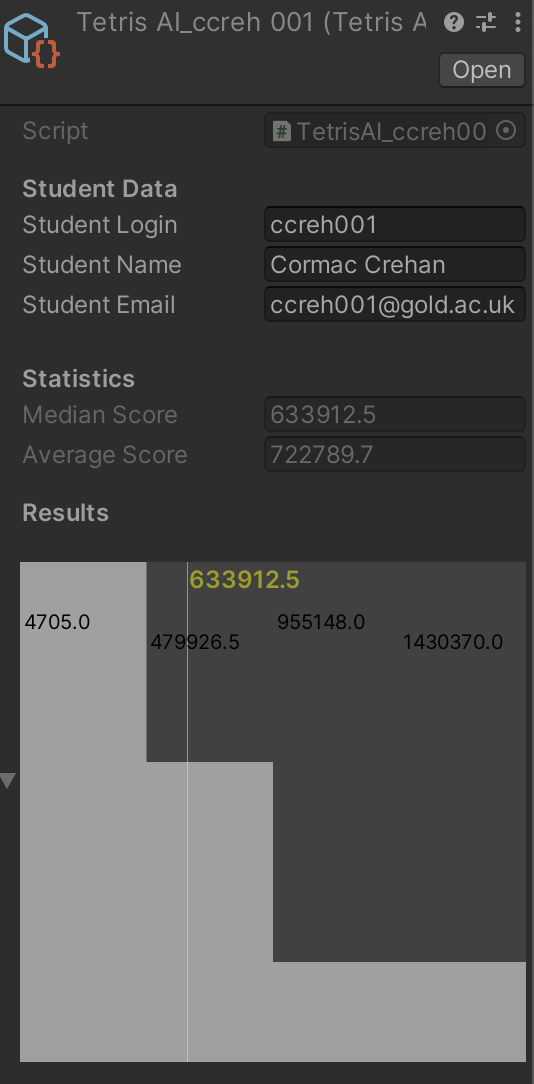
After the score of 30,000 which was obtained from 4 heuristics, it was optimized by separating the hole heuristic into open, and closed holes. This was done to test out if one of them was more desirable than the other.

Another method to optimize this heuristic was to tally the holes that were adjacent to another hole. Within each hole function I made a 2D array of all the holes, and created another output which was called wellNumber. A well is what was categorized as more than one hole next to each other.

**Height heuristics**

As well as the GetMaxHeight() heuristic (which obtained the height of the highest column), I wanted to implement some height heuristics that took into account all of the column heights. To satisfy this, one was made which tallied all the heights (GetTopHeaviness()). Another heuristic was implemented which included the standard deviation of heights which indicates how much the heights deviate from the average height.

**Final results**



The best score I achieved through the evolution was 2,272,669. I used these optimized heuristic weight values for this automation which is shown in the graph.

After running the script 10 times:

The final Median I got was 633912.5

The Average score was 7227789.7

The best score in this automation was 1,4303,770

**Critical Reflection**

Overall, the solution developed for the Tetris game was very effective in getting a high score with my highest score being 2.2 million. However, there are many ways in which my AI could have been improved.

In terms of code implementation, efficiency was important because of how many simulations were being run, and the time it took to run them would affect how optimal the weights were. For the sake of making my code tidy I calculated most heuristics by scanning through the columns and rows, and finding the relevant information needed for that heuristic. This could have been made more efficient by scanning through all the information needed in each column and adding it to a global array. I did have some functions for example the ‘GetHolesAndWells’ function which used the same information to output 2 or 3 heuristics, therefore minimizing the amount of for loops and array scanning . For example my GetHolesAndWells() function outputted the weighted hole value, the hole number and the well number all in the same function. The main reason I didn’t create a global array was because I found debugging to be difficult using a global array due to the fact it would have had to be erased and reset after each simulated iteration so I couldn’t print individual pieces of information to the console. Figuring out a work around could have made things more efficient however.

One heuristic I didn’t implement which probably could have increased my score is by using the predict move method, simulating one move ahead of each variation of the next move ahead and choosing the move with the best heuristic based off this simulation. Due to time constraints and CPU limitations, I anticipated this would have meant much more time was needed to leave the simulation running. In its current state, the evolution script took a whole day to get the right combination of heuristics, as there are already 11 heuristics.

There are also other methods that would have been good to explore. One method that has proven successful in research, is using a convolutional neural network, which would need to be trained with images of Tetris that highlights the grid format and the relative score. From this It would develop its own heuristics based off the data it is given. To do this a convolutional neural network library for C# would have to be implemented (SRDJAN, 2020).

Despite these limitations, the solution was able to achieve good results overall, and the use of the evolutionary algorithm was instrumental in finding the best combination of weights for the heuristics.

Bergmark, M. (2015) “Tetris: A Heuristic Study ,” *Using height-based weighing functions and breadth-first search heuristics for playing Tetris* [Preprint]. Available at: <https://doi.org/http://www.diva-portal.org/smash/get/diva2:815662/FULLTEXT01.pdf>.

Lee, Y. (2013) “Tetris AI – The (Near) Perfect Bot,” *Code my road* [Preprint]. Available at: https://doi.org/https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/.

Srdjan (2020) *Machine learning: Ai plays Tetris with Convolutional Neural Network*, *Ask For Game Task*. Available at: https://www.askforgametask.com/tutorial/machine-learning/ai-plays-tetris-with-cnn/ (Accessed: January 29, 2023).