CMP304 Coursework
Project Report

Genetic Algorithm for Tetris

by: Stylianos Zachariou - 1800997

1. Introduction

Tetris is a puzzle game created by Alexey Pajitnov in 1984. (Tetris Holding, 2022) The goal of the game is to achieve the highest score possible, by placing shapes called "Tetrominoes" and filling horizontal lines on the board. Although the gameplay seems simple, a lot of critical thinking and decision making is required to last longer and achieve a high score.
There are certain parameters that the user must consider when making any decision, therefore, in this project, a genetic algorithm was used. Genetic Algorithms are recognized for efficient function optimization, (Mathew, 2012) meaning that given a problem, they can efficiently find the optimal input, resulting in the most advantageous output. (Browniee, 2021) Thus, for each move during the game, given parameters like: lines cleared, holes made and new maximum height, the algorithm should be able to decide which factors are more important and which should be avoided.

2. Background

Genetic algorithms are a type of evolutionary algorithm. (Hosch, 2017) Evolutionary algorithms follow evolutionary principles to improve and adapt to a specific problem. (Whitley, et al., 1996) Genetic algorithms build on that by using core principles of Darwin's evolutional theory like natural selection and gene simulation. (Mirjalili, 2018)
Even though genetic algorithms can be used in a variety of different problems, all of them follow the same steps: Initialization, Evaluation, Selection, Crossover and Mutation. (Gad, 2018)

Initialization:
Firstly, a population of chromosomes must be initialized. Chromosomes represent different candidates that will attempt to solve the given problem. Each chromosome includes specific parameters, called genes, which will affect its behavior. (Mitchell, 1998) Genes can be represented as floats or binary strings which are randomly generated at the beginning of the application.

Evaluation:
Once all chromosomes are initialized, each of them is evaluated by attempting to find a solution to the problem. Depending on their result, every chromosome is evaluated using a positive fitness or a negative cost rating. When they are done, the beneficial genes can be identified as well as the unrewarding ones by inspecting the fittest individuals. (Beasley, et al., 1993)

Selection:
Natural selection is simulated; the chromosomes with the least cost or the highest fitness are chosen as parents for the next generation. (Sastry & Goldberg, 2005)

Crossover:
To create the new generation from the selected parents, the new chromosomes must inherit genes from both parents. To achieve that, depending on the application, different crossover operators are used to create a child chromosome which has equal characteristics from both the parents. (Erbatur, 2000)

Mutation:
The new chromosomes are also subjected to random mutations, which if beneficial, will help the future generation achieve better results. (De Jong, 1988)

The previously mentioned steps are repeated, however, each time instead of initializing new random chromosomes, the new generation made by the previous parents is used. (Sastry & Goldberg, 2005)

3. Methodology

The Genetic Algorithm created makes use of the OpenGA library. (Mohammadi, et al., 2017) OpenGA provides a genetic algorithm structure (Figure 1) that was extended with custom functions for specifically playing Tetris.

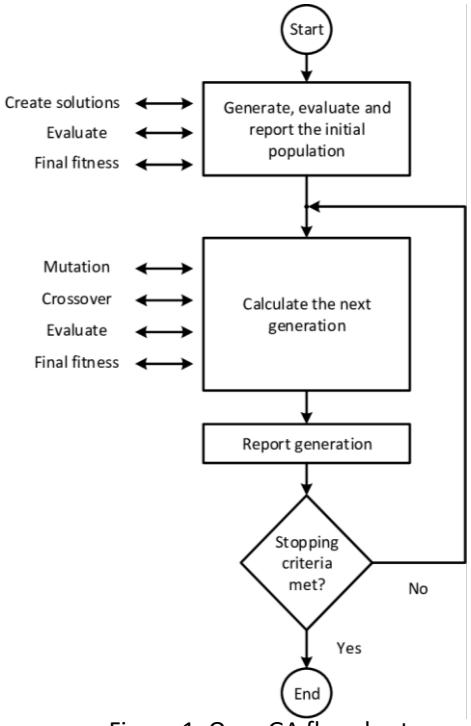


Figure 1: OpenGA flowchart

Representation:

Before designing the genetic algorithm, the genes must be decided. This can be achieved by observing how a human plays Tetris and what parameters they consider before making a move.

Lines Cleared:

The player can only increase their score by clearing lines, therefore, it would be beneficial for the chromosomes, to consider that parameter, making it the first gene.

Holes:

Holes are gaps blocked by Tetrominoes, created on the board by a misplaced piece. These are disadvantageous and most of the times avoided since the only way of fixing them is clearing the above blocks. Accordingly, a gene is created concerned about the number of holes.

Roughness:

In an interview, Jonas Neubauer, seven-time winner of the Classic Tetris World Championship, expressed that the first strategy to consider is “playing flat”. (Ramos, 2019) This means that the player must attempt to drop pieces so as to keep the top of the collective shape as smooth as possible. Hence, a gene was added, that cares about the roughness of the board.

Maximum Height:

To lose in Tetris, a Tetromino must reach the board’s top height. To avoid this, the maximum height of the board must be kept to a minimum. In an attempt of making the chromosomes aware of this factor, a weighted height gene was added.

Total Height:

The best move the player can perform in the game is called Tetris. Tetris happens when 4 rows are cleared at the same time, rewarding the player with the maximum number of points. If the player goes for this technique, the total height of the board must be at least 4 blocks high. (Shaver, 2017) On the other hand, the player can try to keep all columns as low as possible, resulting in a slower but safer way of playing. To simulate this decision, a gene for the cumulative height is supplemented.

Relative Height:

In order to successfully complete a Tetris move, there should be a column that remains to a much lower height than the others. To make the chromosomes aware of this, the relative height (the difference between the highest and the lowest height) is appended to the list of genes.

Highest Hole:

Even if there are holes on the board, the player must avoid creating higher holes on the board. The higher the holes, the less the space the player has to clear lines. Therefore, the last gene added to the chromosomes is the “highest hole” gene.

Each of the above genes can be represented using a float in the range of -0.5 – 0.5. The chromosomes are structures holding the above floats. (Figure 2) Each chromosome is also connected to a "Cost" structure that holds an integer variable with the fitness score of the chromosome.

```
//Chromosome Structure
struct Chromosome {

	//Genes
	float linesCleared;
	float holes;
	float roughness;
	float weightedHeight;
	float cumulativeHeight;
	float relativeHeight;
	float highestHole;
};
```

Figure 2: The Chromosome Structure

Initialization:

The first step of the Genetic Algorithm is initializing a population of 50 chromosomes. For each gene in each chromosome a random number generator is used returning a value between 0 and 1. That value is then subtracted by 0.5 to bring it in the range of -0.5 to 0.5 for each gene. The result of the initialization is 50 chromosomes with random gene values ready to play Tetris.

Evaluation:

The next step is evaluating all chromosomes based on how well they have played the game. To achieve this, an evaluation function is called which creates a new level for the chromosome to play in. This function also keeps track of how many lines the chromosome cleared and whether it has lost the game. The game loop includes three processes: finding the best move, carrying out the move and updating the game.

Deciding on the next move:

A move made by a chromosome can be described using the "Move" class. This class includes a vector holding all stages that were carried out to complete a move as well as a vector holding the board visual after the move is completed. Moreover, the class also includes an evaluation function which goes through the board after the move was carried out and stores the number of lines cleared, the number of holes created, the board's roughness, the weighted height, the cumulative height, the relative height, and the highest hole.
To find all possible moves that the chromosome can perform, the function "allPossiblePositions" in the "Board" class is used. This function moves the current piece from the left to the right side of the board by executing a complete drop at each position and saving the move as an object of the move class. When the piece reaches the right side of the board, it is reset to the left side and rotated. The same process is repeated for all four rotations and the results are returned in a vector.
The move vector along with the chromosome's genes are passed as parameters in the "bestMove" function. Then, this function performs a linear combination calculation (Figure 3), and the result is stored as the move's score. The calculation is executed for all moves and by comparing the scores, the move considered as more favorable by the chromosome's genes is decided.

```
float thisMoveScore =
allMoves[i]->getLinesCleared() * individual.linesCleared +
allMoves[i]->getHoles() * individual.holes +
allMoves[i]->getRoughness() * individual.roughness +
allMoves[i]->getWeightedHeight() * individual.weightedHeight +
allMoves[i]->getCumulativeHeight() * individual.cumulativeHeight +
allMoves[i]->getRelativeHeight() * individual.relativeHeight +
allMoves[i]->getHighestHole() * individual.highestHole;
```

Figure 3: Move Score Calculation

Playing a move:

After finding the best move, the chromosome must attempt to play it. To be fair, the chromosomes are only allowed to move the piece one block each frame. This is done using a move counter variable which saves the move-stage the chromosome is currently playing. With that counter and the vector holding all stages required to complete a move, the move type is passed into the "Level" class's "handleInput" function. Finally, the move counter is incremented. When the piece changes then the move counter is reset, and a new "best move" is decided.

Updating the Game and Returning the Cost:

The game is updated using the "Level" class's "update" function which returns whether the chromosome lost the game or is still playing. If the chromosome loses, the game loop is terminated, and the total score earned is returned to the chromosome's cost variable. The level is deleted, and the genetic algorithm moves on to the next step.

Selection:

To select the best chromosomes of the generation, the score saved as middle cost must be converted to the total cost. This is done using a custom fitness function which takes the middle cost (score gained in the evaluation function) and makes it negative. This is done so that the chromosomes with the highest scores will have less cost than chromosomes with less scores. (Figure 4)

	Middle Cost/ Score	Total Cost	
Chromosome 1	100	-100	**Least Score -> More Cost**
Chromosome 2	1500	-1500	**More Score -> Less Cost**
Chromosome 3	1430	-1430	**Medium Score -> Medium Cost**

Figure 4: Middle Score to Total Cost Examples

After calculating all chromosome's total costs, the openGA framework selects the fittest chromosomes (with the lease cost) and lets them reproduce.

Crossover:

The crossover function is responsible for creating new child chromosomes from the two selected parents. At the top of the function, a new chromosome is created that is an exact copy of the first parent. Then, for each gene, a random number generator that returns either 0 or 1 decides whether the gene will remain the same as the first parent's or will change to be identical to the second parent's. (Figure 5)

```
Chromosome newChromosome=X1;
if ((int)rnd01()*2 == 0)
{
        newChromosome.linesCleared = X2.linesCleared;
}
```

Figure 5: Crossover example with "Lines Cleared" gene

Mutation:

The mutation function is used after all new children are created and changes some of their genes to create more variation and probable new beneficial gene values. This function takes in the base chromosome and using a random number generator returns a number between 0 and 6. Then depending on that random number, a respective gene is chosen to be re-initialized; meaning that the gene's value becomes a new random number between -0.5 - 0.5. The mutation rate can be changed before the genetic algorithm starts, to allow experimentation with different mutation percentages.

Generation Report:

The last function called at the end of each generation is the report function. This function is responsible for printing out useful information about the generation and the best chromosome. Moreover, it re-plays a game using the best chromosome and renders it in a window to show the user the chromosomes approach. (Figure 6)

Figure 6: A generation's report

4. Testing and Results

Even though this genetic algorithm has a lot of factors affecting its performance, three decidedly important ones were chosen to be tested. Each factor was tested using 5 different values, which were then iterated 5 times each. The goal of the tests is finding the values which allow the genetic algorithm to perform as best as possible.

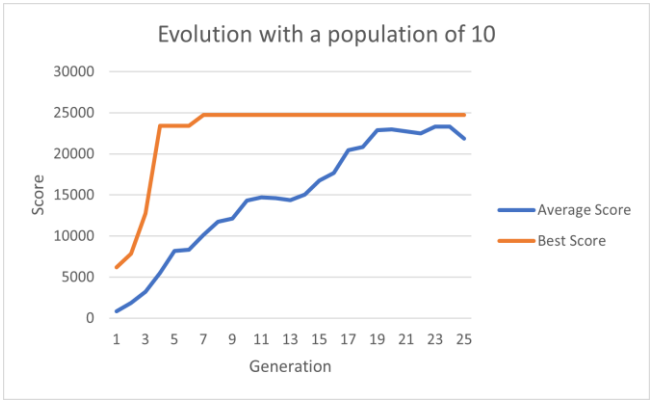
Mutation Rate:

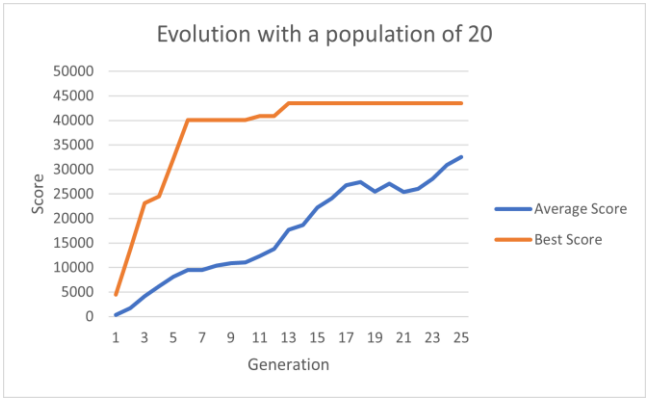
The mutation rate dictates what percentage of the population is mutated each generation. The mutation was tested in the range of 0%-100% at 25% intervals. In each test, both the average and best scores were considered showing how the population grew with the given mutation.

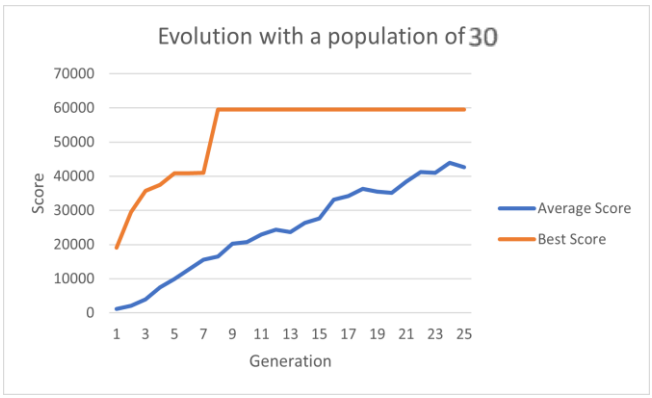
Figure 7: Generation results of varying mutation rates

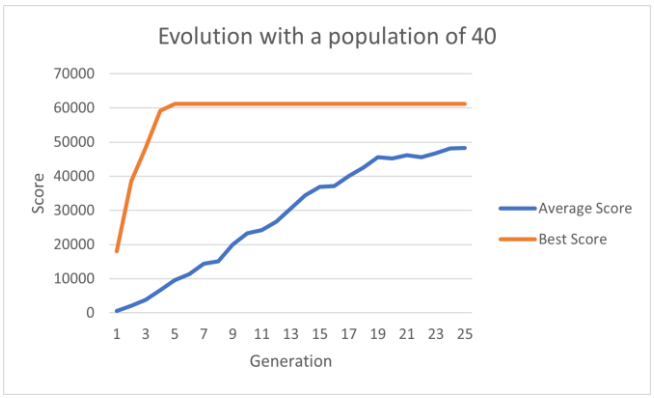
Population:

The population signifies the number of chromosomes that actively try to play the game each generation. The population was tested with values of 10,20,30,40 and 50; the best and average score of each were recorded and plotted.









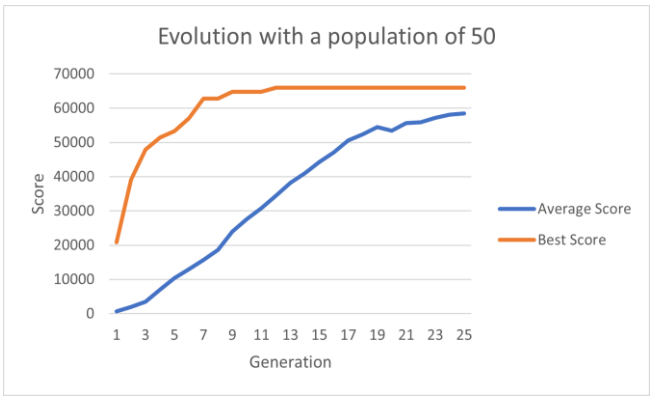


Figure 8: Generation results of varying population sizes

Line Limit:

When each chromosome attempts to play the game, they either lose or reach the cleared line limit. By experimenting with different line limits, the adaptation of the genes can be observed. Therefore, using the line limits of: 50,100,250,500 and 1000, different patterns can be identified for each gene.

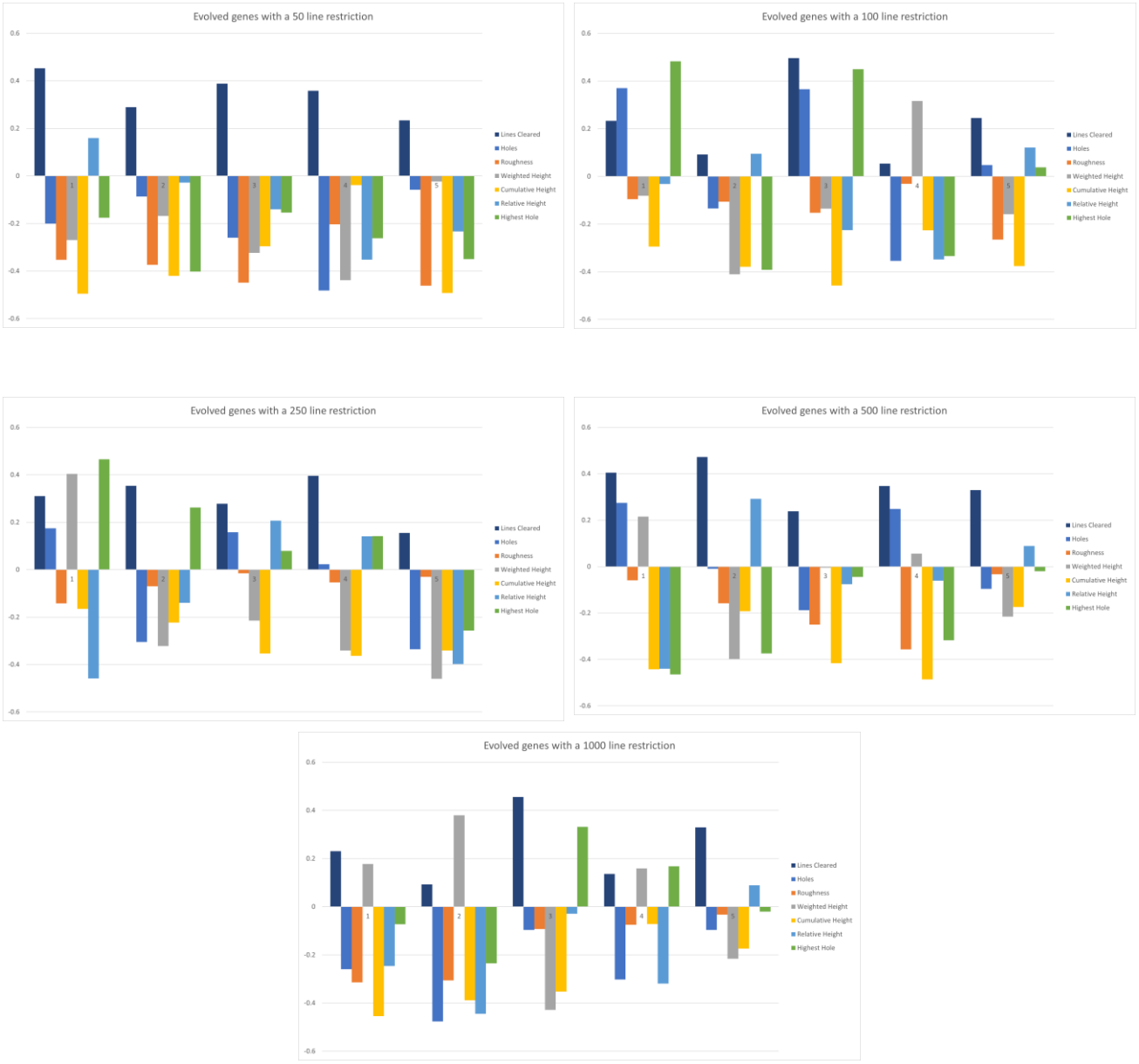


Figure 9: Best genes with varying line limits

5. Discussion

Firstly, although the algorithm had a range of results for each of the above tests, the same pattern can be distinguished in all of them when looking at the average and best score graphs. In all tests using a higher mutation rate than 0%, the increase of the best score throughout the generations has a logarithmic pattern. This signifies that as the generations progress the genes move closer to their optimal values which limits the chances of another better mutation. It can also be understood that, at the beginning since all chromosomes are randomly initialized, there is a lot of room for improvement. which is observed by the steep score increase at the beginning of the graphs.

Mutation Rate:

As indicated by the results, the mutation rate is essential in allowing the chromosomes to continually achieve better scores. By observing the 0% mutation graph, a huge disadvantage is immediately noticeable; the best score is stable. By having no mutations, the population can only have a combination of the initial gene values which restrict the chromosomes' performances. To get a high score when using such a low mutation rate, the chromosomes have to be "lucky" with the initial values of the genes, since new gene values are never introduced. Once the most advantageous combination is found, no further development is available.

Another pattern that can be identified by the results, is that when the mutation rate is 100%, the average score between all chromosomes has a higher difference with the best score than algorithms with a lower mutation rate. This is because even though the best chromosomes reproduce for the next generation, no chromosome will only include the genes given by their parents. This introduces to much randomness which doesn't allow the chromosomes to evolve steadily.

The most advantageous mutation rate is 50%, since there is continuous increase of the best scoring gene but also the average score of all chromosomes comes very close to the best score, meaning that there are a lot of chromosomes that achieve high scores even though they are not the best. A 50% mutation rate allows both child chromosomes with a mixture of the parent's genes and chromosomes with new mutation to grow.

Population:

The population size has dramatic changes to the performance of the genetic algorithm as presented by the results. By looking at the graphs, it is obvious that as the population increases, the score increases as well. The difference between best scores of the sized 10 and sized 50 populations is almost 40000. This was expected since with a larger population, there is more room for mutations and constructive crossovers.

Another sequence that can be noticed in the graphs, is that the highest the population, the smoother the increase of the average scores. This happens because there is more room for different genes to exist. When having only 10 chromosomes, there is a high chance that most of them will be negative or positive, where when the population is 50 there is a higher chance that there will be a variation of skill levels which increase steadily using crossovers and mutations.

Line Limit:

The line limit adds a sense of urgency to the chromosomes. Even though they don't lose the game, they want to finish the round with as many points as possible.

Lines Cleared:

Even though the line limit changes, the "lines cleared" gene always remains positive. As expected, this shows that there was a positive correlation between this factor and the score.

Holes:

Overall, this gene was kept to a negative value by the best chromosomes, signifying the avoidance of the creating of holes. In a minority of the results though, this gene was kept close to 0, showing that the gene was not considered an important factor, or was a positive amount, indicating that pared with other combinations of genes this gene might be beneficial when positive.

Roughness:

The "roughness" gene was always negative in the most favorable chromosomes. This shows an obvious negative correlation between the board's roughness and the achievement of a high score.

Weighted Height:

Usually, the "weighted height" or "maximum height" gene included in the best chromosomes' genes was negative. As anticipated, this was due to the fact that if the maximum height needs to be kept to a minimum. Surprisingly though, occasionally, this gene was positive, which means that the chromosome was trying to

have the highest maximum height. However, at those instances the other two height variables have very high negative values, which balances this gene and prevents the chromosome from immediately losing the game by stacking Tetrominoes in a tower.

Cumulative Height:

Throughout all testing, this gene remained a high negative value, compelling the chromosome to keep the total board height as low as possible, which allows it to play safer and therefore, last longer. This indicates a clear negative correlation between the cumulative height and a high score.

Relative Height:

Similarly to the "weighted height" gene, although this gene was negative most of the times, sometimes a positive value was assigned to it, forcing the chromosome to try to get the highest difference between the highest and lowest heights. Although unanticipated, this gene was balanced by the other two height variables which were always negative when this, was positive. Because of that, the chromosome was unable to build a tower and lose the game immediately.

Highest Hole:

Although this gene was mainly negative, there are a considerable number of instances with it assigned a positive value. This signifies how in general this gene is a negative attribute but if paired with the right combination of other genes then a high score can be achieved.

6. Conclusion

To conclude, all basic functionality of genetic algorithms was successfully implemented into the project. All functions worked well together to create a population of chromosomes that are skilled to play Tetris and achieve extremely high scores. The genes selected were sufficient for the intended purpose of the chromosomes and could easily have been extended to make even more sophisticated decisions. If the project was going to be recreated, it would have had better results if the chromosomes could have had a sense of losing the game. As the project is today, the chromosomes play a move because they think it's the best, without considering if that move will make them lose the game. Moreover, instead of simulating all possible ending positions, all possible move type combinations could have been simulated, allowing the chromosomes to complete moves like sliding under a block or even "T-Spins". (Shaver, 2017) Finally, genetic algorithms were an excellent choice of AI for playing Tetris and with some modifications this algorithm can become an unstoppable Tetris player.

7. References

Beasley, D., Bull, D. R. & Martin, R. R., 1993. *An overview of genetic algorithms: Part 1, fundamentals.* Cardiff: s.n.

Browniee, J., 2021. *A Gentle Introduction to Function Optimization.* [Online]
Available at: https://machinelearningmastery.com/introduction-to-function-optimization/#:~:text=Learning%20is%20treated%20as%20an,maximum%20output%20from%20the%20function.
[Accessed 25 March 2022].

De Jong, K., 1988. Learning with genetic algorithms: An Overview. *Machine Learning,* Volume 3, pp. 121-138.

Erbatur, F., 2000. Evaluation of crossover techniques in genetic algorithm based. *Computer and Structures,* 78(1-3), pp. 435-448.

Gad, A., 2018. *Introduction to Optimization with Genetic Algorithm.* [Online]
Available at: https://towardsdatascience.com/introduction-to-optimization-with-genetic-algorithm-2f5001d9964b
[Accessed 26 March 2022].

Hosch, W. L., 2017. *Genetic Algorithm.* [Online]
Available at: https://www.britannica.com/technology/genetic-algorithm
[Accessed 26 March 2022].

Mathew, T. V., 2012. *Genetic Algorithm,* Mumbai: Indian Institute of Technology Bombay.

Mirjalili, S., 2018. Genetic Algorithm. In: *Evolutionary Algorithms and Neural Networks.* s.l.:Springer, Cham, pp. 43-55.

Mitchell, M., 1998. *An Introduction to Genetic Algorithms.* 1st ed. s.l.:Massachusetts Institute of Technology.

Mohammadi, A., Asadi, H., Mohamed, S. & Nelson, K., 2017. *openGA, a C++ Genetic Algorithm library,* Canada: s.n.

Ramos, J., 2019. *Tetris tips from a seven-time world champion.* [Online]
Available at: https://www.polygon.com/guides/2019/2/22/18225349/tetris-strategy-tips-how-to-jonas-neubauer
[Accessed 27 March 2022].

Sastry, K. & Goldberg, D., 2005. Genetic Algirthms. In: *Search Methodologies.* s.l.:Springer Boston, pp. 97-125.

Shaver, M., 2017. *How to Perform a T-Spin in Tetris.* [Online]
Available at: https://tetris.com/article/70/how-to-perform-a-t-spin-in-tetris
[Accessed 28 March 2022].

Shaver, M., 2017. *My Favorite Tetris Strategy - The Tetris Line Clear.* [Online]
Available at: https://tetris.com/article/45/my-favorite-tetris-strategy-the-tetris-line-clear
[Accessed 27 March 2022].

Tetris Holding, 2022. *History of Tetris.* [Online]
Available at: https://tetris.com/history-of-tetris
[Accessed 25 03 2022].
Whitley, D., Rana, S., Dzubera, J. & Mathias, K. E., 1996. Evaluating evolutionary algorithms. *Artificial Intelligence,* 85(1-2), pp. 245-276.