Tetris Bot Using Genetic Algorithms and Evolutionary Programming

This paper discusses the design, implementation and testing of the Tetris Artificial Intelligence Bot for the ICT219 Project. This paper will examine the AI methods implemented as well as the tools and evaluation techniques used. The final conclusion, that the Artificial Intelligence Bot achieved the original goal is presented to the reader, along with the evidence to support this conclusion.

Introduction

The game of Tetris was first introduced by the famous Russian video game designer and computer scientist, Alexey Pajitinov in 1984. Though there have been many versions of the game to come out since the original version in 1984 its popularity, success and fame has remained throughout the years.

Tetris is a falling block game where the player must arrange different shapes called 'tetrominoes' which comprise of 4 blocks. These must be manoeuvred in the grid to form complete lines and earn points. The overall goal is maximise the points scored, and thus there is no way to *win* Tetris.

This paper describes the application of Genetic and Evolutionary algorithms used to implement a Tetris Artificial Intelligence Bot with the goal of evolving a Bot capable of beating the average human player's high score. The Tetris Bot chooses the best move by evaluating all possible moves that a tetromino could be placed and rating each of these moves based on a variety of rating functions. These rating functions form the basis of their Genetic chromosome and are optimised using the process of selection to find the optimum weights.

Background

The game of Tetris has been a popular candidate for the creation of Artificial Intelligence. Many notable Computer Scientists and enthusiasts have analysed the game and tried their hand at an Artificial Intelligence method for maximising the high score. One of the most famous examples of the application of a Genetic Algorithm to the Tetris game was done by Pierre Dellacherie with a one-piece solving algorithm (a Bot that has no knowledge of the future game tetromino). Other examples of simple Tetris Artificial Intelligence include work by coder Lee Yiyuan. In his online guide, 'Tetris AI- The (Near) Perfect Bot'[6] Mr Yiyuan explains the application of a genetic algorithm to optimise the Tetris Bot's score.

Erik Demaine is commonly known for his work in the proof that the Tetris solution is NP-complete [3],[4], which refers to a group of problems that are both NP and NP-Hard, where NP refers to the term 'nondeterministic polynomial time'[5].

Given the previous success of other programmers with the Genetic Algorithm problem solving approach, the application of a genetic algorithm and evolutionary programming was chosen the complete the ICT219 Project.

AI Methods and Tools

Game Engine

The Tetris game forms the basis of which the Tetris Bot can operate in and the environment that allows the Bot to play the game. Implementing a version of Tetris could be done using a previously made Tetris Clone, or a custom built design.

For the ICT219 Project the game engine was custom built according to the specifications and standards documented (see Appendix A.). This was done to maximise the functionality of the code and integrate the Tetris Bot in to the game more seamlessly. This also removed the need to work with difficult scripting languages and maximised the scope of what could be done with the Tetris Bot.

Genetic and Evolutionary Algorithms

Genetic algorithms are commonly used to generate useful solutions to optimisation problems. This is done through the application of the evolutionary process of natural selection and genetics. In the case of the Tetris Bot the goal is to optimise the Bot's score.

To a human player maximising the game score is somewhat intuitive if we know how to earn points. Humans have complex reasoning faculties that allow them to simultaneously evaluate the relevant factors and we use these to make choices throughout the game. For example; in a game of Tetris the players decides where to place the next tetromino. The human player intuitively understands that there are a variety of factors that will allow them to maximise their high score, such as: clearing the most lines, minimising the height and avoiding blockades. It was this process that needed to be translated in to an algorithm.

This made the AI for the Tetris game an ideal candidate because we are looking at an optimisation problem. The question we are most trying to answer is; what is the optimum position for the tetromino; and how do these factors vary based on the current game state and future game state. Unfortunately however, there is no specific solution or recipe for how each tetromino should be placed in order to maximise the score of a Tetris game. For example; we cannot say that it is ALWAYS best to place the tetromino to minimise the height of the grid, because sometimes it may be necessary to sacrifice height in order to clear a row. Some combination of these factors are the necessary 'ingredients' to produce a fairly robust solution but we do not have a clearly defined 'recipe'.

The basis of the genetic algorithm is built around the chromosome, also sometimes called the genotype, which is modelled off the chromosomes located in plants and animals. Each of these real chromosomes is made of protein and a single cell of DNA which contains specific instructions that make each living creature unique. The chromosome within a genetic algorithm is therefore designed to represent a solution to the overall goal we are trying to achieve. It contains the 'instructions' on how to solve the given problem. In this project the values that comprised the chromosome came in the form of weights of the tetromino ratings.

The rating functions are based off the elements of the game that can help a player prolong losing. For example, minimising the height of a tetromino is almost always a safe strategy. So the rating function penalises the Bot by giving a higher height value for tetromino places that increases the overall height of the game state. The goal is to minimise this value so the evaluation will favour Bot's that avoid piling up tetrominoes. The rating functions are used to rate the state of the game grid after the tetromino has been placed in a hypothetical position. Each position a tetromino could be

placed in is evaluated based on these rating functions. The position that is rated as the best place overall for the tetromino, is where the AI moves the tetromino.

The following rating functions were used for evaluation:

- *Aggregate Height* - The sum of the individual column heights on the game grid
- *Aggregate Bumpiness* - The absolute difference in height between individual column heights on the game grid
- *Blockade Count* - The number of holes in the game grid surrounded by filled blocks
- *Rows Cleared* - The number of full horizontal rows on the game grid

The next integral part of the evolutionary algorithm was the design of the fitness function. The fitness function evaluates the fitness of an AI Bot to gauge how well it is achieving in comparison to its competitors. The Tetris game conveniently already had an adaptable method for rating each Bot's fitness. The fittest of the bots are those that achieved a high scoring game.

Using these functions the program was designed as a tournament style competition, populated at the start with two Bots. The game is run and these two Bots compete for the highest score. The best scoring Bot of these two is selected to become the next challenger bot. The challenger bot has the same chromosome as the tournament winner only the mutation function is applied and randomly changes one characteristic of the chromosomal makeup, by a small amount. This process is repeated until a sufficient scoring Bot has evolved. A new generation is considered to be created when the challenger Bot wins a round over its predecessor. This method of selection is referred to as 'fitness proportionate reproduction[2]' where, 'the process continues until a suitable termination condition is satisfied.[2]'.

Code Techniques and Challenges

The early versions of the program were both interesting and exciting to work on. Implementing the program posed a number of different and challenging algorithms that needed to be mastered before the genetic algorithm could be added.
For the game engine learning how to organise and update the game grid, as well as implement the player controls took time to master. Once the game engine was completed the task was how to implement the AI game play and what data structures would be most suitable.

The internal function that determines every possible position a tetromino could be placed and how the tetromino will get there, is a path finding algorithm combined with a flood-fill type algorithm. The rating function for blockades was also written using a flood fill algorithm, which required the use of a recursive algorithm.

Results

Human Players

The goal of the ICT219 Project was to create a Tetris Bot that could beat the average human player's high score. To gauge what this value might be five candidates were selected to test and play the custom build game engine. Each player was asked to play five games to the best of their ability. The following results were obtained:

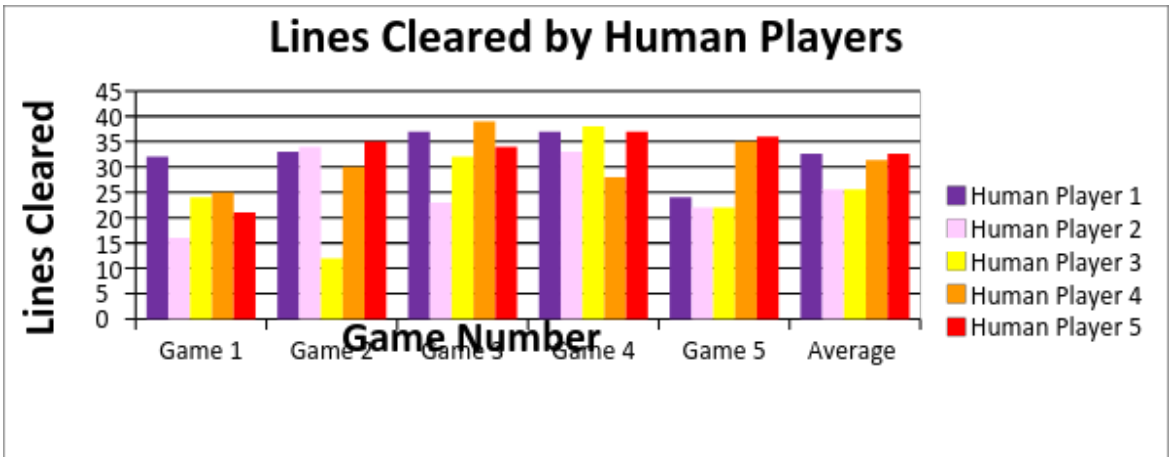
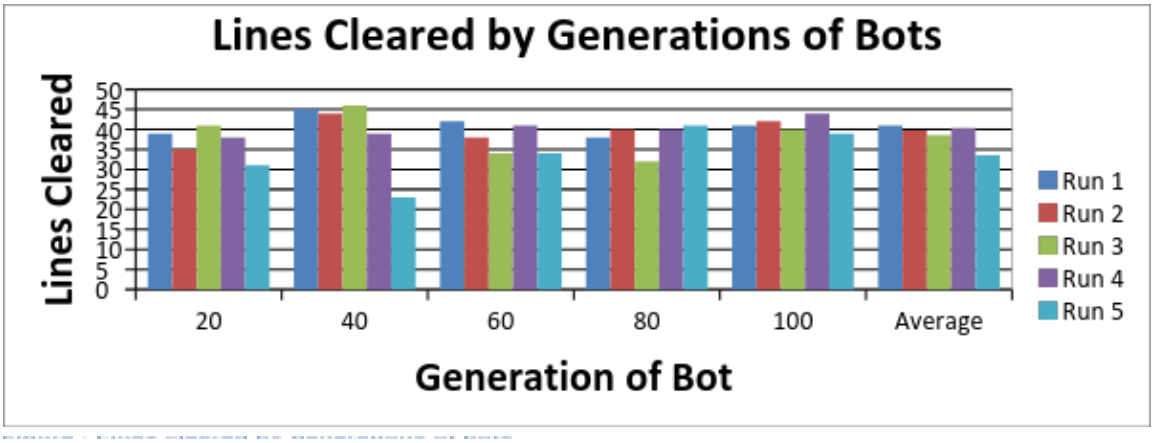


Figure 2: Lines cleared by human players

Thus the maximum average high score for the players was 30 lines cleared. This is the value selected as the score to beat.

Tetris Artificial Intelligence Bot

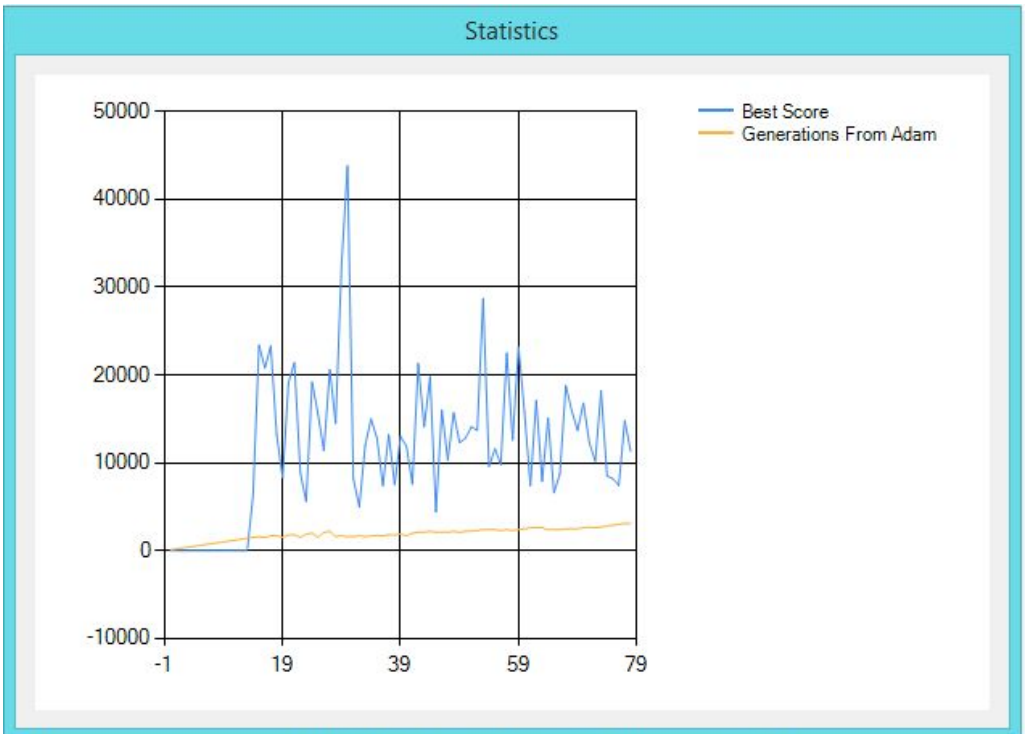


The results obtained from the Tetris Bot program averaged a high score of around 40 lines, which has been a great success.

Evaluation

Previous Versions

Initially, the graph showed an average increase in the Bot's skill over successive generations. However the best score varied quite dramatically over the generations. While the Bot has been successful at beating the average human players score it was underachieving in comparison to its own known potential. This suggested that the Bot was stuck at a local maximum.



This could be happening due to a number of factors. One of the possible reasons could be due to the rating functions. The Genetic Algorithm is only as good as the tools it has been given to solve the problem; in the case of the Tetris Bot these are the rating functions. If the algorithm is lacking the necessary rating functions to achieve its full potential the Bot will meet a local maximum.
Due to time constraints only the most basic rating functions were added, given additional time more rating functions could be added to enhance the Bot's capabilities.

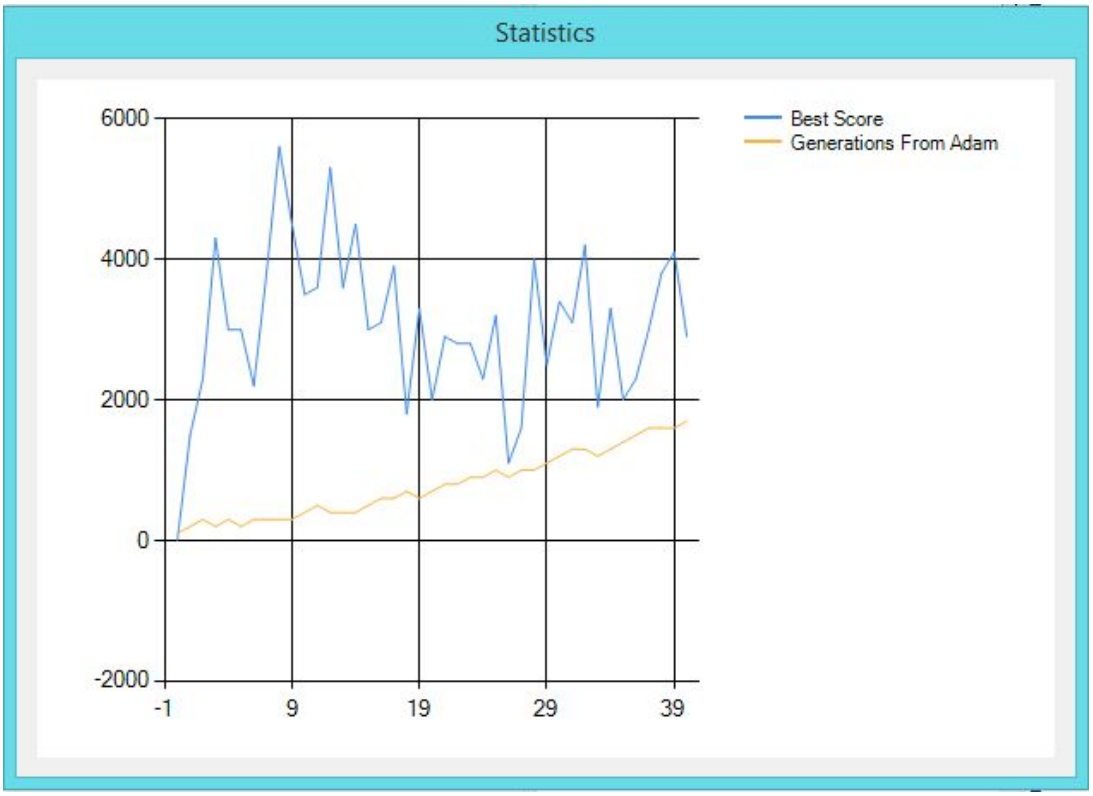
The rating functions were also written as individual, independent rating functions, and the rating was given as a constant value regardless of the conditional factors of the state of the game grid. But some conditional rating functions could be of benefit. For example; there are times in the game where a high score in Function A may be the desired score, provided the state of the game has a high score in Function B; otherwise a low score in Function A may be desired.

The big differences in the Bots score across generations could also be due to the implementation of the Genetic Algorithm and program functionality. Due to the limitations of computational power and time constraints a number of features were omitted in this version of the program.

The first feature that was omitted was the “Think Ahead” feature. In this version the program is designed so that the Bots evaluate all of the places the current tetromino could hypothetically be placed. But the human player has access to more information via a box in the top right corner, the “future” tetromino, the next tetromino that is coming in the game. Using the “Think Ahead” feature involved evaluating all possible hypothetical combinations that the current and future tetromino could be placed which proved to be to computationally demanding on the program, especially with multiple Bot clients running simultaneously.

After a few hours only a small number of generations had passed:

The



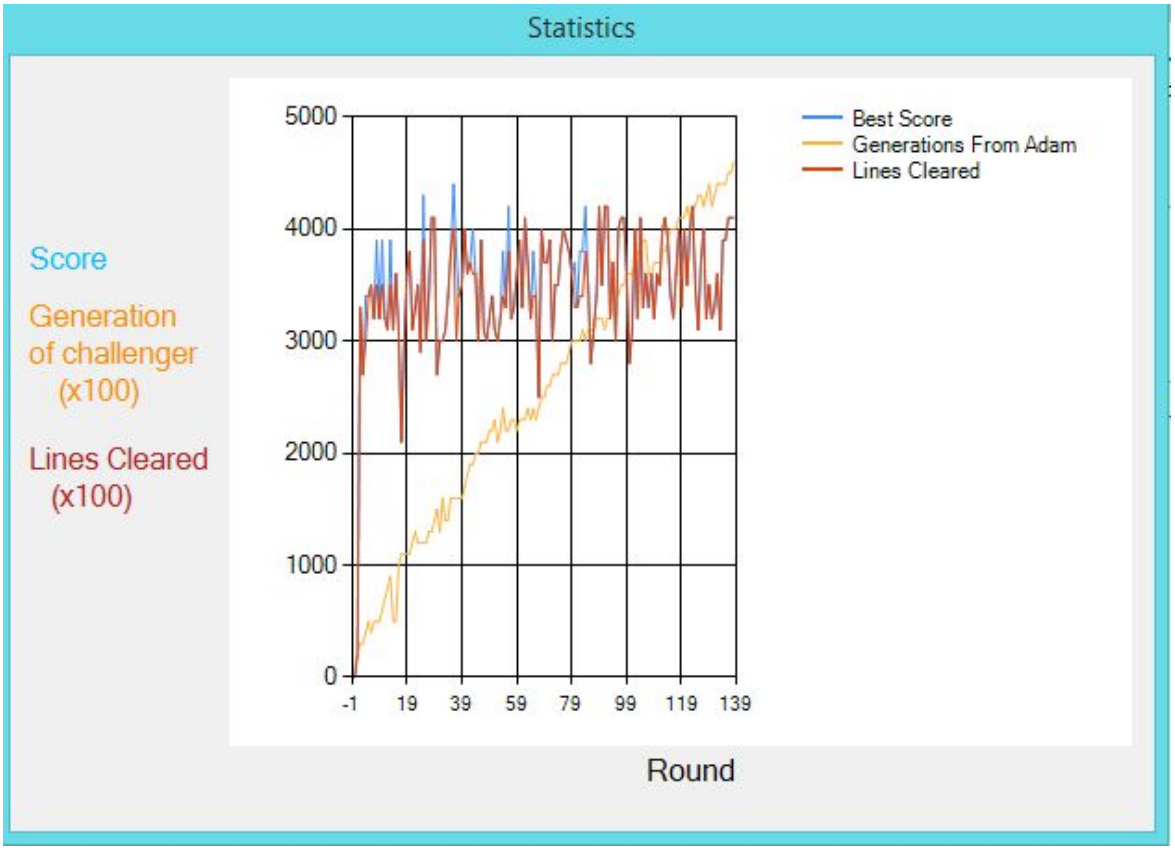
number of Bot clients was also reduced from the orignally planned 5 or 10 clients to 3, in a Tournament style selection. Again this was due the computational limitations and also done in an effort to streamline the prototype so it could be completed.

However, the main concern in this version of the program was the lack of generations from Adam. In the initial stages of evolution the Bots were dying at a much faster rate, causing their games to last only a short amount of time. This meant in the early stages of evolution many generations were generated as the Bots died out faster. As the Bots began to play the game more adeptly their games lasted much longer, causing the generations to taper off dramatically. If we consider the Bot’s as a “live” population of “people” these results tell us that the bots live too long; which shows a lack of evolutionary pressure. Also, as the Bot’s get better it is exponentially harder for them to improve as their games take a longer and longer amount of time. This problem was realised and solved in the final version.

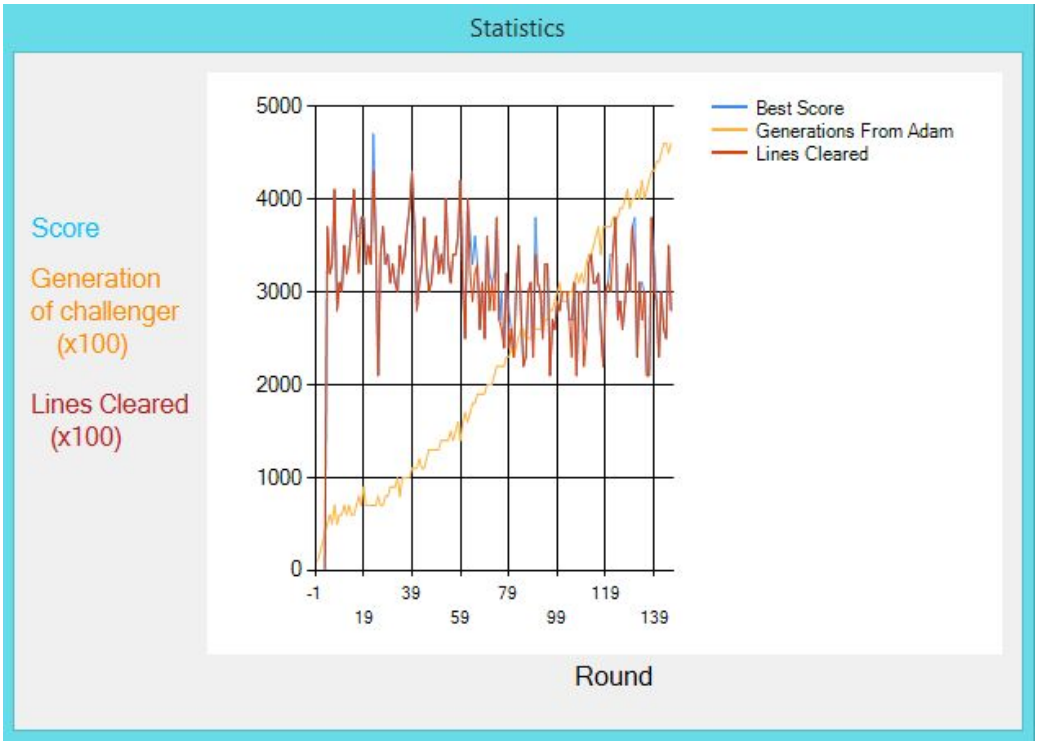
The Final Version

The first problem addressed was the lack of generations being produced. To counteract this problem the level of difficulty within the game was increased. In doing this, Bot’s once again died out faster,

and more generations were generated in a shorter time. The increase in game difficulty can be compared to an increase in evolutionary pressure, where we place more pressure on the Bot's in the game to encourage a higher rate of evolution. Already this helped to improve the overall upward trend and produce more generations:



The "Think Ahead" feature was also restructured to make it computationally faster. Previously every possible combination of the two tetrominoes was evaluated and rated, and the highest rated combination picked. This feature was changed to evaluate the possible positions of each tetromino and pick the combination of the two highest rated positions.



These changes decreased the overall lines cleared and best score, but it did help to increase the number of generations.

Another thing that remained constant was the jaggedness of the graph and dramatic highs and lows. These jagged lines are a reflection of the gene pool size. Due to the random nature of the Tetris game, low scores often mean entire generations of Bots are wiped out due to upsets. A bigger gene pool size would give Bots a greater chance of stability and probability of recovering from these random upsets. Increasing the size of the gene pool would also better reflect the favourable traits amongst the Bot's and contribute to a more stable evolution. Though it is worth noting that this would come at the cost of increasing the time it takes to evolve a Bot. In the book 'Introduction to Genetic Algorithms[1]' S.N.Deepa states, 'it is established that the time required by a GA to converge is O (nlogn) function evaluations where n is the population size.[1]'

Discussion

The Tetris Artificial Intelligence Bot was designed with the purpose of beating the average human player's high score. The performance of the Tetris Bot was measured against 5 capable, average human players and given a maximum of 100 generations to beat their average high score. The Tetris Bot has been, overall, very successful in achieving this goal and a great learning experience.

The most surprising results have been from the "Think Ahead" feature; from the data collected within the graphs it was not apparent if the additional knowledge of this feature vastly improved the Bot's overall performance. Given the other aspects of the prototype that could have been improved, we may have seen a more apparent advantage in the "Think Ahead" feature if some of these improvements were implemented.

From the results obtained the most needed improvement could be said to be an increase in the size of the population/gene pool. The book 'Introduction to Genetic Algorithms[1]' discusses the importance of beginning with a large population so that the total search space can be explored[1].

In conclusion the Tetris Artificial Intelligence Bot has been an overall success and has achieved the original goal, scoring higher than the average human high score of 30 lines. The design, testing and implementation of the Tetris Artificial Intelligence Bot have been successfully analysed and reviewed. The Tetris Artificial Intelligence shows great potential success with future development and improvement.

Acknowledgements

Special thanks must go to Mr Khan Maxfield, Lead Programmer and Developer on the Bioverse Project at Euclideon Pty. Ltd. for his time, advice and assistance with this project.

References

[1]S. Sivanandam and S. Deepa, *Introduction to genetic algorithms*. Berlin: Springer, 2007.
[2]L. Chambers, *The Practical handbook of genetic algorithms*. Boca Raton, Fla.: Chapman & Hall/CRC, 2001.
[3]E. D. Demaine, S. Hohenburger and D. LIben-Nowell, *Tetris is Hard, Even to Approximate*, 1st ed. 2002.
[4]N. Bohm, G. Kokai and S. Mandl, *An Evolutionary Approach to Tetris*, 1st ed. Erlangen-Nuremberg: Departments of Computer Science 8 and 2, University of Erlangen-Nuremberg, 2015.
[5]Wikipedia, 'NP-completeness', 2015. [Online]. Available: https://en.wikipedia.org/wiki/NP-completeness. [Accessed: 23- Oct- 2015].
[6]L. Yiyuan, 'Tetris AI - The (Near) Perfect Bot', *Code My Road*, 2013. [Online]. Available: <https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/>. [Accessed: 23- Oct- 2015].

Bibliography

- C. Fahey, 'Tetris', 2010. [Online]. Available: <http://www.colinfahey.com/tetris/tetris_en.html>. [Accessed: 23- Oct- 2015].
- E. Shahar and R. West, *Evolutionary AI for Tetris*, 1st ed. Massachusetts: University of Massachusetts, 2010, pp. 1,2,3.
- Lucky's Notes, 'Coding a Tetris AI using a Genetic Algorithm', 2011. [Online]. Available: <https://luckytoilet.wordpress.com/2011/05/27/coding-a-tetris-ai-using-a-genetic-algorithm/>. [Accessed: 23- Oct- 2015].
- Local Maxima, 'The problem of local maxima', 2015. [Online]. Available: <http://www2.denizyuret.com/pub/aitr1569/node6.html>. [Accessed: 23- Oct- 2015].
- L. Yiyuan, 'Tetris AI - The (Near) Perfect Bot', *Code My Road*, 2013. [Online]. Available: <https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/>. [Accessed: 23- Oct- 2015].
- Meatfighter.com, 'Applying Artificial Intelligence to Nintendo Tetris', 2015. [Online]. Available: http://meatfighter.com/nintendotetrisai/#The_Algorithm. [Accessed: 23- Oct- 2015].

User Guide

<u>To run on a PC:</u>

The Tetris Artificial Intelligence Bot runs from inside a pre compiled program. To run simply click on the .exe file labelled Tetris AI Bot.

<u>Using the system:</u>

After opening the .exe file click the ‘New Game’ button to see program in action and watch the bot evolve before your eyes.

The slider bar labelled ‘Game Speed’ controls the frame rate that the game is drawn on screen, you can slide this bar up or down to the desired speed, this does not increase the games difficulty.

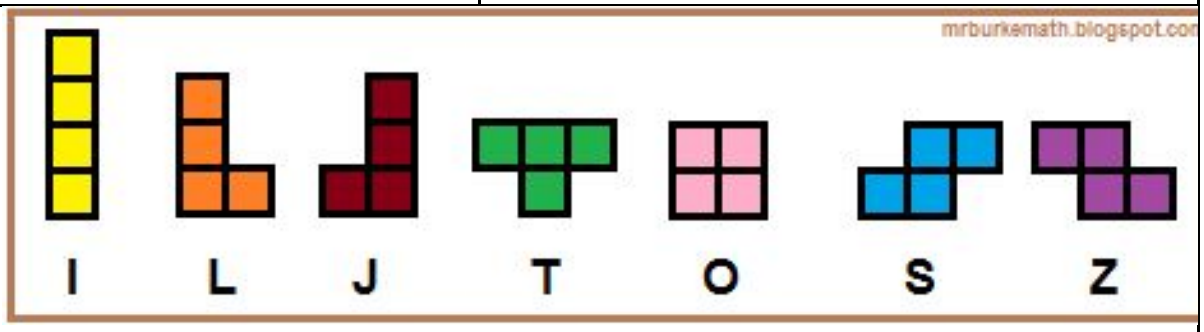
The ‘Pause’ button will pause the Tetris Bots, and clicking on it again will restart the Bots.

The ‘Show Graph’ button will show the Best Score of that generations round, the number of generations passed since the first Bot (Adam) and the most number of lines cleared for that generation.

The ‘Hide Graph’ Button will hide the graph.

Appendix A: Standard and Specifications of the Custom Built Tetris Game Engine

Tetris Game Specifications	
Game Grid Specifications	• 10x22, with the first two game rows hidden.
Game Play	1. Create a horizontal line unit of 10 blocks without gaps to clear a row. 2. When a certain number of lines has been cleared, the game enters a new level. 3. Each level causes the tetromino's to fall faster. 4. The game ends when the stack of tetromino's reaches the top and no new tetromino's are able to enter.
Tetromino Specifications/Scoring	1. All tetromino's are capable of single and double clears. 2. I, J and L can clear triples. 3. Only I can clear four lines simultaneously, this move is known as a 'Tetris'. 4. A single line clear is worth 100 points. 5. A 'Tetris' is worth 800 points.



Appendix B: Code

Due to the amount of code present in this project, the code has been moved to a separate folder. Please see the ‘Source Code’ File for a complete folder of classes.