Chapter 1 | Introduction

1.1 Tetris

Tetris is a widely popular tile-matching puzzle game originally designed and programmed by Alexey Leonidovich Pajitnov in 1984 [1]. The name "*tetris*" has been derived from the Greek word tetra meaning four as all the pieces in the game are made of four segments. The game design was inspired from Pajitnov's favorite recreational mathematical puzzle and a very popular tile-matching puzzle in USSR called Pentominos, which consists of pieces made from five segments, and had 12 different permutations of these 5-segment pieces. To reduce the complexity of the game for masses, Pajitnov switched to four segment pieces, which resulted in 7 permutations of 4-segment pieces, and named the game TETRIS. It was one of the first entertainment software to be exported from the Soviet Union to the United States, and within a span of 15 years, was crowned the title of the "Greatest Game of All Time", by the Electronic Gaming Monthly magazine in their 100th issue in 1997. In 2019, Tetris occupies the 7th place in the "Top 100 Greatest Video Games of All Time" list published by IGN Entertainment Inc.

Fig 1.1 The creator of Tetris, Alexey Pajitnov

Fig 1.2 All tetrominos used in Tetris

1.1.1 The rules of the game

The game of tetris is played on a 2-dimensional board, generally consisting of 20x10 cells, where each cell can either be empty or full. As mentioned above the game consists of 7 different 4-segment pieces which are known as tetrominos, which occupy cell space on the board. The game comprises of tetromino pieces falling from the ceiling of the board one by one. These pieces keep moving downward on their own, and they stop when the piece has either reached the bottom of the board or has hit another cell that is full. The objective of the player is to score maximum by clearing lines on the board. When every cell in a row is full, that row gets cleared, i.e., all the cells in that row are emptied, all rows above the cleared row are shifted downwards and a row is added to the top of the board. The player can rotate pieces and move them along the horizontal axis to place pieces as such to clear more rows or perform optimal winning moves. The game ends when the board is so full that the next piece falling from the ceiling does not have any space to enter the board, and as such cannot be placed on it without overflowing the board [2]. It should also be noted that a player can clear multiple lines in a single move, the maximum being 4 lines due to the I shaped piece, which is also referred to as a tetris in tetris' terminology.

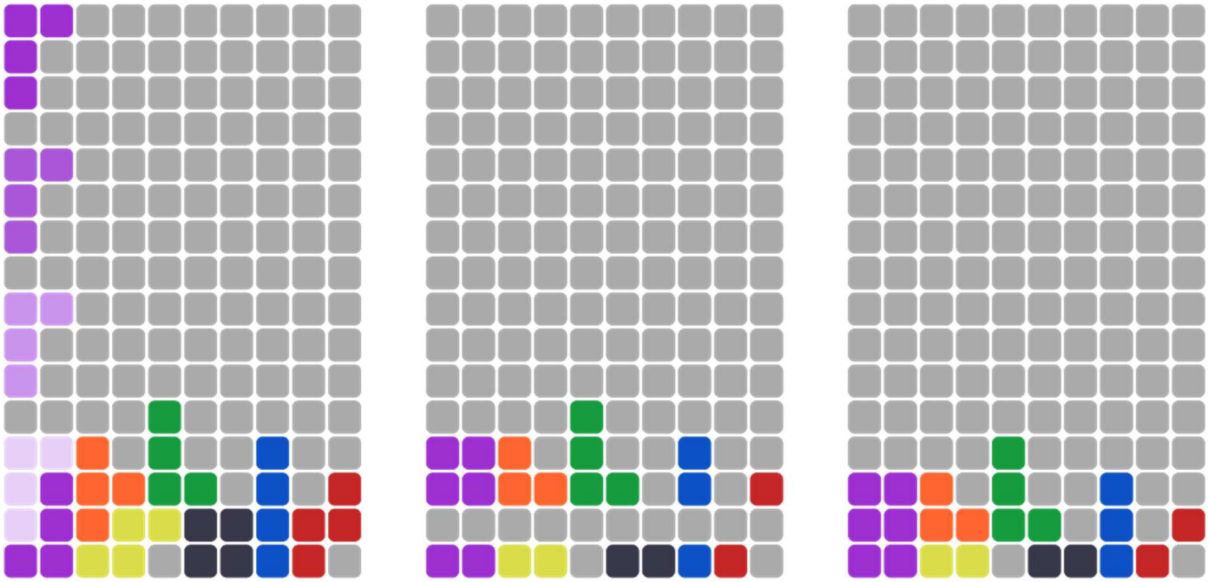


Fig 1.3 The process of clearing a row from the tetris board

The traditional scoring system rewards a player 1 point each time a piece moves down on the board, 40 points for clearing 1 line, 100 points for 2 lines, 300 points for 3 lines and 1200 points for 4 lines. Although this is the general standard which commercially available versions of tetris software implement but, in modern times, due to tetris being a highly

researched topic for artificial intelligence, a more standardized scoring system has been developed which rewards a player 1 point for each line cleared. This helps compare results directly from the score, without having to compute abilities of players from their score. In our project, we have implemented the modern scoring system so that comparison between AI agents learning to play the game becomes intuitive.

1.1.2 Mathematics behind the Tetris Game Board

Since the game board has 200 cells in it, where all cells can have only two states, empty or full, the maximum number of board configurations is $\mathbf{2^{200}}$. But, due to the fact that each piece adds 4 full cells to the board and each line cleared removes 10 pieces from the board, the number of full cells always stays even on the board. This leads to an approximate of $\mathbf{2^{199}}$ total board configurations. But, we also need to keep in mind that in these possible configurations are boards which consist of filled rows, which essentially don't stay on the board as they are removed right after the move that fills all cells of a row, and boards which consist of non-empty rows above empty rows, which essentially can't take place in the tetris setting as the vertical movement of tetris pieces is restricted to only one direction, i.e., downward. This brings us to an estimate of roughly **3.74%** less than $\mathbf{2^{199}}$ possible board configurations. It is also important to note that there are roughly $\mathbf{7x2^{199}}$ possible game states considering the current falling piece on the different board configurations [2]. This leads to the result that although the best move can be directly computed for a piece on the board iteratively, it is computationally very expensive and is not the best approach to get the best move. This led to tetris being a center of attention for AI researchers to implement different approaches to navigate the search space so that computational expenses can be minimized.

It is also interesting to note that Tetris is a **NP-complete problem** [3] and although, it might seem that the game can go on indefinitely if moves aren't limited, but that is not the case and it has been proven that *every tetris game ends with a probability of 1*, because of the presence of the Z and S shaped pieces [4]. Burgiel establishes two important principles in the gameplay of tetris,

(i) If a game has only Z and S shaped pieces appearing alternatively, then a maximum of **69,660** such pieces can only be placed on the board before the game ends inevitably

(ii) A normal tetris game must end after **127,200** Z and S shaped pieces have been placed on the board.

All these features of the game make it an *interesting optimization problem* and an area of interest for *AI research*.

1.2 The Objective of this Project

The goal of this project is to explore the applications of genetic algorithms in the game of tetris, understand how genetic algorithms evolve, and visualize the evolution of genetic agents. To fulfil our goal, the objective is to create a platform for evolving, testing, trying and analyzing different tetris agents.

We achieve this by implementing a tetris controller for genetic agents to evolve on, an extremely modifiable genetic algorithm which can vary most of its parameters based on what needs to be tested and analyzed, and an analysis module which would take data collected from the agents and try to make sense of that data in terms of statistics and visualization.

A detailed discussion about genetic algorithms and data visualization can be found in the *Methodology Chapter* of this report.

Chapter 2 | Literature Review

In this project, we have implemented genetic algorithms to evolve tetris agents. Every related work that implements some sort of genetic algorithms to evolve tetris agents, use a controller and an agent algorithm, which are somewhat similar to ours. The basic differences come in the number or type of the weights used to evaluate their agents, and their optimization methods. The below tables summarize what kind of weights, controllers, and optimization methods different studies have used.

AUTHOR, YEAR	OPTIMIZATION METHOD	CONTROLLER TYPE
Lippmann et. al., 1993	N/A	1
Tsitsiklis and van Roy, 1996	Value Iteration	1
Bertsekas and Tsitsiklis, 1996	Lambda Policy Iteration	1
Kakade, 2001	Natural Policy Gradient	1
Lagoudakis et. al., 2002	Least-Squares Policy Iteration	1
Dellacherie, 2003	Manual	1
Fahey, 2003	Manual	2
Ramon and Driessens, 2004	RRL-KBR	1
Llima, 2005	Genetic Algorithm	1
Bohm et. al., 2005	Genetic Algorithm	2
Farias and van Roy, 2006	Linear Programming	1
Szita and L orincz, 2006	Noisy-Cross Entropy	1
Thiery and Scherrer, 2009	Noisy-Cross Entropy	1
Langenhoven, 2010	Particle Swarm Optimizer	1
Scherrer, 2013	Lambda Policy Iteration	1
Our Project, 2019	**Genetic Algorithm**	**1, 2**

Table 2.1 A summary of the controller-types and optimization methods used by different studies

	FEATURE	Lippmann et. al., 1993	Tsitsiklis and van Roy, 1996	Bertsekas and Tsitsiklis, 1996	Kakade, 2001	Lagoudakis et. al., 2002	Dellacherie (Fahey, 2003)	Fahey, 2003	Ramon and Driessens, 2004	Llima, 2005	Bohm et. al., 2005	Farias and van Roy, 2006	Szita and L orincz, 2006	Thiery and Scherrer, 2009	Langenhoven, 2010	Scherrer, 2013	**Our Project, 2019**
1	Jags	x															**x**
2	Max Height	x	x	x	x	x			x	x	x	x	x		x	x	**x**
3	Holes	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	**x**
4	Column Height			x	x				x			x	x			x	
5	Column Difference			x	x				x			x	x			x	
6	Landing Height						x			x	x			x	x		
7	Cell Transitions									x							
8	Deep Wells									x							
9	Embedded Holes									x							**x**
10	Height Differences					x											
11	Mean Height					x			x								
12	Δ Max Height					x											
13	Δ Holes					x											
14	Δ Height Differences					x											**x**
15	Δ Mean Height					x											
16	Removed Lines	x				x		x			x				x		**x**
17	Height Weighted Cells							x			x				x		**x**
18	Wells							x	x		x				x		**x**
19	Full Cells							x			x				x		**x**
20	Eroded Piece Cells						x							x	x		
21	Row Transitions						x				x			x	x		
22	Column Transitions						x				x			x	x		
23	Cumulative Wells						x							x			
24	Min Height								x								
25	Max - Mean Height								x								**x**
26	Mean - Min Height								x								
27	Mean Hole Depth								x								
28	Max Height Difference										x				x		
29	Adjacent Column Holes										x				x		
30	Max Well Depth										x				x		
31	Hole Depth													x			
32	Rows with Holes													x			
33	Pattern Diversity													x			

Table 2.2 A summary of different features used in different studies

Lipmann et. al. [5] had a very different approach towards making an AI tetris agent which worked with the combination of pattern classification with neural networks, machine learning and statistics. The neural network compared two moves, and this neural network was used in association with a preference network that would learn to play the game by observing a human player. The better the moves the human player would play, the better the framework was going to be able to perform. The scores reported were understandably low, as the framework would never play better than the human player which it mimicked while learning. They reported a score of 18 after 50 pieces had fallen.

Dellacherie and Fahey's Tetris controllers [2] (one-piece and two-piece controllers, respectively) are notable for performing extremely well despite being optimized manually, i.e. they used no optimization method to tune the relative weights of the feature functions. Dellacherie also seems to be the only one who has considered the original implementation of Tetris without the usual simplifications made. This is a significantly harder domain, as it does not permit all moves otherwise considered in the typically simplified domain [6]. Several controllers have been realized using methods of reinforcement learning. Methods include Value Iteration [7], λ-Policy Iteration [8, 9], Natural Policy Gradient [10], Least-Squares Policy Iteration [11], RRL-KBR [12] and Linear Programming [13]. None of the controllers optimized using these methods perform particularly well. This may be due to the nature of RL methods, in which the weights are tuned such that the fitness function approximates the optimal expected score from any given state.

Some controllers use more general purpose optimization methods. Llima [14] created a controller of which the weights were tuned by a genetic algorithm. He reports using a population of 50 individuals (corresponding to a set of coefficients) over 18 generations. The evolution took 500 CPU-hours, distributed over 20 workstations. The controller obtained a score of about 50,000 on average.

Böhm [15] also utilized a genetic algorithm but unfortunately does not report any averaged results due to running time reasons. Böhm also reports the different kinds of rating functions that are used and mentions the reasons for using one over the other. He writes about three major ways of computing ratings:

- Linear Rating Function $R_l(b) = \sum_{i=1}^{n} w_i r_i(b)$
- Exponential Rating Function $R_e(b) = \sum_{i=1}^{n} w_i r_i(b)^{e_i}$
- Exponential Rating Function with displacement $R_d(b) = \sum_{i=1}^{n} w_i |r_i(b) - d_i|^{e_i}$

Szita and Lörincz [16] created a controller using Noisy Cross-Entropy. This is where it gets interesting, as they report obtaining a score of 350,000. Thiery and Scherrer [17] improved on this approach by using the feature functions of Dellacherie along with two of their own. They report obtaining a striking score of 35,000,000, putting them on the top of the list of all time high scores by AI agents in tetris. It is important to note here, that these approaches are free from the limitation of moves on them, which is why obtaining such high scores is even a possibility for these agents.

Lagenhoven used general purpose optimization methods [18], which applied Particle Swarm Optimization to train a neural network. Initially, he tried using a network with hidden nodes, but experiments showed that zero hidden nodes yielded the highest score. The network without hidden nodes simply amounted to a weighted sum of each feature function.

Funk et. al. [19] discusses some newer techniques like Q-learning, a reinforcement learning technique, free from environment modelling, and Auto-encoders, that translate higher-dimensional data into lower-dimensional data, that can be used for creating AI agents for tetris. Q-learning in particular, essentially maps all possible actions to rewards, in much the same way, that genetic algorithms evaluate their genomes. Auto-encoders can be used in conjunction with genetic algorithms and other optimization techniques like stochastic gradient and steepest gradient descent to find optimal solutions to the tetris problem. Stevens et al. [20] having used Q-learning as a technique for optimization reports average scores of around 200 lines for their best agent. They illustrate the different techniques that were used in conjunction with Q-learning, to achieve their results.

Chiroma et. al. [21] reports a unique study whereby they conduct experiments to obtain the correlation between crossover probability and mutation probability. They observed that crossover probabilities were positively associated with the use of mutation probabilities in the implementation of genetic algorithms.

An interesting study, conducted by Czarn et. al., [22] analyses genetic algorithms with the help of statistics. They perform a statistical method called Analysis of Variance (ANOVA) on genetic algorithms to recommend balanced parameter settings for genetic algorithms. Their recommendations suggest using around a 50-100 population size with crossover rates ranging from around 0.5-0.9 and mutation rates ranging from 0.04 – 0.24. It is interesting to see how they explain that too large population sizes and imbalanced parameter settings can often make the genetic algorithm deviate from converging towards the optimal solution upon evolution.

Rollinson et. al. [23] presents a different approach to using genetic algorithms in the context of developing tetris agents by using optimization techniques to converge to optimal solutions faster. They use Nelder-Mead optimization and report average scores of around 51,000.

Chapter 3 | Project Planning and Timeline

3.1 Overview

Our entire project was divided into 5 phases and spanned over a period of 9 weeks. During the project planning process, we have heavily incorporated agile principles as we feel that one of the best productive ways to develop software is using agile principles. For better communication between the stakeholders of the project (essentially me and my project partner), we had taken up the concept of daily and session based stand-ups where we would first discuss about what we have done, what we need to do ahead, and what could be certain dependencies that are stopping us from doing something. We also borrowed sprint cycles from agile, and always made sure that our sprint cycles were never longer than 2 weeks. Although we didn't implement a backlog in the truest sense of agile backlogs, we had our own version of backlogs implemented. We were overwhelmed with the progress of our application development process by using agile principles.

3.2 Phase I

We began the project planning process with a Goal Setting and Task Mapping session, which led to establishing the why and what of the project. We also covered the first and second phase task mappings for the project during this session, whereby we got a brief understanding of what the minimum viable prototype could be, and what could be built directly on top of the prototype to give us the final software. Tasks of researching, putting forth guidelines for data exchange, risk analysis were discussed during this session.

3.3 Phase II

From here, we moved on to phase II, prototyping, which essentially acted as a skeleton during the final development process. Our prototype was based on a web application, since we did not want to initially deal with the complexities of developing desktop applications. We developed a basic tetris controller during this phase which could be played by a human, and a genetic agent.

3.4 Phase III

Then, during phase III, we started visualizing what the final software could look like and quickly made a User Interface for the same. We also got a brief understanding of the desktop architecture, and how IPC (inter process communication) acts as a way of data exchange in this setting. We then started plugging in the game and AI parts from the prototype directly into the final software and made the prototype into a desktop app. We then added additional features into it, like data download, agent upload, data visualization and data analytics. Since human playability did not matter to us in the final iteration of the software, we removed that feature, to make things a little cleaner, and oriented towards our goals and objectives.

3.5 Phase IV

We then moved on to the data collection and analysis phase of the project during which time we exhaustively first tested the software for any possible bugs, and fixed the same during this period. We ran simulations to collect the data and visualize them on the fly within our application. We had invested around 120 CPU hours on simulations during this phase. We also converted our results into CSVs for better analysis with standard spreadsheet software.

3.6 Phase V

We finally moved on to phase V of the project, which dealt primarily with our reporting and documentation. We went thoroughly through our research notes, and other related work, and compiled the project report during this phase. We also created any other assisting material like images, tables, and charts for inclusion in the report during this phase.

TASK BACKLOG NUMBER	TASK TITLE	TASK OWNER	PHASE ONE					PHASE TWO										PHASE THREE														PHASE FOUR							PHASE FIVE						
			WEEK 1					WEEK 2					WEEK 3					WEEK 4					WEEK 5					WEEK 6					WEEK 7					WEEK 8					WEEK 9				
			M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F
1	**Project Conception and Initiation**																																												
1.1	Goal Setting	B Swaika																																										
1.2	Research	B Swaika																																										
1.3	Guidelines	B Swaika																																										
1.4	Risk Analysis	S Jain																																										
2	**Project Prototyping**																																											
2.1	Web Architechture	S Jain																																										
2.2	Basic Controller Developement	B Swaika																																										
2.3	Genetic Algorithms Developement	B Swaika																																										
2.4	Integration of GAs with the Basic Controller	B Swaika																																										
2.5	Debugging and testing	S Jain																																										
3	**Project Developement**																																											
3.1	Desktop Architechture	B Swaika																																										
3.2	UI UX Designs	B Swaika																																										
3.3	UI state Developements	S Jain																																										
3.4	Plugging the logic for the controller and the GA from the prototype	B Swaika																																										
3.5	Setting Up Visualisation Parameters	B Swaika																																										
3.6	Developing the Analytics module	B Swaika, S Jain																																										
4	**Project Performance / Monitoring**																																											
4.1	Application Testing	S Jain																																										
4.2	Simulation Runs	B Swaika																																										
4.3	Visualizing Simulation Runs	B Swaika																																										
4.4	Best agent plays the game	S Jain																																										
5	**Project Reporting**																																											
5.1	Literature Review	S Jain, B Swaika																																										
5.2	Assisting Materials Creation	B Swaika																																										
5.3	Report Writing	B Swaika																																										

Table 3.1 Gantt Chart of Project

Chapter 4 | Methodology

4.1 Tetris Controller

The ideal and completely hypothetical strategy for playing Tetris is that where each possible successor state is considered for an infinite amount of levels. By averaging scores obtained through each possible game play and backtracking the results, the controller would be able to perfectly place any falling tetromino. However, due to huge computational requirements, this may never be possible. Instead, a heuristic approach is chosen and employed by all controllers.

Controllers that utilize the information of the board and the current falling tetromino only are one-piece controllers. When the next tetromino is known in advance and that information is also used, the controller is called a two-piece controller. A two-piece controller will utilize more information than a one-piece controller in its decision-making and can make more educated guesses of what's a sensible placement. Two-piece controllers generally perform better than one-piece controllers, but take considerably longer time to finish. Similarly, there could be possible implementations of n-piece controllers depending upon how much deep we want to go in exploring possible successor game states. In tetris' terminology, these multi-piece controllers are called lookahead controllers as they can look ahead of the current piece and generate more accurate best moves. But the tradeoff comes in terms of computational run time, due to the generation of multi-level trees for each move.

To decide the placement of a tetromino the controller uses the information about the current piece, and lookahead pieces (if available) to create a tree of all possible game states arising from the pieces. All these game states are then rated by a rating function and successor state ratings are added to predecessor states to get the best move possible from the entire tree. The best possible move is then applied to the current piece. This characteristic reduces the problem of playing tetris into creating a good rating function, which can be achieved easily using some game-related mathematics.

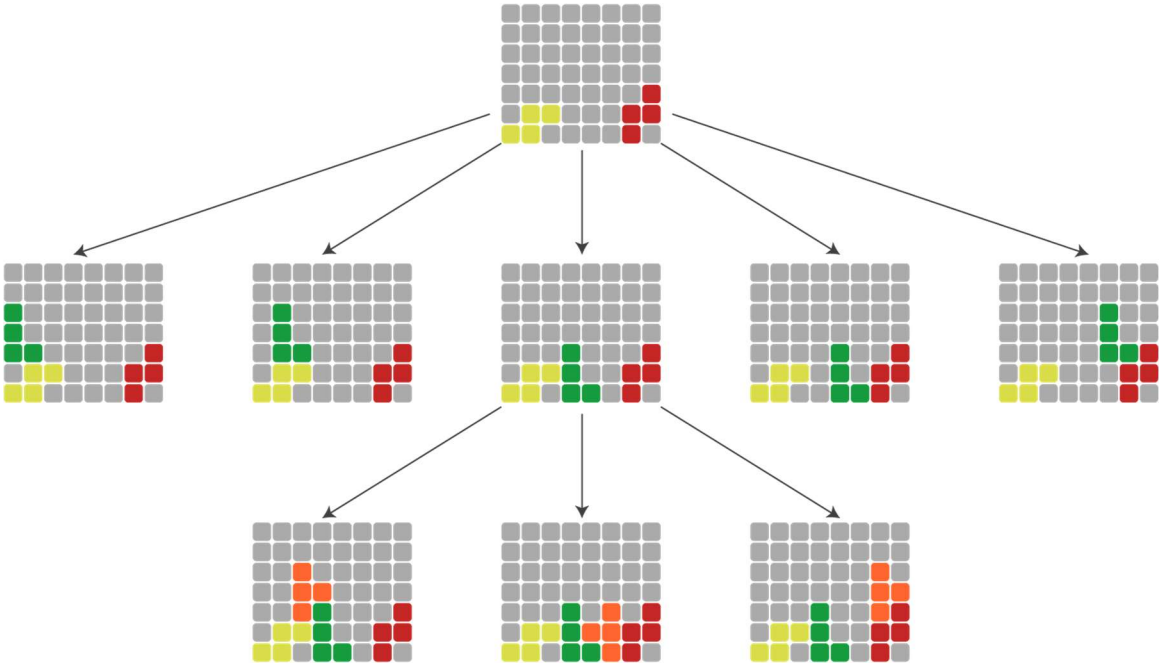


Fig 4.1 A partial illustration of a tree of states constructed by a two-piece controller in a game of tetris. The root level illustrates the current board state, the intermediate level illustrates the permutations of placing the current shape, and the leaf level illustrates the permutations of placing the next upcoming shape.

A rating function is generally the weighted sum of individual feature functions. This is called a linear rating function. As mentioned above in the literature review section, Böhm has used not just linear ratings but also exponential ratings and displaced ratings, which often outperform linear rating functions, but implementing exponential and displaced ratings require more evolution on the genetic algorithm's part, and thus was outside the scope of this project as we wanted our evolution scheme to deal only with the board feature vectors, and not with exponent and displacement vectors.

By weighting a feature positively or negatively, the controller can encourage or discourage, respectively, formations of certain features on the board during a game. If these features affect the duration of the game and help to avoid ending it, they will contribute to an overall performance improvement. The choice of feature functions and their relative weights is what constitutes a tetris strategy.

An example of a typical feature function for a tetris game, is relativeHeight. Relative height is essentially the difference between the maximum height of a full cell and the minimum height of a full cell. It is a negative feature, as we would always like to minimize the difference between the heights due to the fact, that if the heights are balanced then there are more chances of the next piece clearing one or multiple row(s). On the contrary, if the relative height is high, then the board essentially starts getting higher on one side due to the stacking of pieces on top of each other, leading to the possibility of the game ending sooner. A few important feature functions have already been mentioned in the *Literature Review Chapter*.

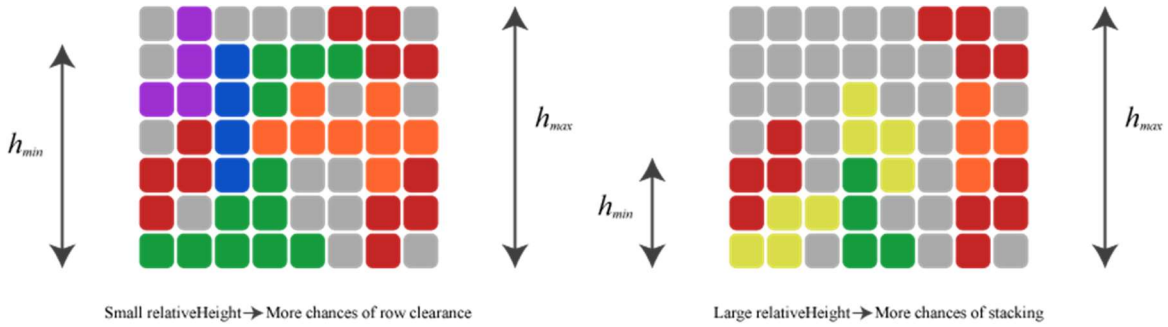


Fig 4.2 Partial illustration depicting the *relativeHeight* feature on a tetris board. The left board has a lesser relative height and thus has more chances of clearing rows, whereas the right board stacks up on a side making the game end sooner.

For our controller, we have kept options for it being either one-piece or two-piece, so that we could actually compare their performances. We have also implemented around 10 feature functions with 6 of them being recommended features, and thus not optional, and 4 more optional features, for comparing the effect of these features on the overall rating of moves which would directly impact the performance of the agent. Our features are listed below:

- rowsCleared
- cumulativeHeight
- holes
- blocks (*optional*)
- wells (*optional*)
- weightedHeight
- relativeHeight
- roughness
- weightedBlocks (*optional*)
- blockades (*optional*)

4.2 Genetic Algorithms

Genetic algorithms are a subset of evolutionary computation, a field of computing that is largely inspired by Charles Darwin's theory of evolution [24] and his explanation of biological diversity in nature and the underlying mechanisms that they are a result of. He explained that in a limited environment that can only host a certain number of individuals and where each individual has an instinctual desire to reproduce, natural selection will inevitably occur because of the competition that arises. Genetic Algorithms are extensively used as search methods to navigate seemingly infinite search spaces and compute the best result from that search space.

Genetic Algorithms encode the decision variables of a search problem into finite-length strings of alphabets of certain cardinality. The strings which are candidate solutions to the search problem are referred to as chromosomes, the alphabets are referred to as genes and the values of genes are called alleles. For example, in a problem such as the travelling salesman problem, a chromosome represents a route, and a gene may represent a city. In contrast to traditional optimization techniques, Genetic Algorithms work with the coding of parameters, rather than the parameters themselves. To evolve good solutions and to implement natural selection, we need a measure for distinguishing good solutions from bad solutions. The measure could be an objective function that is a mathematical model or a computer simulation, or it can be a subjective function where humans choose better solutions over worse ones. In essence, the fitness measure must determine a candidate solution's relative fitness, which will subsequently be used by the GA to guide the evolution of good solutions [25].

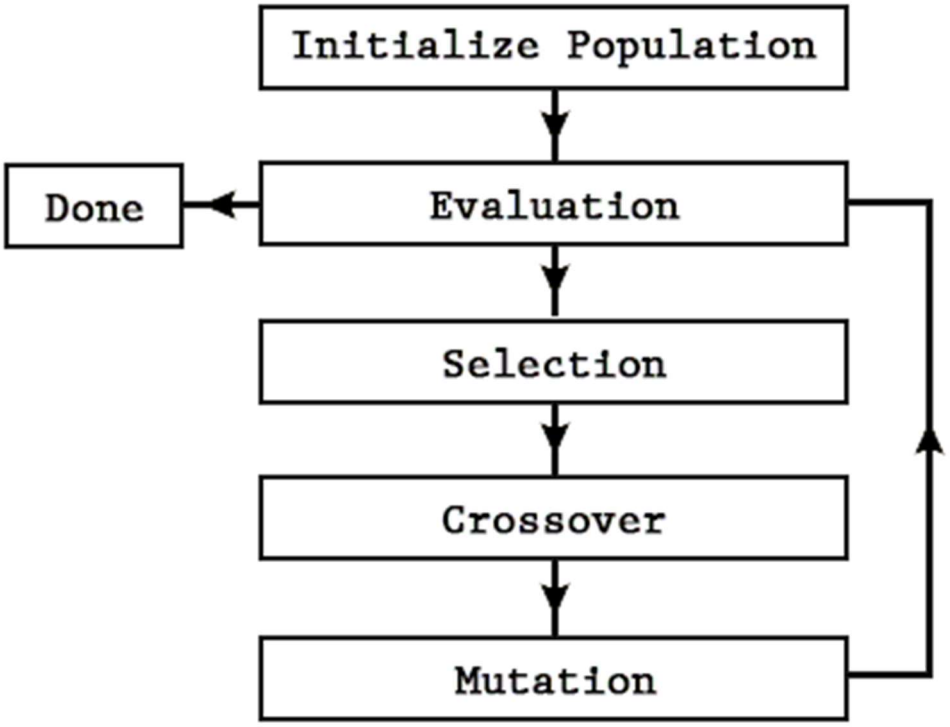


Fig 4.3 A basic flowchart for a Genetic Algorithm

[Source: https://becominghuman.ai/genetic-algorithm-for-reinforcement-learning-a38a5612c4dc **]**

One example of a practical problem where a genetic algorithm has been successfully applied is the design of NASA's ST5 spacecraft antenna, by Globus et al. [26]. The evolved antenna was superior to the conventional antenna design in regard to power consumption, fabrication time, complexity and performance. Another example of a similarly successful employment of an evolutionary algorithm is the satellite dish boom holder by Keane and Brown [27].

To simply the explanation of Sastry et. al. above, Genetic Algorithms function with a few fundamentals described below:

- **Generations of populations**: The main objective of Genetic Algorithms is to evolve generations of genome populations to find the best possible candidate solution. This is done using Darwinian principles of natural selection, reproduction, and mutation and evaluation functions.

• **Genomes**: These are encoded representations of a possible solution to the problem at hand. They are theoretically represented as bit strings but the representation is most commonly problem – specific

• **Genes**: Each of the bits of the genome which deals with a feature aspect of the problem, is called a gene. These are the building blocks of a genome

• **Evaluation Function**: This is typically a function that takes an individual genome as the input and processes the genome's genes against the problem features, to rate the performance of that genome. It outputs a rating that can be represented specifically for the problem.

• **Natural Selection**: It is inspired from Darwinian principles, and often philosophically quoted as "Survival of the fittest". This dictates the fact that in a generation where each genome has a natural tendency to reproduce to create offsprings, there arises competition between suitable mates for the mating process, and only the fittest that survive do actually get to reproduce. In algorithmic terms, before evolving our current generation of genomes, we would optimally want to select only a portion of the generation, which comprises of the fittest genomes, for reproduction. This is also called selection.

• **Reproduction**: This is a universal biological phenomenon. During reproduction, two suitable mates come together to mate and create new offsprings, as a result of which they pass their genes to the child genome. This process in genetic algorithms is typically broken down into two sub processes drawing inspiration from their real world counterparts.

    ◦ **Crossover**: This process is the actual reproduction process whereby two parents give birth to a child. This can be of several different types, depending on the algorithm used to implement it, like

        ▪ *k-point crossover*, which deals with breaking parent genomes into k-parts and randomly selecting a part from each parent to generate the child

        ▪ *uniform crossover*, which deals with each gene being randomly selected from either of the parents

        ▪ *average crossover*, which deals with each gene in the child being generated by averaging the gene values from both the parents

    ◦ **Mutation**: It is a part of reproduction where a child gets a trait or gene which comes from neither parents. In genetic algorithms, this is usually

implemented with a mutation probability parameter, that sets how much mutation should be present during the evolution of a generation.

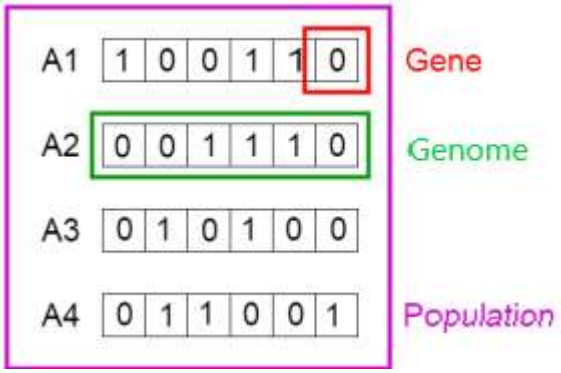


Fig 4.4 Basic Terminology of Genetic Algorithms

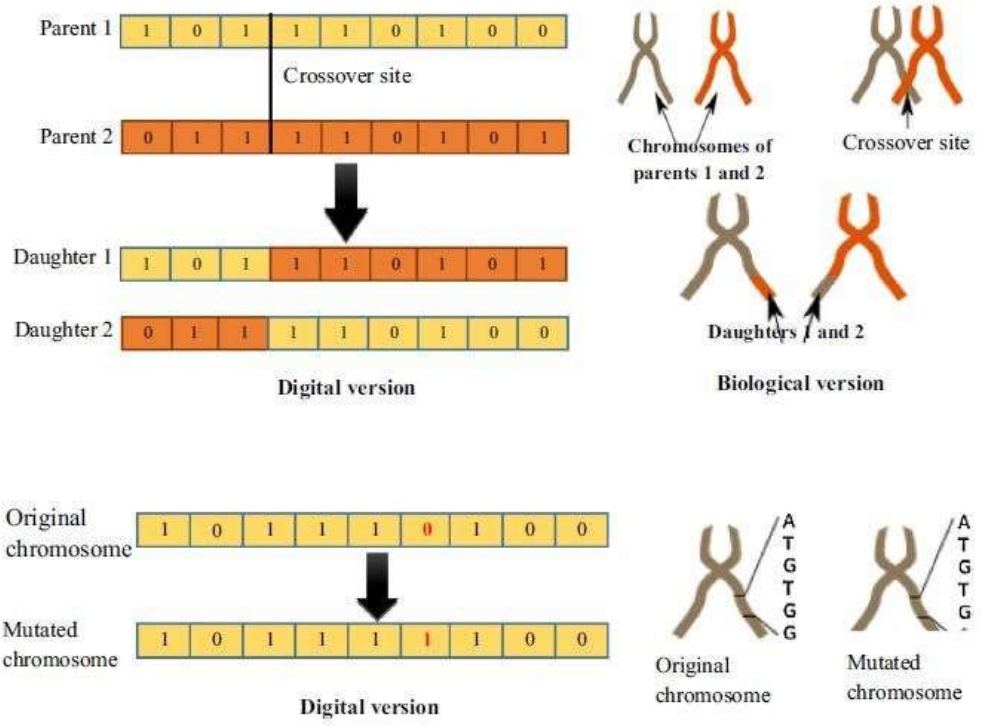


Fig 4.5 Basic Operations of a Genetic Algorithm, CROSSOVER (above) and MUTATION (below)

[Source: https://www.researchgate.net/figure/Genetic-algorithms-operators_fig2_317297833 **]**

In our implementation, we have kept the genetic algorithm extremely open-ended, so that trying and testing values for optimal solutions based on different settings is possible. We have represented genomes as an object having genes as key value pairs, and gene values being represented as floating point numbers. We have used a linear evaluation function as mentioned earlier in the controller section. We have implemented both uniform and average crossovers. For selection, we have given the user a choice between three percentages, to select the fittest. We have also implemented elitism, which is a property of genetic algorithms to carry over a portion of the fittest genomes from one generation to the next. In our case, only 1 elite progresses from the current generation to the next. Although we could have had enabled open-endedness for elitism as well by giving the user a choice, but we have identified this as an opportunity for future work on the project, as well as incorporating an exhaustive list of feature functions from existing studies.

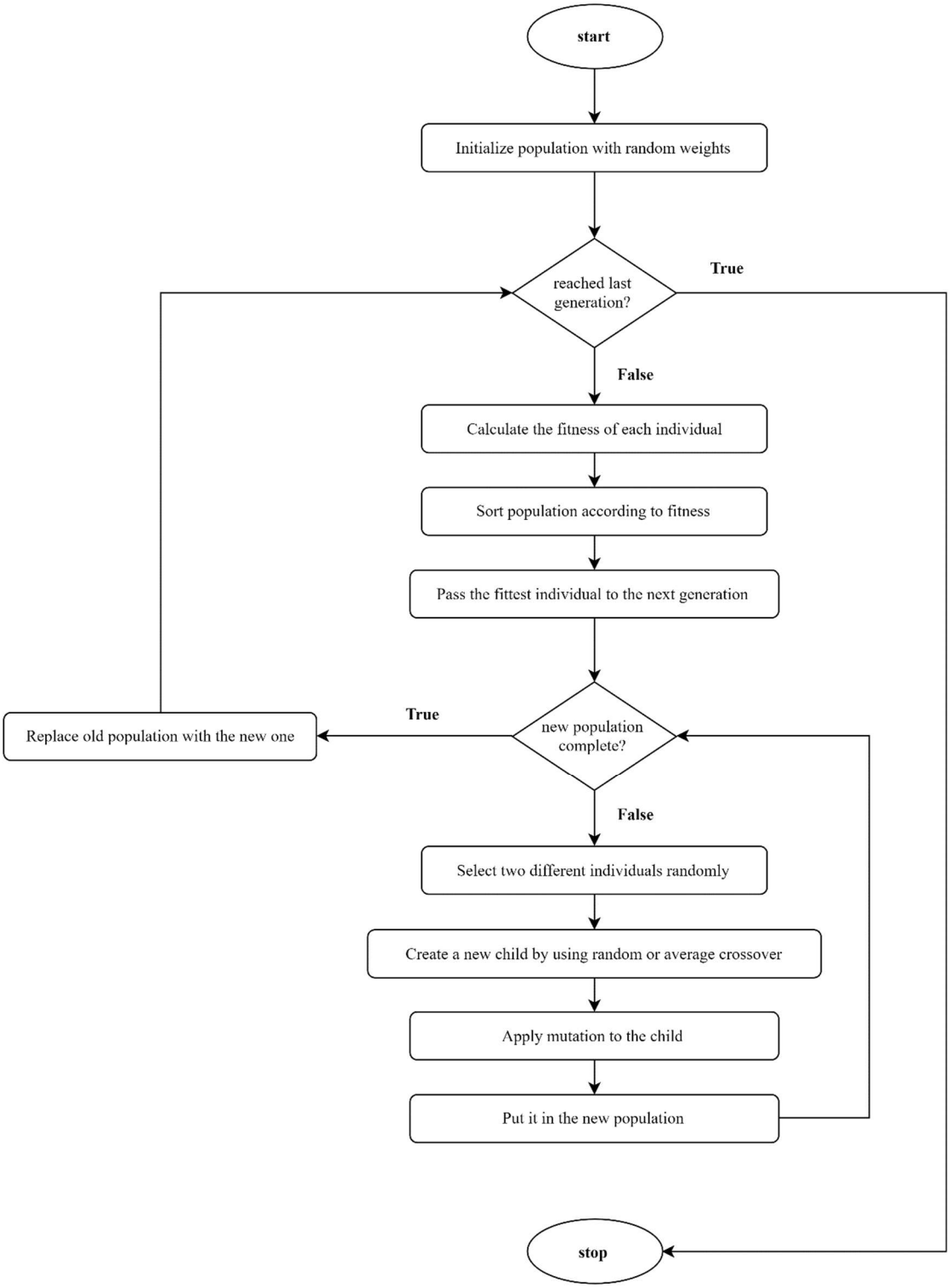


Fig 4.6 The genetic algorithm flowchart as implemented in our project.

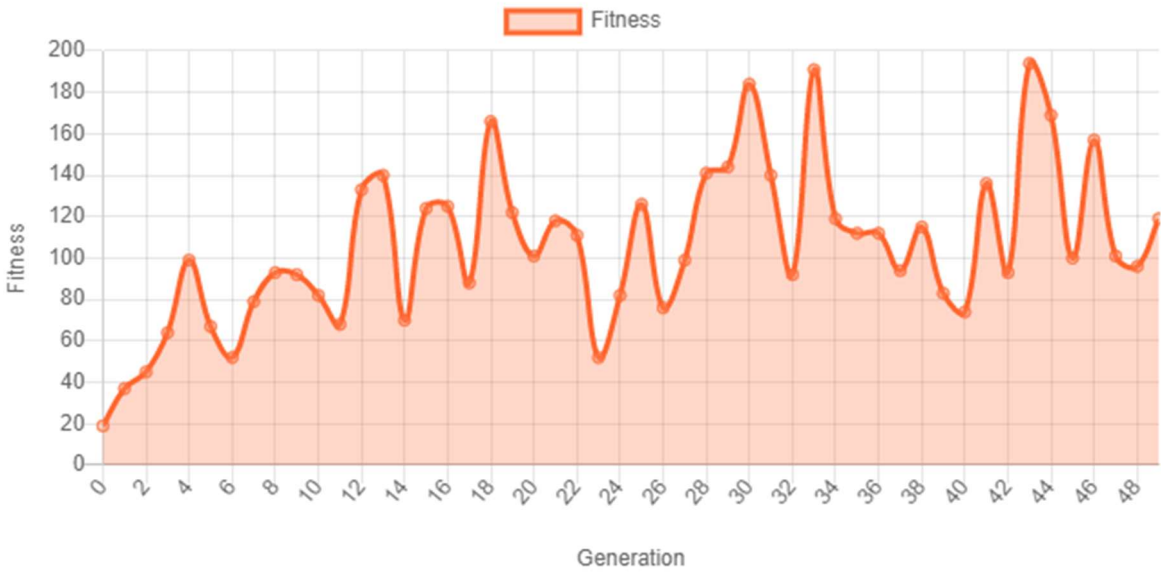
4.3 Data Visualization

With the data that we collect from the genetic algorithm's evolution, we can hardly make any sense of it just by looking at those floating point numbers. Our eyes are automatically drawn to colors and symbols, over plain numbers which often fatigue our brain. We can quickly identify red from blue, square from circle.

Data visualization is a form of visual art that communicates various amounts of information, and the connections between the various elements through simple charts and images [28]. We can quickly identify trends and outliers from visualizations. Proper visualizations can be very effective in both storing data and making it simple to search through for meaningful information, which can be interpreted from emerging patterns or even specific values, elements or even relationships that become easier to understand or follow when properly represented.

We have emphasized on data visualization to simplify what's going on for the end-user of this application. We have implemented the visualizations, in a way that it can convey a precise amount of information about the algorithm with the help of very simple and easy to understand charts. Some of the visualizations come from processed data, which forms the base of the data analytics and statistical operations that we have performed on the data in our project.

To take a deeper dive, let's first start with the analytics section that would then easily build up to the visualization components. To get an overall view from generation to generation, we have used the statistical calculations of mean and variance for the distribution, which give us a measure of the average fitness of each generation and the variability of the fitnesses of individuals in each generation respectively. We also have calculated the correlations of each gene used in the algorithm against the fitness to understand which features are negative features and which are positive features, specific to the application of the algorithm. Our visualizations have the above components represented as bar charts and radar charts respectively, apart from which we also display some other charts like line charts to understand how the elites of each generation have performed and how each genome has performed amongst generations. We also display bubble charts to display the gene values of the best fit elite during the entire evolution process.



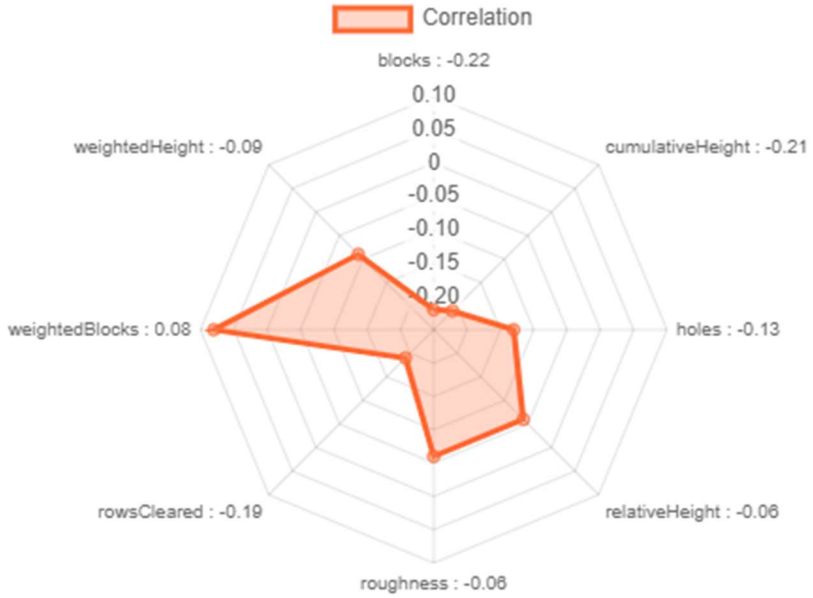


Fig 4.7 Some sample visualizations from our application

4.4 Project Specifications

4.4.1 Technology Stack

The project is implemented as a Desktop Application using a very modern and robust tech stack. We use javascript as the primary language for our project, based on a chromium-based framework electron JS which opens javascript to native OS specific APIs for developing cross-platform Desktop Applications. Electron JS is a widely used industry standard framework and applications like Slack, Skype and Discord are all built on the Electron framework. We use SASS for styling and HTML for structuring the components of the application. Npm has been used as a dependency manager, which again is an industry-standard package management tool for javascript applications. Chart JS has been used for the visualization module, which allows us to create responsive, and beautiful looking charts. It is important to note that the project does not use any external dependency for implementing the genetic algorithm or the game, and that we have not gone into in-depth explorations regarding the deployment of the app.

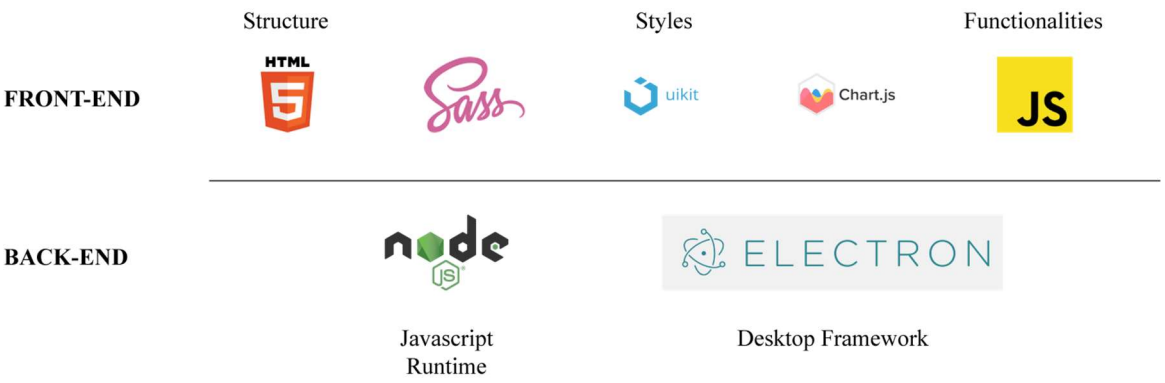


Fig 4.8 Our Technology Stack and Architecture

4.4.2 Software Model

We have used a mixture of *throwaway* and *evolutionary prototyping model*, whereby in the early phases we had prototyped a web version of the application, which was eventually discarded, but the core logic was taken from the prototype and features were added to it to evolve it into the final application. We found this model convenient for us, because, we didn't want to deal with the complexities of desktop architecture up front in the early stages of the development cycle. Web architecture is fairly simpler and uniform compared to the desktop architecture which is a little more complex when it comes to data flow, due to security reasons, and non-uniform due to OS specific APIs, although Electron JS is a good framework that abstracts away OS specific APIs.

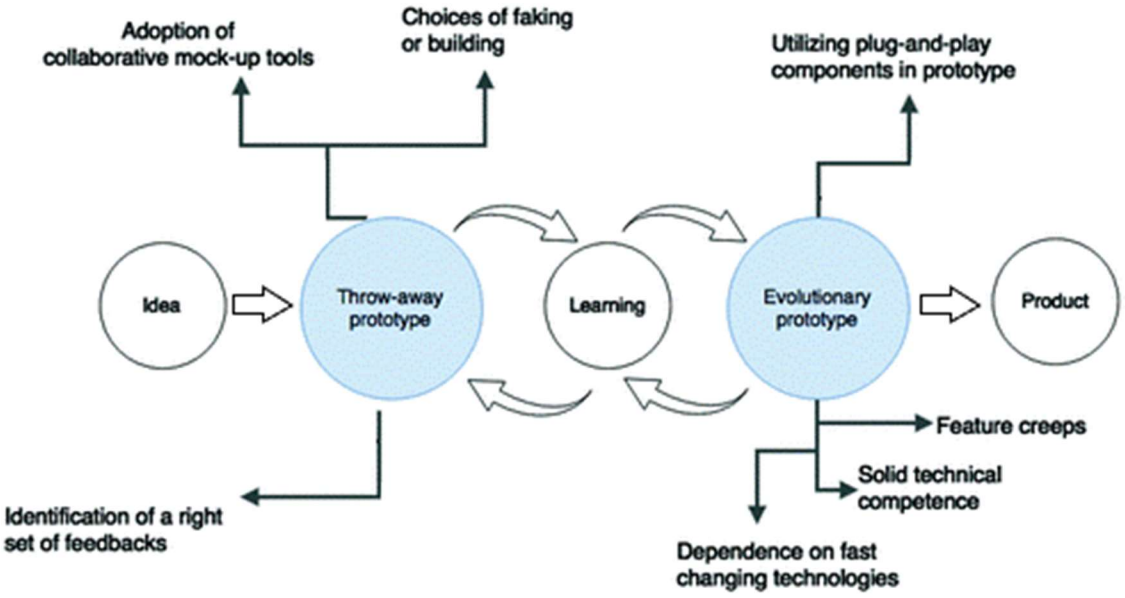


Fig 4.9 Our Software Model

4.4.3 Software Requirements Specification

Our SRS document is essentially a *complete task backlog* that lists down all the features and work that needed to be done during the development process, as we have followed agile principles. The backlog helped us keep track of what has been done and what needed to be done.

TASK BACKLOG NUMBER	TASK TITLE	TASK OWNER
1	**Project Conception and Initiation**	
1.1	Goal Setting	B Swaika
1.2	Research	B Swaika
1.3	Guidelines	B Swaika
1.4	Risk Analysis	S Jain
2	**Project Prototyping**	
2.1	Web Architechture	S Jain
2.2	Basic Controller Developement	B Swaika
2.3	Genetic Algorithms Developement	B Swaika
2.4	Integration of GAs with the Basic Controller	B Swaika
2.5	Debugging and testing	S Jain
3	**Project Developement**	
3.1	Desktop Architechture	B Swaika
3.2	UI UX Designs	B Swaika
3.3	UI state Developements	S Jain
3.4	Plugging the logic for the controller and the GA from the prototype	B Swaika
3.5	Setting Up Visualisation Parameters	B Swaika
3.6	Developing the Analytics module	B Swaika, S Jain
4	**Project Performance / Monitoring**	
4.1	Application Testing	S Jain
4.2	Simulation Runs	B Swaika
4.3	Visualizing Simulation Runs	B Swaika
4.4	Best agent plays the game	S Jain
5	**Project Reporting**	
5.1	Literature Review	S Jain, B Swaika
5.2	Assisting Materials Creation	B Swaika
5.3	Report Writing	B Swaika

Fig 4.10 Our Task Backlog

4.4.4 Core Logic Flowchart

The core *logic flowchart* demonstrates the working of the genetic algorithm in conjunction with the tetris controller. It also highlights the *data flows* that happen from function to function.

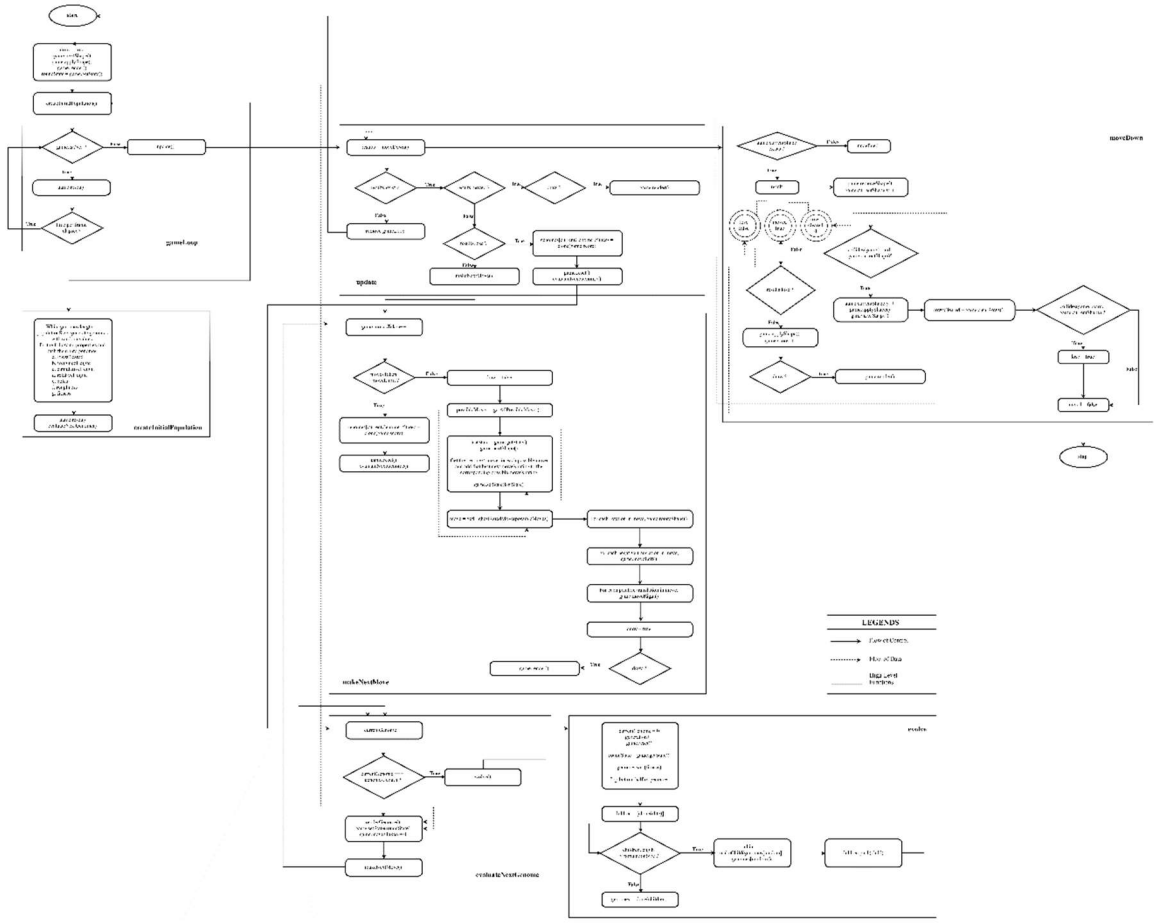


Fig 4.11 The core logic flowchart of our application, showing data flow, and functional interaction alongside the logic break down for high level functions in the algorithm. A larger version of this diagram, is available at the end of this report.

4.4.5 UI Design

The below images show our design process for the user interface of the application. The UI has been designed to keep the aesthetics very modern. We have also worked on the entire iconography and logo design to keep our design consistent with the entire system. We have only used UIKit as a styling dependency for styling our settings form. Apart from that everything has been done with pure SASS.

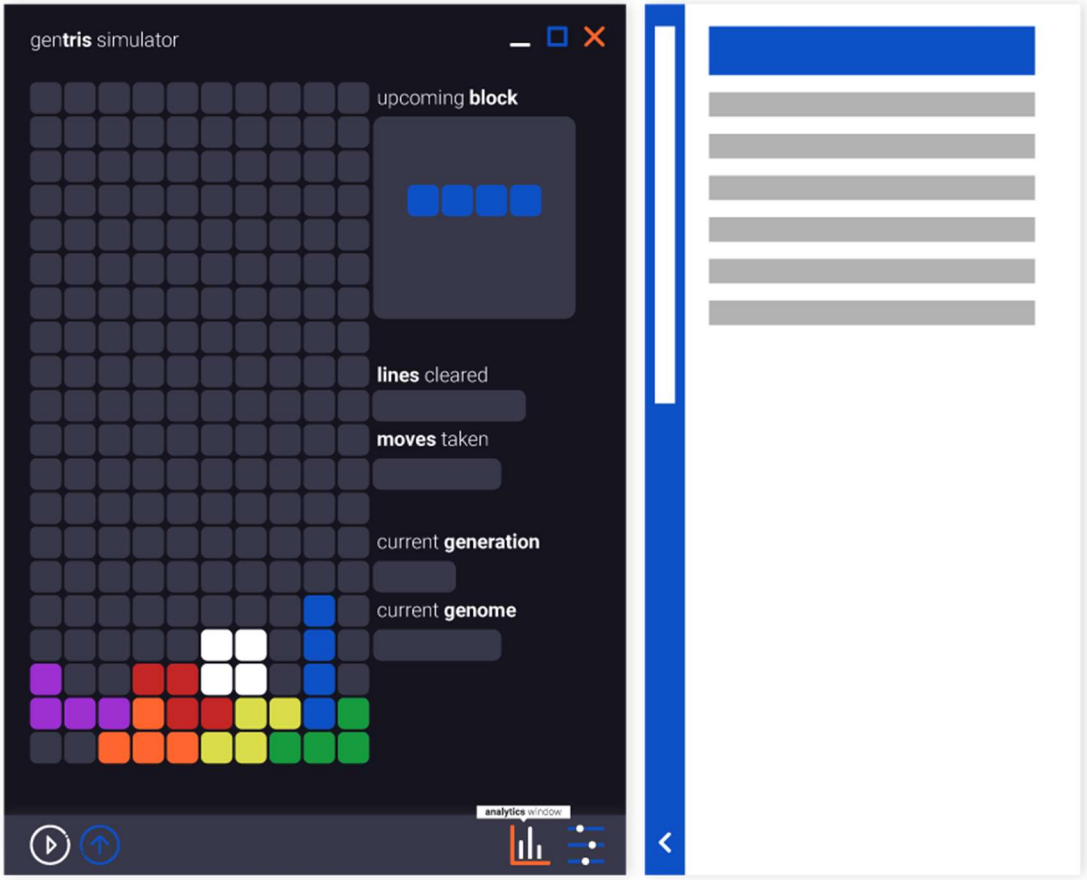


Fig 4.12 UI Wireframe

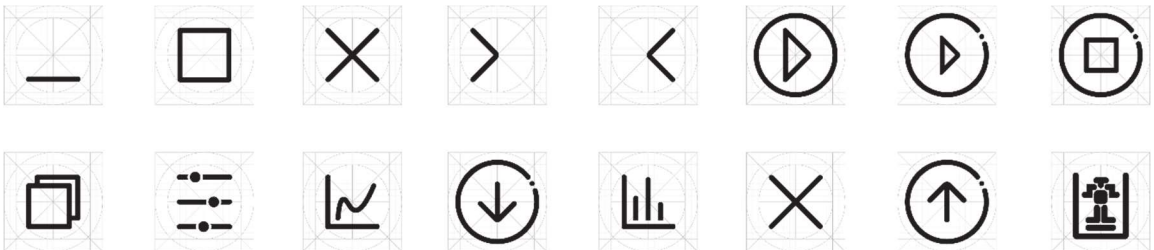


Fig 4.13 Icon Design Process

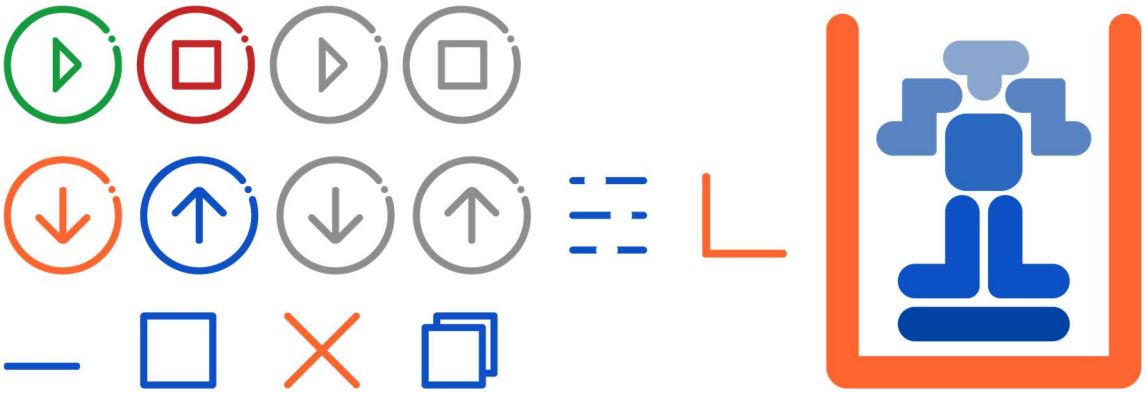


Fig 4.14 Final Icons exported for usage. The right most image is of the application logo. It is important to point out that, parts of different icons are not visible because, they have whites in them, and have thus become invisible against the white background.

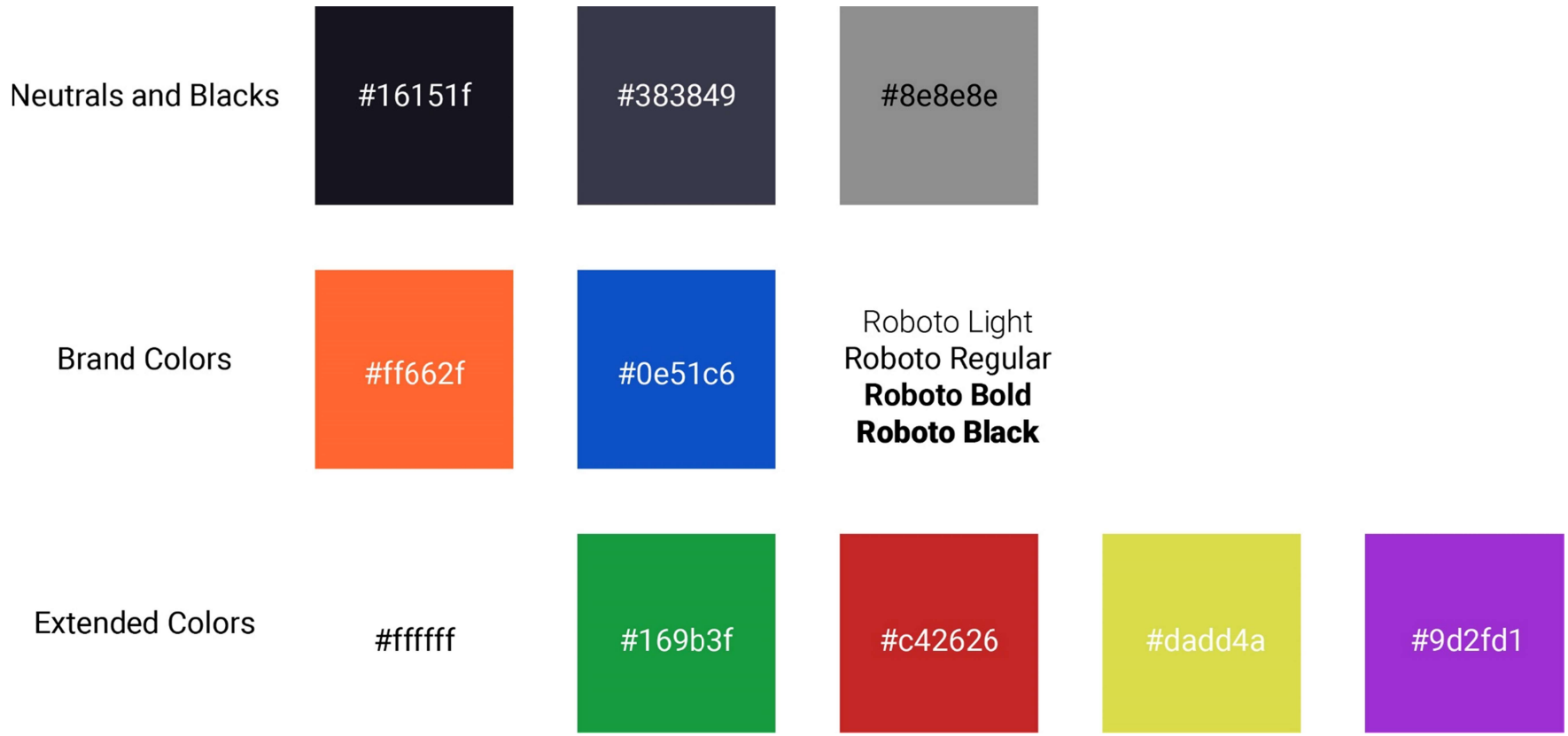


Fig 4.15 UI Guidelines for the application. This includes the colors for the application and the typography used.

4.4.6 Data Flow Guidelines

We had agreed on a uniform set of guidelines to be conformed to for our results object, so that we could easily develop on an individual but uniform basis, without having to go back and forth with how the individual developer has planned for the data to look like. Given below is a sample of our data guideline for our result object.

```
{
        “config”: {
                “config-key”: “config-value”
        },
        “elites”: [{
                “elite-gene”: “elite-gene-value”
        },
        {
                “elite-gene”: “elite-gene-value”
        }],
        “genomes”: [
                [{
                        “genome-gene”: “genome-gene-value”
                }],
                [{
                        “genome-gene”: “genome-gene-value”
                }]
        ]
}
```

Chapter 5 | Issues and Challenges

5.1 Implementation Issues

We have exhaustively tested the application to figure out and remove any bugs and issues in the application. But, there are a few issues that need to be reported and documented here. To start with the implementation, a minor issue that we have on comparison with the works of others on the same subject, is the fact that other tetris controllers generate the next piece to be falling with the help of a seeded random number generator, which does help these studies in a major way, i.e., since the pieces are seeded, the algorithm always gets a similar sequence of pieces, which helps it score more, because the weights had been adjusted for those sequences of pieces. We have identified this issue very early on during the project prototyping, but we intentionally kept this as a difference from other studies, so that our agents are always trained on totally random pieces, and they actually learn to play the game with totally random pieces, although they score comparatively less. We also have an issue with the application itself, which doesn't crash the application or harm the functioning of the application or the computer on which it is running, which is gpu acceleration. Due to Electron JS setting gpu acceleration on by default, and us not checking whether a computer has gpu, and then turning it on depending on the availability of a gpu, the application in a development environment might report some graphics buffer error to the console in case the environment running the application does not have a gpu available. We did not want to go into messing around with hardware data on a computer, as exposing the application to such sensitive data might lead to security issues, which are outside the scope of the project, and thus left the issue in its own state. Lastly, to talk about a major issue that exists is that we have not implemented any special optimization technique in particular, to enhance our genetic algorithm that helps reach optimal solutions faster, because of a lack of understanding of the mathematics that goes behind different optimization techniques, that are mentioned in the literature review section.

5.2 Challenges during the Project

Coming to some of the challenges, we had limited CPU hours and inefficient hardware to test the software, which led to us not being able to run the simulations for as long as we wanted. It is known that, the more the genetic algorithm evolves, the more it converges towards the optimal solution, and thus, we are not very sure as to whether we could have had achieved better performance from agents. It is also important to note that our implementation had a limit on the moves taken by an agent to play the game, for which our maximum score is automatically capped at a much lower bound than what we would have had wanted. This has been kept due to the fact that if more moves would have been permitted then it would have had taken more time to complete and on our testing hardware, it would have created certain undesirable problems, like that of excessive RAM usage, blocking the entire CPU on its own, etc.

Chapter 6 | Results and Discussion

6.1 Data Collection and Analysis Process

We carefully implemented our tetris controller in such a way that most of the data is collected during the runtime of the algorithm, in a predefined fashion, which is easier to implement any sort of parse-automation on. This allowed for a uniform data standard with which data processing became easier moving forward with the project.

We allowed all of our algorithms to run for a total time of around 115 hours in a period of 1 week. This boiled down to around 16.5 hours of runtime on a daily basis. During these simulation runs, we constantly kept varying different parameters of the genetic algorithm so as to collect enough data to compare different evolution approaches taken by different algorithms. The variations were thought out ahead of time, because we wanted to perform a correlation study, between the different parameters of an algorithm and the results that it produced. Table 6.1 summarizes the different configurations used to run simulations.

	Method 1	**Method 2**	**Method 3**	**Method 4**	**Method 5**
Number of Generations	50	50	50	50	50
Population Size	50	50	50	50	50
Mutation Rate	0.05	0.05	0.05	0.1	0.2
Mutation Step	0.2	0.2	0.2	0.3	0.3
Selection Rate	50%	50%	50%	25%	10%
Crossover Type	random	random	average	random	random
Controller Type	two-piece	two-piece	one-piece	one-piece	two-piece
Weights Used	cumulativeHeight holes relativeHeight roughness rowsCleared weightedHeight	cumulativeHeight holes relativeHeight roughness rowsCleared weightedHeight wells blockades	cumulativeHeight holes relativeHeight roughness rowsCleared weightedHeight	cumulativeHeight holes relativeHeight roughness rowsCleared weightedHeight	cumulativeHeight holes relativeHeight roughness rowsCleared weightedHeight blocks weightedBlocks

Table 6.1 Continued on the next page.

	Method 6	Method 7	Method 8	Method 9	Method 10
Number of Generations	200	70	70	100	50
Population Size	50	60	60	70	100
Mutation Rate	0.2	0.1	0.1	0.01	0.15
Mutation Step	0.3	0.25	0.25	0.2	0.32
Selection Rate	10%	25%	50%	25%	10%
Crossover Type	random	random	random	average	average
Controller Type	two-piece	two-piece	one-piece	two-piece	one-piece
Weights Used	cumulativeHeight holes relativeHeight roughness rowsCleared weightedHeight blocks weightedBlocks	cumulativeHeight holes relativeHeight roughness rowsCleared weightedHeight	cumulativeHeight holes relativeHeight roughness rowsCleared weightedHeight	cumulativeHeight holes relativeHeight roughness rowsCleared weightedHeight	cumulativeHeight holes relativeHeight roughness rowsCleared weightedHeight blocks weightedBlocks

Table 6.1 A summary of configurations used for our testing and analysis purposes to run simulations.

We have also performed on the fly correlation studies between the weights of an algorithm and the fitness of the algorithm, with our data visualization module. This was specifically done with one purpose in mind, to visualize which parameters were positively related to an algorithm's performance and which were negatively related. Although, the results for this correlation were ecstatic compared to our expectations, we can put enough reason behind it by stating two factors that led to the ecstatic nature of this correlation study:

- The initialization is random for a particular simulation, which often makes that particular simulation oriented to perform in a certain manner, depending on the initial value of the weights, and since a variety of these weight values actually effect the rating function for deciding moves, in some cases, certain weights outweigh other weight values to affect their expected correlation to the algorithm's performance
- We had very limited time and resources to run these simulations, which directly ensures that the entire search space was not navigated through by our algorithms, and these correlation values would become a lot closer to their expected values if we could have run our simulations for longer periods of time

It is also crucial to note a very important limiting factor of our project, the number of moves that were allowed for the algorithms. We put this limitation consciously, because of the lack of good computing resources, with which we could endlessly evolve our genetic algorithms, to obtain much higher scores.

A lot of studies done under the umbrella of the game tetris, implementing some form of AI agents has been performed under these two evaluation schemes:

- **Conventional Evaluation** → This model uses the traditional tetris scoring system whereby points are awarded for blocks moving down, and extra points are awarded for clearing consecutive lines. In this model move limits are used, but the scores obtained are relatively higher for the actual number of lines cleared. This model was used earlier to compare performances of human players who competed on achieving high scores in the game, throughout the world.
- **Non-Conventional Evaluation / Modern Standard** → This model uses the total number of lines cleared as the score for the game. This model was put into use to have uniform comparison grounds between different AI agents, but these models have generally been tested on mainframes, and cloud machines which are more capable of running these resource-intensive algorithms, which is the main reason why they don't use move limits. This model has been extensively used to provide evaluations for different AI agents, in modern literature. This also leads to very high scores for different agents as they get more time to evolve and explore the search space.

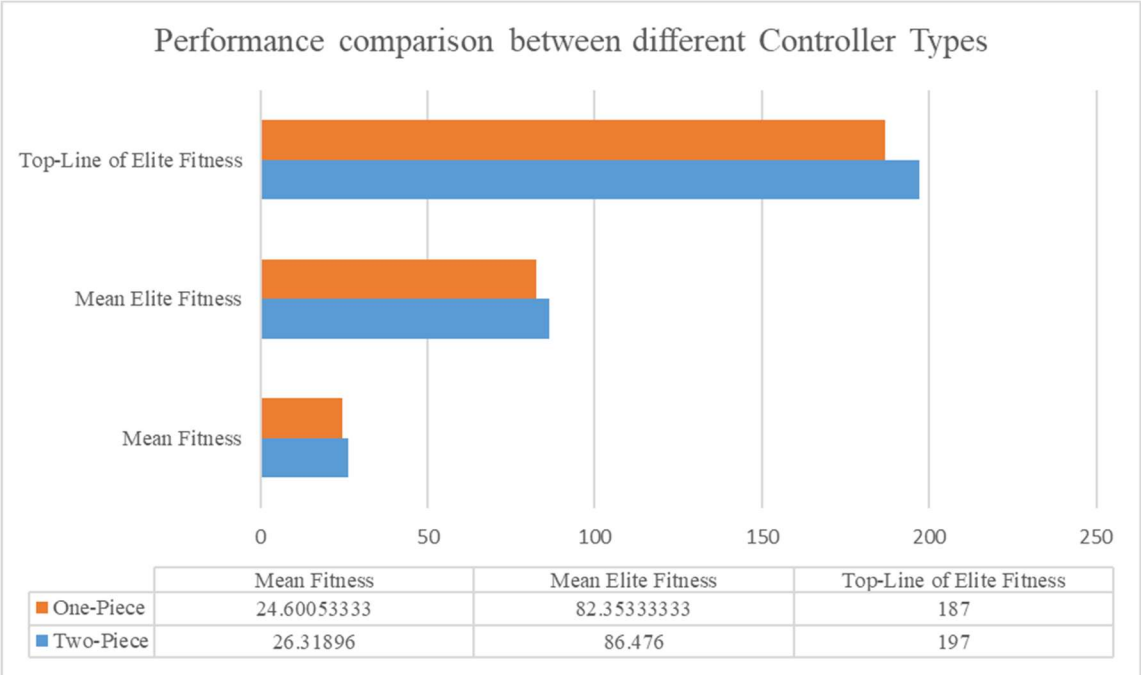
Although, we have tried to stick to standards, both of these models were not proving suitable for our project, because we wanted to compare our agents uniformly against other agents, but at the same time had to limit our moves as we did not want our algorithms to be very resource-intensive and cause unexpected crashes. We have used a midline between the two models by taking the number of lines cleared as our scoring metric and limiting the number of moves at the same time. We had to therefore, go ahead and do some further processing to provide comparisons between other agents that use the non-conventional evaluation scheme. We have extrapolated our scores using linear regression techniques, to provide a good comparison between other AI agents. We have also performed some basic comparisons with the conventional evaluation scheme.

We have divided our results into two main areas, which are discussed hereafter, whereby the first area focuses on simulation-based results comparing the results obtained from each individual simulation, and the second area focuses on a more overall look at the results performing analytics on the entire data set, including all simulation runs.

6.2 Experimental Results

6.2.1 Simulation-Based Results

We would like to begin with one of the most obvious parameters for result comparisons, that is, the type of controller the agent uses to play the game. In philosophical terms, the controller type is analogous to the amount of future knowledge provided to the algorithm. We had kept two options for this parameter, a) one-piece, and b) two-piece. The one-piece controller only had knowledge about the current piece on the board, whereas the two-piece controller had knowledge about both the current tetromino and the next tetromino to fall. The two-piece controller, thus had more information about the game, and leveraged this information to carefully choose such a move that would give it an edge for placing the next piece. We have seen two piece controllers outperform one-piece controllers by quite some margin. Fig 6.1 shows such a performance comparison between agents using different controller types.



	Mean Fitness	Mean Elite Fitness	Top-Line of Elite Fitness
One-Piece	24.60053333	82.35333333	187
Two-Piece	26.31896	86.476	197

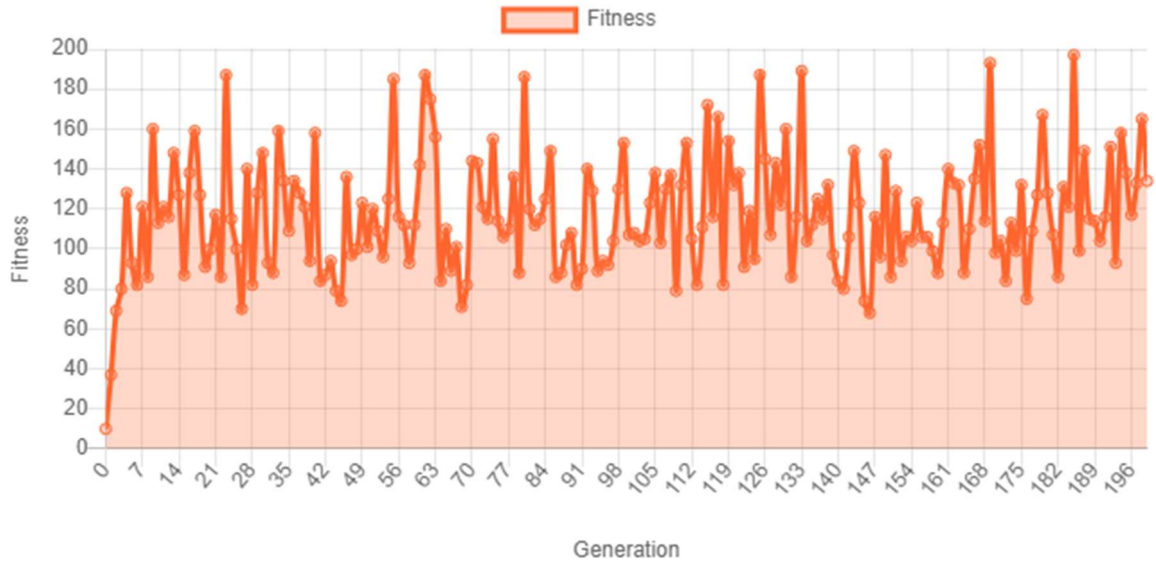
Fig 6.1 Performance comparison between agents using different Controller Types.

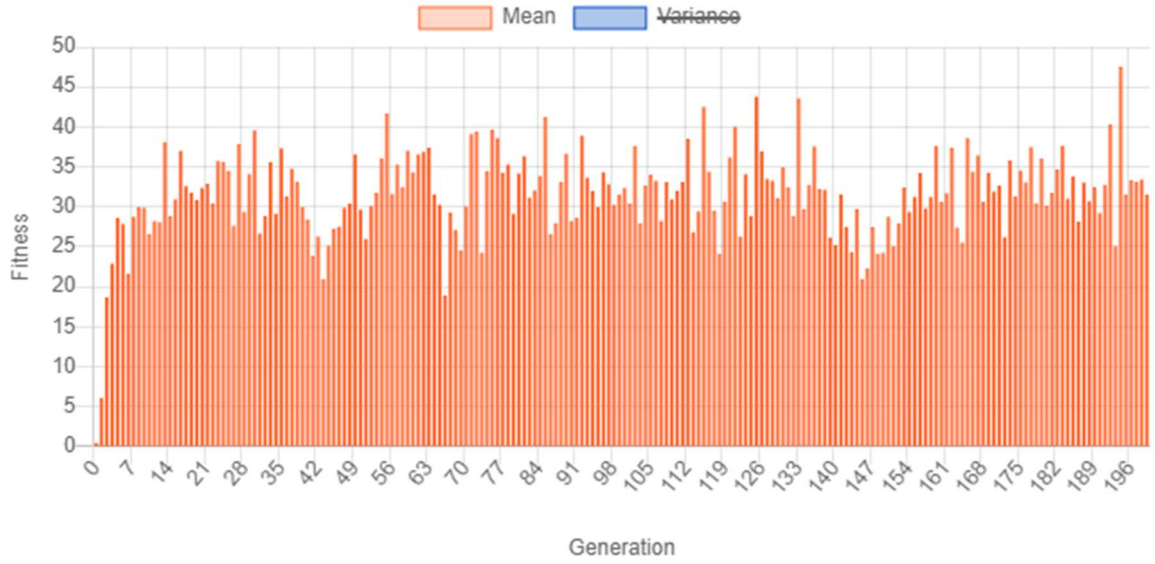
We can clearly see, that one-piece controllers perform worse than two-piece controllers, due to the fact that they have lesser information about the game and thus can only utilize that much information to play the game, which puts them at a less advantageous position.

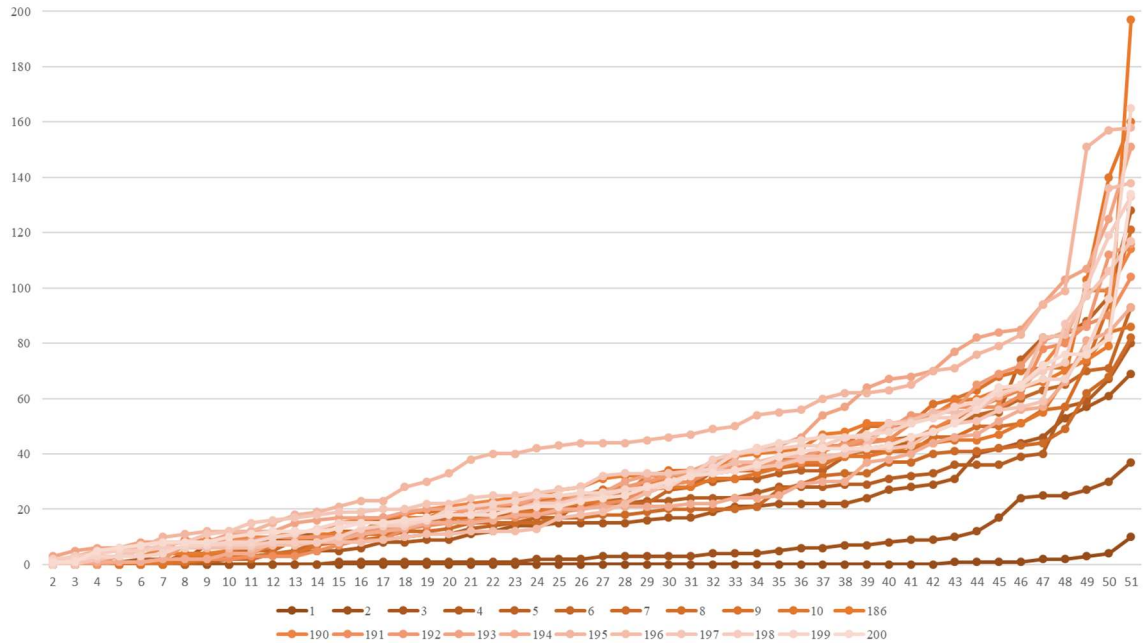
We would now like to talk a little bit in-depth about our best-performing agent which is **Method 6**. This method uses the configuration as mentioned in Table 6.1. It has achieved the highest score of *197 lines* among all of our algorithms. This method uses additional weights apart from our fixed set of weights, which are blocks and weightedBlocks and is essentially an extended version of **Method 5** which generated the high score of *194 lines*. Blocks and Weighted Blocks are features of the board which govern how many blocks are present on the board in the current game state, and how much of that is concentrated towards the bottom respectively as weighted blocks essentially attaches a weight for each row a block is present in. The lower the presence of a block on the board, the more is the weight of the board. These weights have proven to be more effective than the other options for additional weights like wells, and blockades, which have not performed as well as the weightedBlocks and blocks weights. A thorough visualization of **Method 6** is provided in Fig 6.2. In this algorithm, we observe positive correlations with fitness amongst the weights weightedBlocks and roughness, whereas negative correlations for all other weights. It is crucial to point out that rowsCleared should be a positively correlated weight, which draws its logic from common sense, but in this scenario shows a negative correlation. The best fit elite obtained from this algorithm has the following weight distribution:

- weightedHeight : **-0.164**
- cumulativeHeight : **-1.408**
- relativeHeight : **-0.043**
- holes : **-0.113**
- roughness : **-0.091**
- rowsCleared : **-1.384**
- blocks : **-1.256**
- weightedBlocks : **0.07**

It is important to convey that both **Method 5** and **Method 6** had run out of moves to achieve their high scores. We strongly believe that in a limitless game environment, these two methods can compete almost neck-to-neck in clearing lines against well-known evolved agents of different tetris researchers, who have achieved scores ranging in millions of lines cleared. This aspect of comparison has been dealt with in more detail in the upcoming section where we compare our agents against some of the other well-known agents in literature.


Fitness
Fitness
Generation


Mean
Variance
Fitness
Generation


1 2 3 4 5 6 7 8 9 10 186
190 191 192 193 194 195 196 197 198 199 200

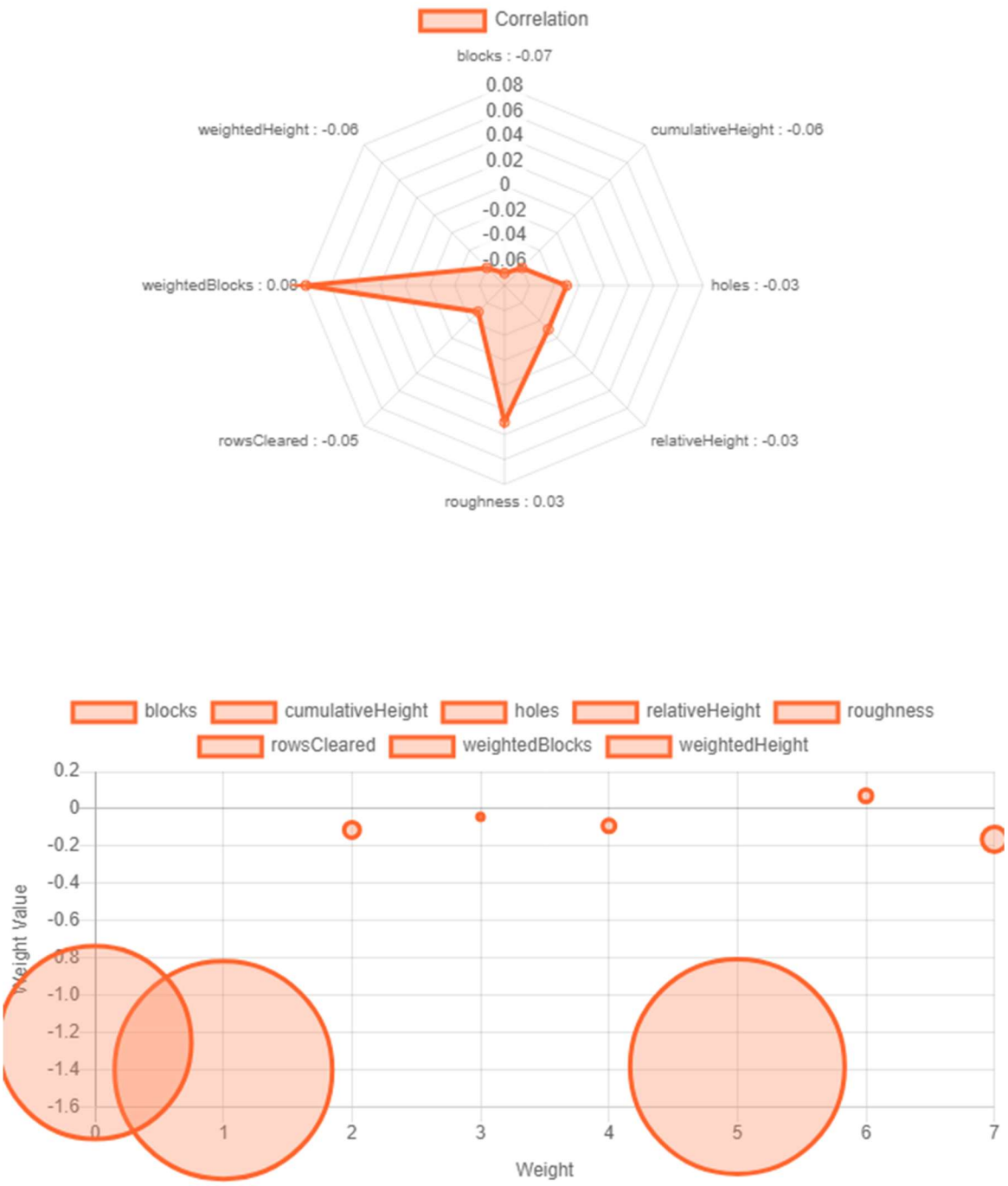


Fig 6.2 A complete visualization of Method 6. The components on the previous page illustrate the evolution part of the algorithm and helps figure out performance trends. The components on this page, highlight the correlation of weights for the algorithm *(above)* and also summarizes the weight values of the best fit elite *(below)* for this algorithm.

Fig 6.3, 6.4, 6.5, and 6.6 show visualizations for **Method 2, Method 4, Method 9,** and **Method 10** respectively taken directly from our analytics module. Going through these visualizations give us some sort of clarity on how genetic algorithms work by seeing their evolution graphs. These charts also help us understand the nature and magnitude of correlations that different algorithms present for their weights and their overall fitness and how each generation in these algorithms has performed by giving us a glance of generational fitness means. The above methods are illustrated below because they vary greatly on their configurations, and display a variety of the parameters that we had set up.

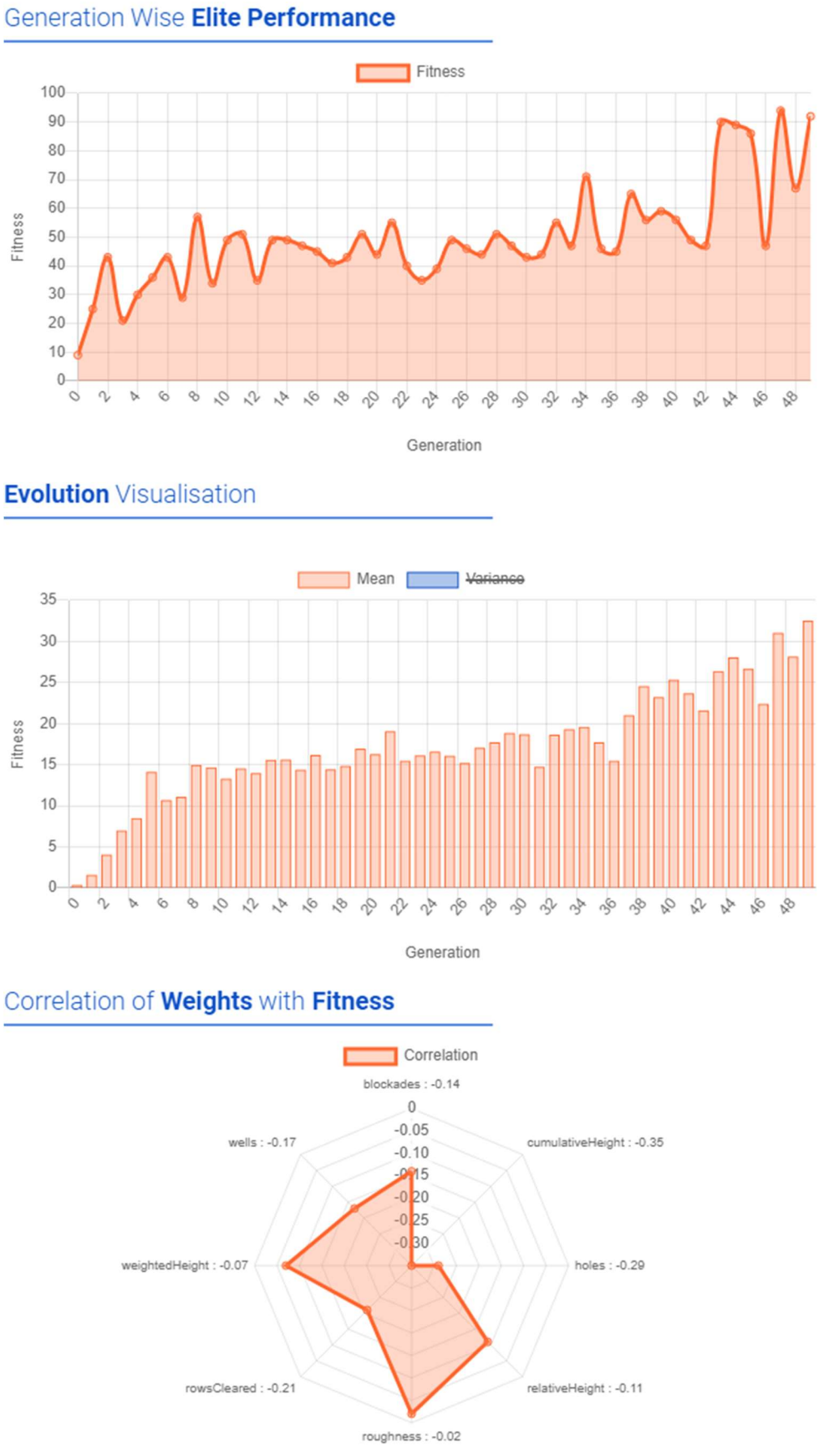


Fig 6.3 A thorough visualization of Method 2

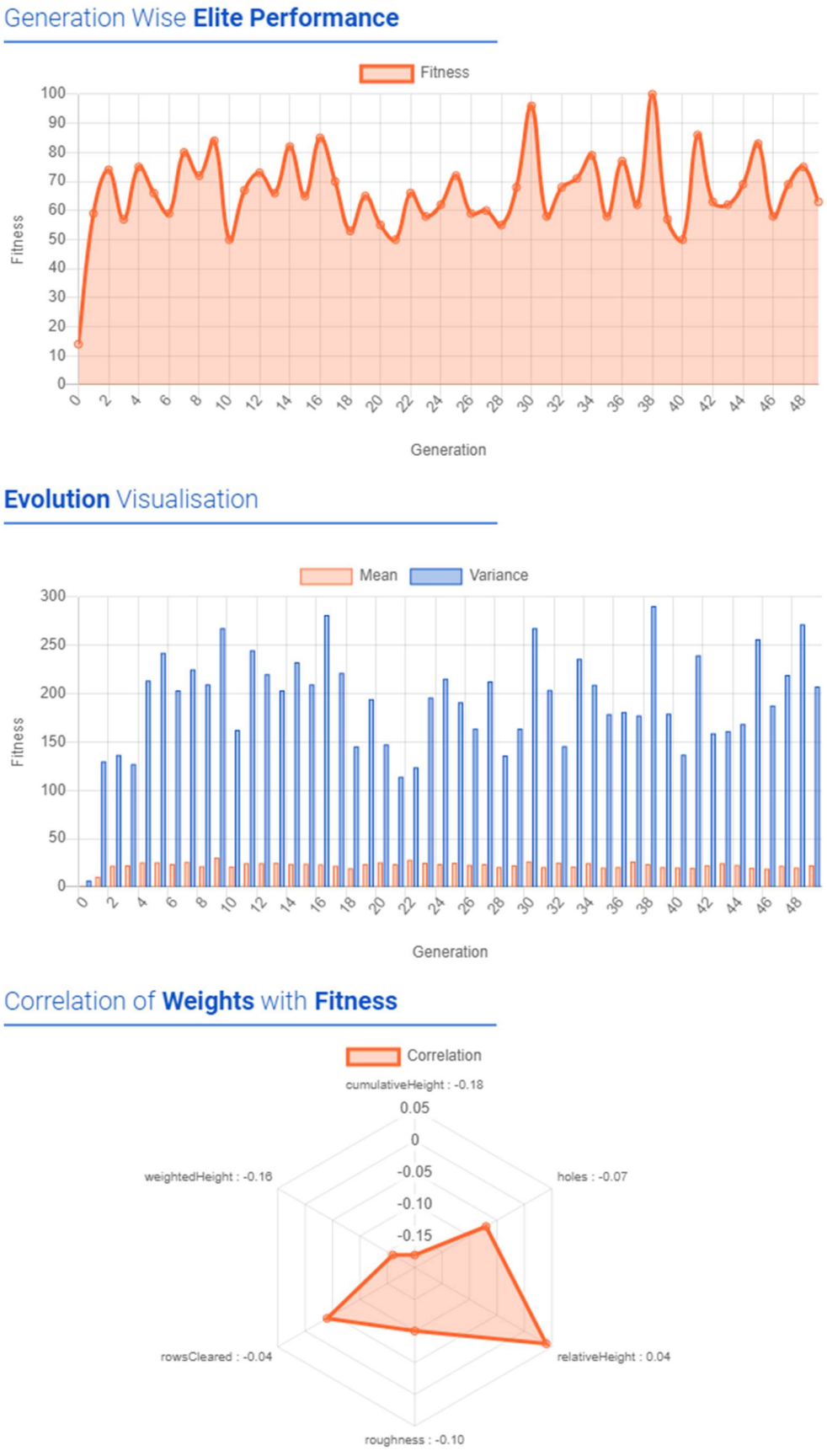


Fig 6.4 A thorough visualization of Method 4

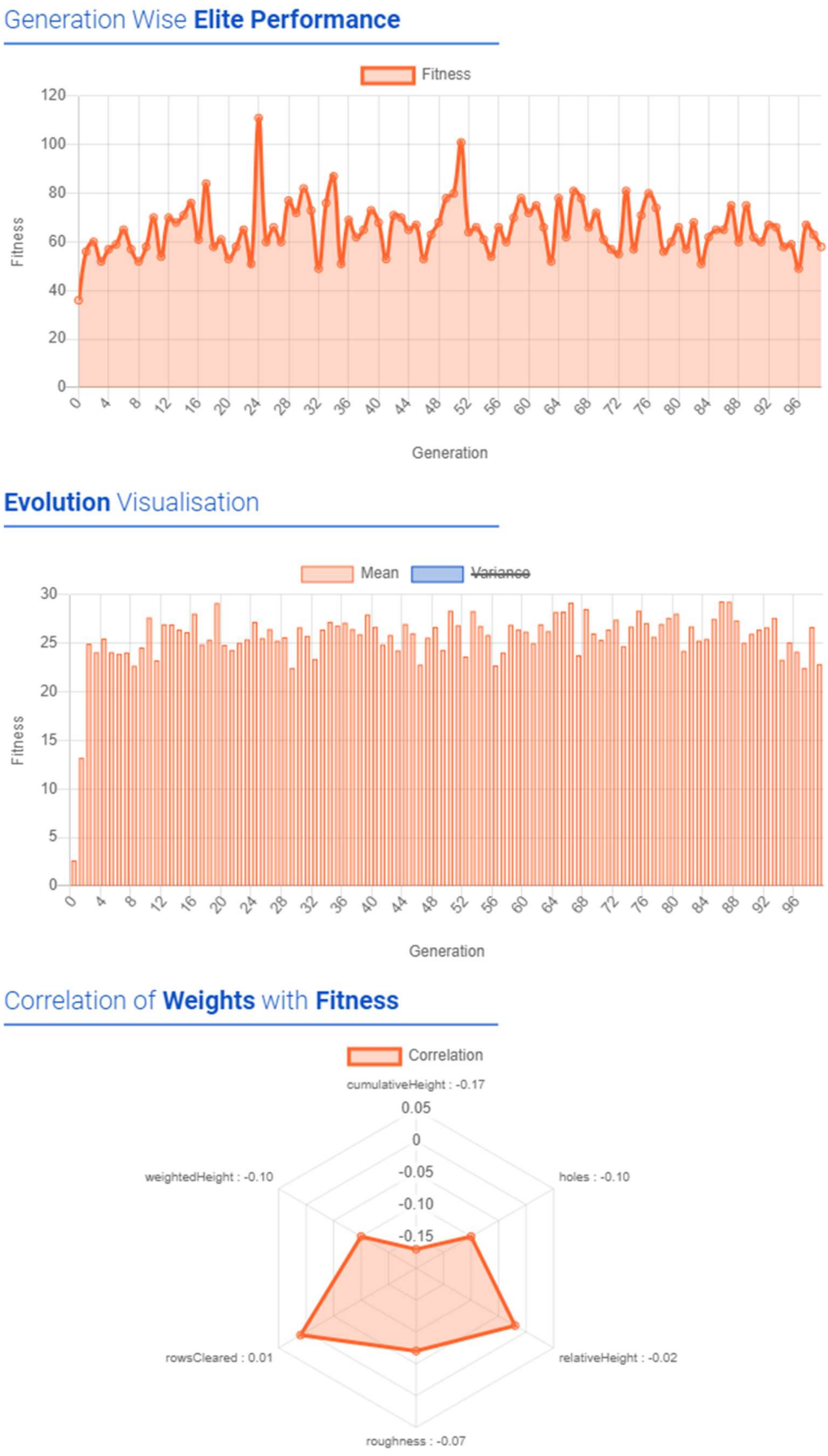


Fig 6.5 A thorough visualization of Method 9

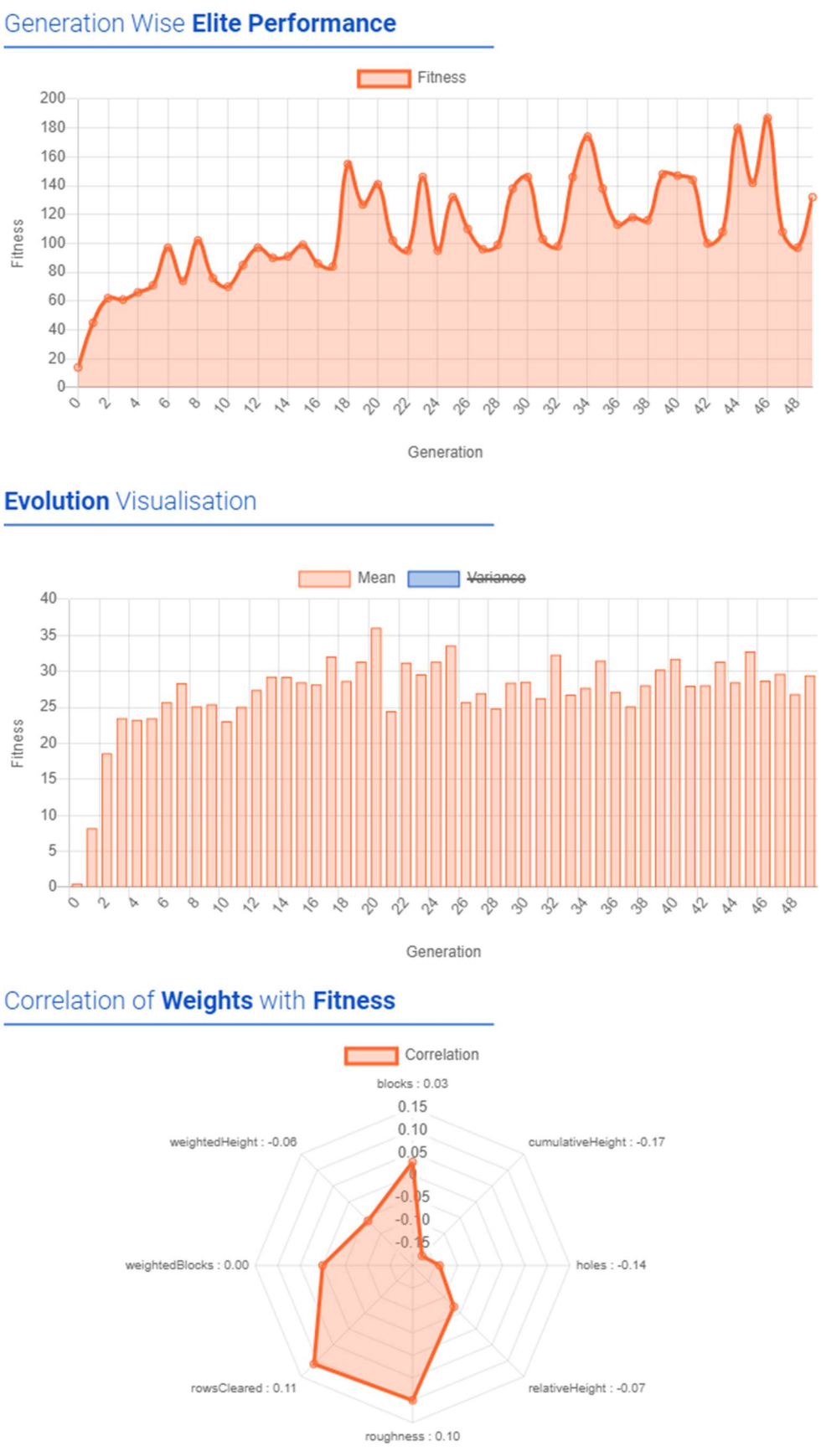


Fig 6.6 A thorough visualization of Method 10

6.2.2 Overall Results

To start this section, we present a correlation study that was performed among **Method 1**, and **Methods 4-10**. The objective behind this was to identify which generic parameters of genetic algorithms have more impact on the algorithm itself, and in what directions do these relationships exist. We observed that the number of generations had a much lesser impact on the mean elite fitness, which directly implies that smaller number of generations can also evolve highly fit elites, whereas the number of generations had a much bigger impact on the mean fitness of the algorithm, which directly implies that larger number of generations can evolve more fit algorithms overall. We have also seen that selection rate plays a negative association with the fitness of genetic algorithms, which essentially proves the fact that if there is more competition, i.e., a lesser selection rate, then the fitness level of that algorithm is bound to be more than an algorithm with lesser competition. This also draws from a philosophical standpoint whereby; a more competitive set of individuals are likely to perform better in a scenario than a less competitive set of individuals.

METHOD	1	4	5	6	7	8	9	10	CORRELATION (MF)	CORRELATION (MEF)
Generations	50	50	50	200	70	70	100	50	**0.779718539**	**0.426994404**
Population Size	50	50	50	50	60	60	70	100	**0.136753894**	**0.22527373**
Mutation Rate	0.05	0.1	0.2	0.2	0.1	0.1	0.01	0.15	**0.687348127**	**0.888782059**
Selection Rate	0.5	0.25	0.1	0.1	0.25	0.5	0.25	0.1	**-0.728506561**	**-0.754764106**
Mean Fitness (MF)	**22.6164**	**22.634**	**26.9636**	**31.4848**	**24.92**	**24.022**	**25.61**	**27.1456**		
Mean Elite Fitness (MEF)	**68.66**	**66.5**	**105.92**	**116.65**	**75.67**	**71.54**	**65.48**	**109.02**		

Table 6.2 Results of the correlation study.

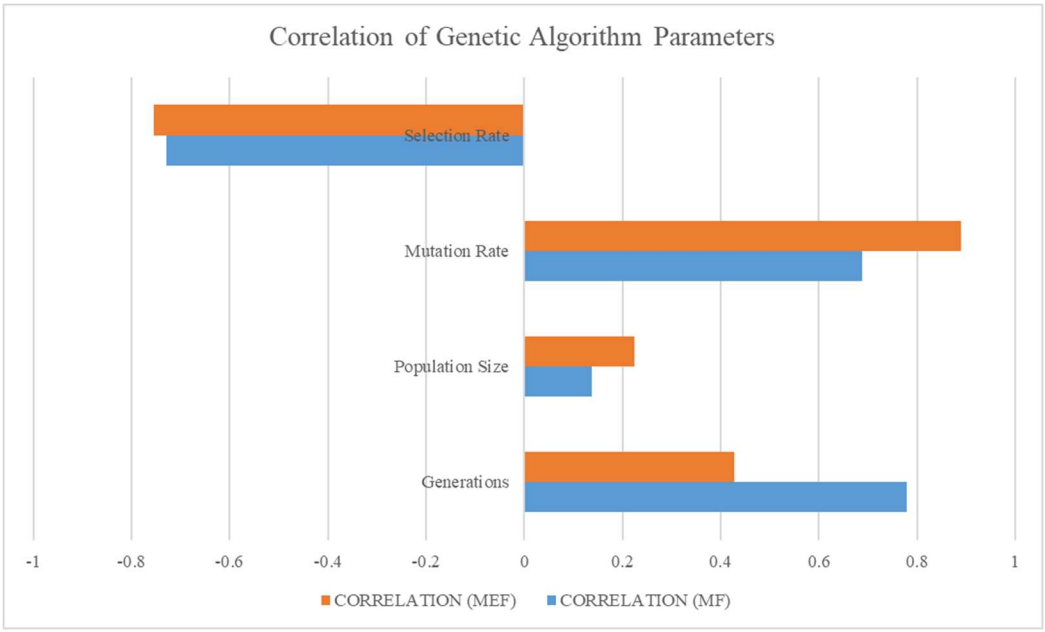


Fig 6.7 An illustration of how different genetic algorithm parameters are correlated to their fitnesses.

As mentioned earlier, we have move limitations set up on our implementation of the tetris controller. We would like to briefly elucidate how we have extrapolated scores obtained by us, into more comparable scores with literature-drafted scores. We implemented simple linear regression to extrapolate our results, which is a widely used estimation technique used for purposes of forecasting univariate or multivariate time-series equivalent data. This technique employs the concept of finding a good fit line amongst the provided data points, by calculating slopes and intercepts from the given dataset. It then uses this best fit line, to predict future values that could possibly arise in the dataset. In our case, the regression model is very weak, essentially due to the fact, that we had a dataset that consisted of only 500 points, but we had to extrapolate values in the range of millions. It should be noted that this kind of a regression is generally optimistic in nature, and often discards outliers that affect a dataset negatively. We obtained a high score of *197 lines* in *500 moves*. On extrapolating the data, we have reason to believe that our algorithm, **Method 6**, which is our best agent yet, can clear roughly around ~ *434,000 lines* in *1,000,000 moves*. It was also necessary for us to take a note of the actual time that it takes to run the small limited algorithm and extrapolate it to figure out exactly how long would it have had taken to achieve the extrapolated score using the same configuration. Although, we did not have any statistical measure for the above, mainly due to the lack of data that could actually provide insight on such an extrapolation, we would like to present our best estimates for the same. The small limited algorithm takes around *18 hours* to complete its evolution process. Given a million moves per individual, this algorithm could roughly take around *12,000 hours (500 days)* to run on our hardware. It is for this reason that we chose to limit the moves. Fig 6.8 illustrates the extrapolation of scores for **Method 6**.

We have also compared our scores against different tetris high scores. The highest score recorded to date in tetris by any human player, is credited to Mathew Buco who had a conventional *score of 999,999* in Nintendo Tetris [29], by clearing *207 lines*. Our algorithm lags behind by 10 lines, and thus touches a *score > 980,000*. This is by far, the best achievement that our genetic agent has successfully achieved without any extrapolation. Table 6.3 summarizes a comparison of our genetic agent against different AI agents that have already been credited with achieving high scores in literature.

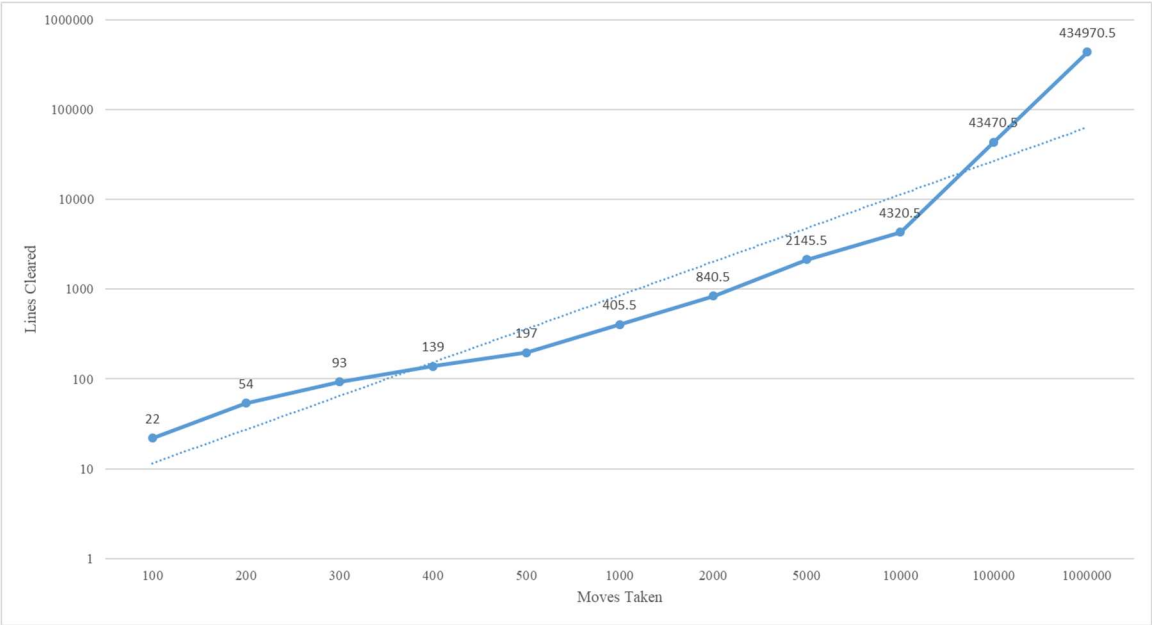


Fig 6.8 Extrapolating scores obtained from Method 6.

AUTHOR, YEAR	SCORE
Lippmann et. al., 1993	N/A
Tsitsiklis and van Roy, 1996	31
Bertsekas and Tsitsiklis, 1996	3,200
Kakade, 2001	6,800
Lagoudakis et. al., 2002	2,000
Dellacherie, 2003	6,50,000
Fahey, 2003	72,00,000
Ramon and Driessens, 2004	50
Llima, 2005	50,000
Bohm et. al., 2005	N/A
Farias and van Roy, 2006	4,700
Szita and L orincz, 2006	3,50,000
Thiery and Scherrer, 2009	3,50,00,000
Langenhoven, 2010	2,75,000
Scherrer, 2013	4,000
Our Project, 2019	**~ 434,000 (in a million moves)**

Table 6.3 A summary of score comparisons between different authors in literature and our agent.

6.3 Discussion

A lot of the results obtained, match our expectations and prove this experiment successful. We can clearly see that our implementation is comparable to other researches, and in fact, given a longer period of runtime, and better computing resources, could easily outperform a lot of the mentioned agents. We observe that other optimization techniques like Noisy Cross-Entropy and Particle Swarm Optimizations are actually very viable to create intelligent tetris agents, whereas some like Lambda Policy Iteration and Linear Programming are not. We can also clearly differentiate that genetic algorithms converge slower than other optimization techniques, due to an extensive search procedure. The only outlier is the large variation in weight to fitness correlation, the reasons of which have been stated above already. This variation does not affect the experiment adversely, due to the fact that, every algorithm is different and correlations obtained hugely depend on the search path of the algorithm.

Our agents perform much better than at least a 99%ile of human tetris players, and performs above average and gives tight competition to other AI agents. It is important to mention that the time taken by our algorithms to converge could effectively be reduced manifolds by implementing additive optimization techniques to tune the weights further and narrow down the search space for our genetic algorithm.

This project exposes us to the inner workings of genetic algorithms and how they can be implemented to learn tasks that are either computationally expensive to program, or are so abstract that they do not present a clear way to implement programmatically. These genetic algorithms can be extended for implementation in a variety of games, and other areas which require the algorithm to take some time and learn how to perform a task before perform actually performing it.

Chapter 7 | Future Scope and Conclusion

7.1 Future Scope

The project, although complete, is definitely not over. We have intentionally left out a variety of things from this project that can be developed further. Some of the mention-worthy aspects of future scope for the project are, inclusion of the exhaustive list of weights provided in Table 2.2, addition of optimization techniques to converge to solutions faster, fixing issues that arise during linux deployments, and conducting ANOVA tests on the various results, to obtain a statistical and philosophical understanding of how genetic algorithms can be constructed more efficiently.

7.2 Conclusion

This project has been a great exploratory experiment to understand the workings of genetic algorithms, and take up the process of software development and engineering, and creating production-ready software. We have undergone a humongous learning experience in the 9 weeks of implementing this project, and every phase of implementation has introduced us to newer concepts, some which apply specifically to the problem at hand, and some which apply to the overall field of computer science engineering. It has also been a good exposure into scientific writing, which definitely becomes an added skill.

We have successfully realized our objective, of creating a platform for testing, developing, and analyzing different evolutionary algorithms for the game of tetris.

References

[1] Wikipedia. *Tetris* [Online]. Available: https://en.wikipedia.org/wiki/Tetris

[2] C. P. Fahey, "Tetris," Colin Fahey's personal website [Online], July 2012. Available: https://www.colinfahey.com/tetris/tetris.html

[3] H. J. Hoogeboom, and W. A. Kosters. (2005, January). "The Theory of Tetris." ResearchGate [Online]. Available: https://www.researchgate.net/publication/228610685_The_theory_of_tetris

[4] H. Burgiel, "How to lose at tetris," *Mathematical Gazette*, vol. 81, pp. 194-200, 1997.

[5] R. P. Lippmann, L. Kukolich, and E. Singer, "Lnknet: Neural network, machine-learning, and statistical software for pattern classification," *Lincoln Laboratory Journal*, vol. 6, pp. 249–249, 1993.

[6] C. Thiery, B. Scherrer, et al., "Building controllers for tetris," *International Computer Games Association Journal*, vol. 32, pp. 3–11, 2009.

[7] J. N. Tsitsiklis and B. V. Roy, "Feature-based methods for large scale dynamic programming," *Machine Learning*, vol. 22, no. 1-3, pp. 59–94, 1996.

[8] D. P. Bertsekas and J. N. Tsitsiklis, "Neuro-dynamic programming," *Athena Scientific*, Belmont, MA, 1996.

[9] B. Scherrer, "Performance bounds for λ policy iteration and application to the game of tetris," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1181–1227, 2013.

[10] S. Kakade, "A natural policy gradient," *Advances in Neural Information Processing Systems*, vol. 14, 2001.

[11] M. G. Lagoudakis, R. Parr, and M. L. Littman, "Least-squares methods in reinforcement learning for control," in *Methods and Applications of Artificial Intelligence*, pp. 249–260, Springer, 2002.

[12] J. Ramon and K. Driessens, "On the numeric stability of gaussian processes regression for relational reinforcement learning," in *ICML-2004 Workshop on Relational Reinforcement Learning*, pp. 10–14, Citeseer, 2004.

[13] V. F. Farias and B. V. Roy, "Tetris: A study of randomized constraint sampling," in *Probabilistic and Randomized Methods for Design Under Uncertainty*, pp. 189–201, Springer, 2006.

[14] R. E. Llima, "Xtris readme." [Online]. Available: ftp://ftp.x.org/contrib/games/xtris.README

[15] N. B¨ohm, G. K´okai, and S. Mandl, "An evolutionary approach to tetris," in *6th Metaheuristics International Conference* (August 22-26, 2005), 2005.

[16] I. Szita and A. L¨orincz, "Learning tetris using the noisy cross-entropy method," *Neural computation*, vol. 18, no. 12, pp. 2936–2941, 2006.

[17] C. Thiery and B. Scherrer, "Improvements on learning tetris with cross-entropy," *International Computer Games Association Journal*, vol. 32, 2009.

[18] L. Langenhoven, W. S. van Heerden, and A. P. Engelbrecht, "Swarm tetris: Applying particle swarm optimization to tetris," in *Evolutionary Computation* (CEC), 2010 IEEE Congress on, pp. 1–8, IEEE, 2010.

[19] N. Funk, M. Goel, A. Pramanik, S. Schaefer, and S. Stevanof, "Tetris Agent Implementation", Dept. Comp. Sc., Univ. of Singapore, Singapore, 2018.

[20] M. Stevens, and S. Pradhan, "Playing Tetris with Deep Reinforcement Learning", Dept. Comp. Sc., Standford Univ., California, Rep. 121, 2016.

[21] H. Chiroma, S. Abdulkareem, A. Abubakar, A. Zeki, A. Y. Gital, and M. J. Usman, "Correlation Study of Genetic Algorithm Operators: Crossover and Mutation Probabilites", in *International Symposium on Mathematical Sciences and Computing Research*, Perak, Malaysia, 2013, pp. 39-43.

[22] A. Czarn, C. MacNish, K. Vijayan, B. Turlach, and R. Gupta, "Statistical Exploratory Analysis of Genetic Algortihms," *IEEE Transactions on Evolutionary Computation*, vol. 8, issue 4, pp. 405-421, Aug 2004.

[23] D. Rollinson, and G. Wagner. (2010, Feb). "Tetris AI Generation Using Nelder-Mead and Genetic Algorithms." Semantic Scholar [Online]. Available: https://pdfs.semanticscholar.org/7d82/a0915b46c4b982e73792b05cb40f48e70188.pdf

[24] C. Darwin, On the Origin of Species. 1859.

[25] K. Sastry, D. Goldberg, and G. Kendall, "Genetic Algorithms," in *Search Methodologies : Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke, and G. Kendall, Eds. Boston : Springer, 2005, pp. 97-125.

[26] A. Globus, G. Hornby, D. Linden, and J. Lohn, "Automated antenna design with evolutionary algorithms," 2006.

[27] A. Keane, "The design of a satellite beam with enhanced vibration performance using genetic algorithm techniques," *The Journal of the Acoustical Society of America*, vol. 99, no. 4, pp. 2599–2603, 1996.

[28] A. M. L. da Cruz, "Visualization for Genetic Algorithms," Master's dissertation, Dept. of Informatics Eng., University of Coimbra, Portugal, 2014.

[29] Tetris World Records. (2012, Jan) [Online]. Available: https://recordsetter.com/world-record/-fewest-lines-completed-tetris-achieve-highest-possible-score-nes/13630?autoplay=false