Playing Tetris with Genetic Algorithms

Soft Computing Project Report

Submitted By:

Ankit Chawla (7)
R M Hritik Jena (69)

Original Paper:
`https://cs229.stanford.edu/proj2015/238_report.pdf`

November 12, 2025

Abstract

The paper explores how the classic game Tetris can be framed as an optimization problem, solved using a Genetic Algorithm (GA). The approach evaluates every possible game state using a weighted combination of features, and the GA optimizes these weights to produce an efficient automated player. After 30 generations of training, the best evolved player achieved an average game length of 179,531 moves over 50 trials, demonstrating the effectiveness of the method.

1 Introduction

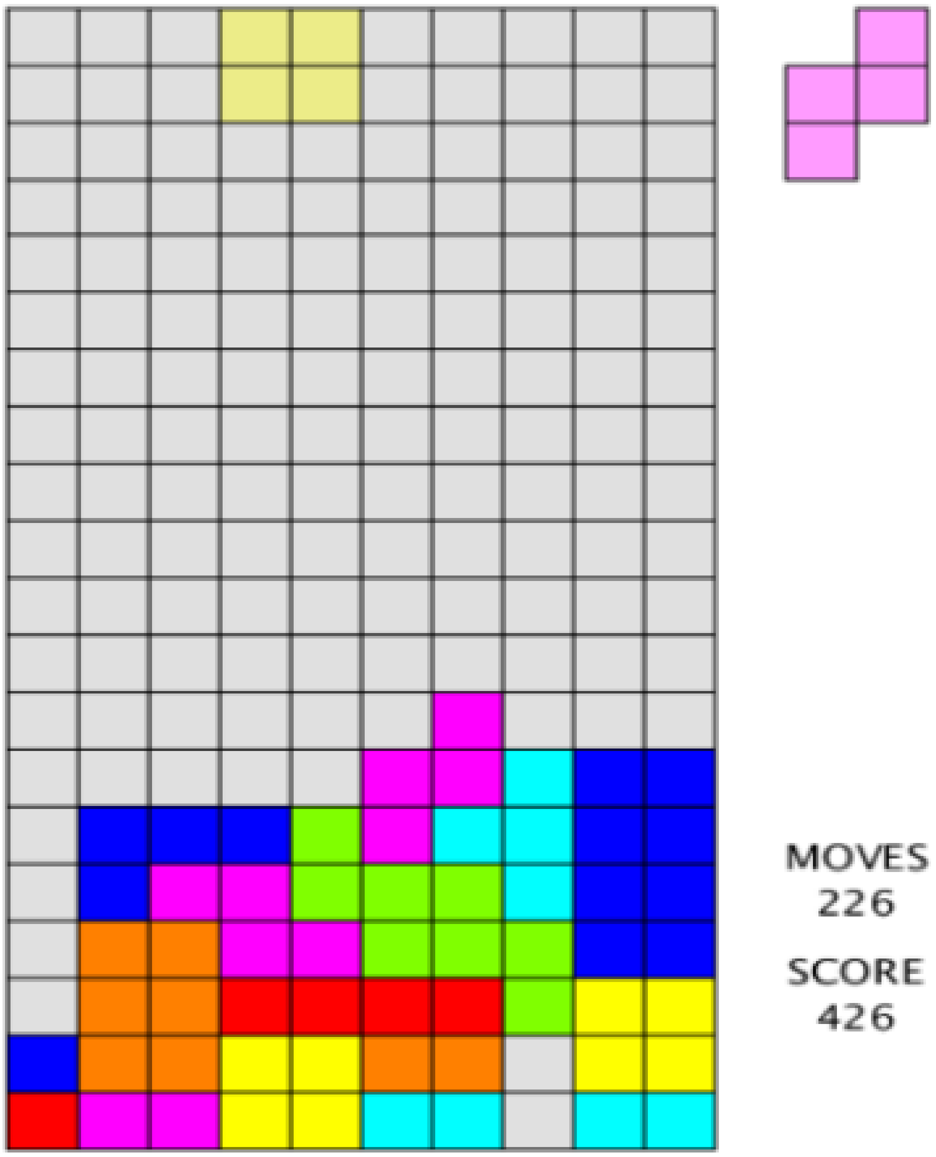


Figure 1: Tetris Gameplay

Tetris, created in 1984 by Alexey Pajitnov, is a tile-matching puzzle game where blocks (tetrominoes) fall into a grid and fill the space in grid. Each game state can be represented as:

- The current game score,
- The next piece type, and

- A 20×10 binary grid showing occupied spaces.

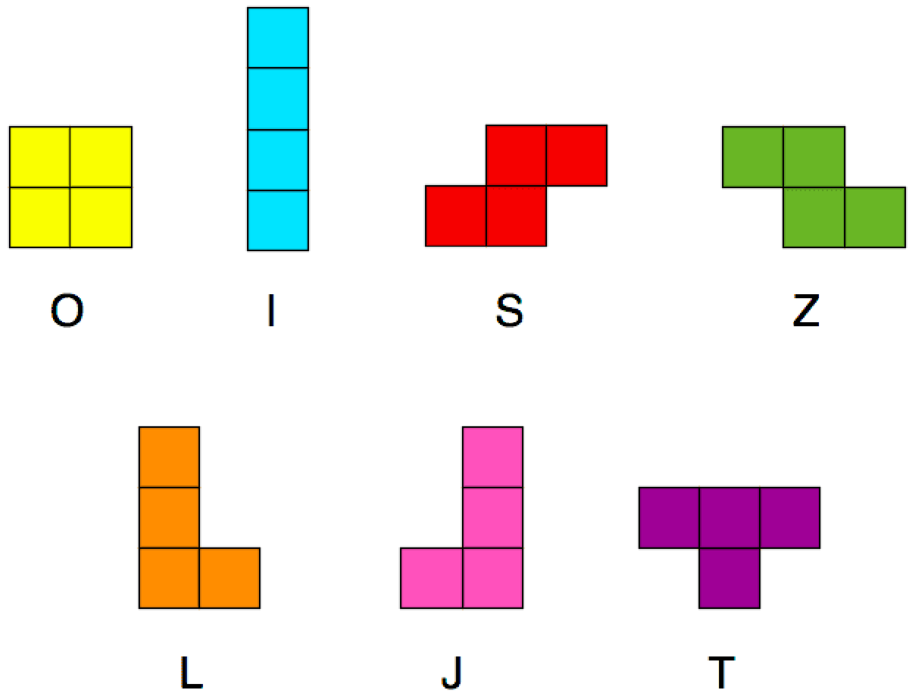


Figure 2: Tetris Pieces

At each step, there are only a finite number of possible moves. Since Tetris can be expressed mathematically as an optimization problem, it becomes suitable for algorithmic methods such as Genetic Algorithms, which simulate natural evolution to find near-optimal solutions.

2 Related Work

Application of genetic algorithms in tetris has been previously explored in multiple previous publications. All of them use the similar approach of using the feature set to evaluate the feature vectors for future Tetris states and using GA to find the best weights on those features.

- Flom & Robinson (2004) used bit-vector weights with random mutations.
- Böhm et al. (2005) tested both linear (dot product) and exponential rating functions.

- Rollinson & Wagner (2010) improved GA results using Nelder-Mead optimization.

These works generally measured success by lines cleared, not by scoring, meaning efficiency in multi-line clears (points) wasn't directly evaluated.

There has also been previous work on the offline version of Tetris, where players already know the entire fine finite tetromino(pieces) sequence in advance. Clearing the Max lines from that sequence is proved to be NP hard (Breukelaar et al., 2004).

3 Genetic Algorithms

Genetic Algorithm we attempt to find the inputs that help in maximizing our output of some function. That function is commonly known as the fitness function. The algorithm maintains a constant population of N candidates.

At the start of each iteration, the existing population represents a generation of candidate solutions, which serves as the basis for producing a new population of N individuals for the next generation.

Each generation is sequentially numbered according to how many iterations of the algorithm have been completed.

The initial population, referred to as Generation 0, is created using randomly generated inputs drawn from the defined input space. In subsequent generations, new candidates are produced by probabilistically selecting individuals from the current generation, where the likelihood of selection is directly proportional to each candidate's fitness score, as determined by the fitness function.

Once all candidates have been evaluated, the formation of the next generation occurs through three distinct phases:

- Selection: A subset of candidates is carried forward unchanged into the next generation.
- Mutation: Selected candidates are slightly modified by introducing random variations.
- Crossover: Pairs of candidates are chosen, and their corresponding parameters are combined.

The population evolves over multiple generations until convergence, aiming to maximize the fitness score.

4 Tetris Implementation (Game Rules)

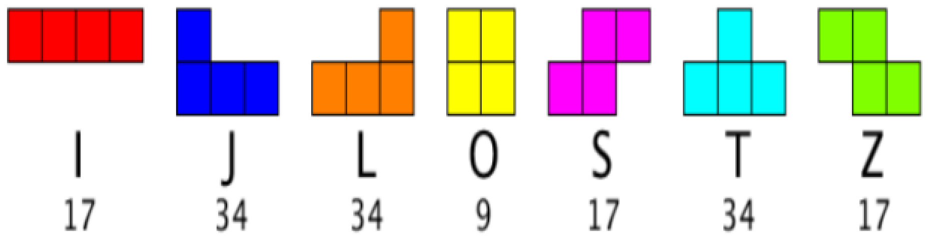


Figure 3: Max Possible Ways

The Tetris board used for this problem has the dimension of 20 x 10 and allows players to drop pieces at any column for all possible rotation of a tetromino. Resulting in fixed possible moves possible for each piece shown in figure 3.

There's no falling time constraint, assuming instant computation. The simplifications make the problem computationally efficient for GA evaluation. Also instant gravity is also not considered as it makes few moves impossible.

Score rules:

- 1 line $\rightarrow$ 2 pts
- 2 lines $\rightarrow$ 5 pts
- 3 lines $\rightarrow$ 15 pts
- 4 lines $\rightarrow$ 60 pts

5 Tetris Optimization Problem

In the Tetris optimization problem described in this paper, each candidate Tetris player is represented by a fixed weight vector in R^M, where M denotes the total number of features used to describe the current state of the game. Given the current game configuration—which includes both the active tetromino and the next piece—the player evaluates all possible future Tetris states that could arise after the next two moves.

For each possible state, the player computes the dot product between its weight vector and the game state's feature vector, selecting the move that results in the highest dot product value two moves ahead.

The overall performance of a Tetris player is assessed using two key criteria:

- the ability to make a large number of moves before losing, and
- efficiency, or the ability to score points effectively by clearing multiple lines at once, ideally achieving tetrises (clearing four lines simultaneously).

Both of these aspects are integrated into the fitness function, which quantitatively measures a Tetris player's overall performance. The fitness function is defined as the average number of points earned per move over the course of a game and can be expressed mathematically as:

$$F = \frac{S}{N_{max}}$$

where:

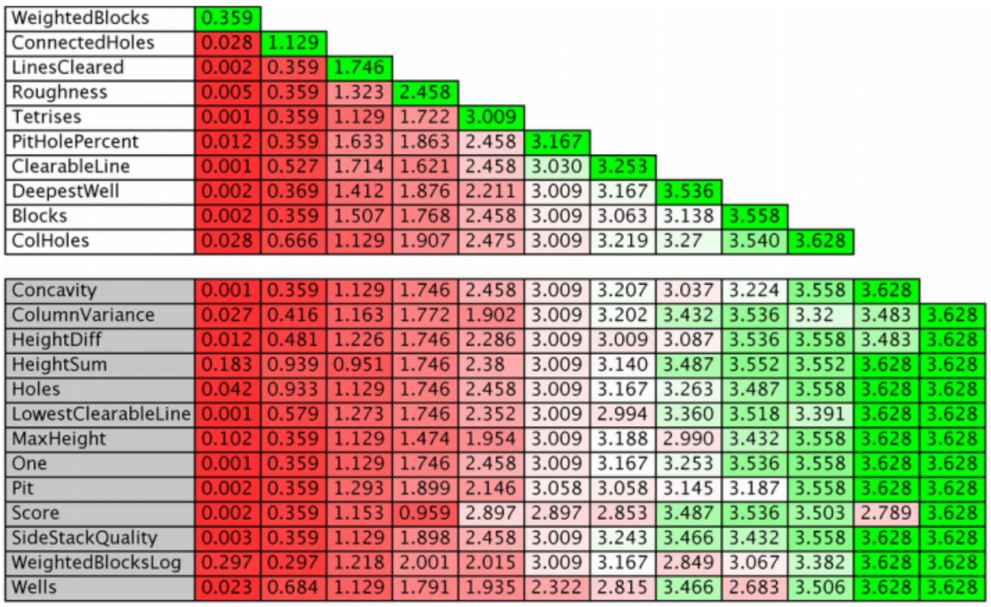
- F = fitness value of the player,
- S = total score achieved by the player, and
- N_{max} = maximum number of moves allowed in a game.

If a player reaches a game over before completing N_{max} moves, the total score S is still divided by this upper limit, ensuring that players who lose early are penalized proportionally to the number of moves missed. This prevents games from continuing indefinitely and standardizes the evaluation process across all candidates.

An additional metric used to assess player performance is the **score efficiency**, which measures how close a player's performance is to the theoretical optimum. A perfect player—one that clears only tetrises (four lines at once)—would earn 60 points every 10 moves, equivalent to an average of 6 points per move.

6 Feature Set

Feature design critically impacts performance. The author initially proposed 23 potential features and used a forward search method to identify the 10 most useful ones.



WeightedBlocks	0.359											
ConnectedHoles	0.028	1.129										
LinesCleared	0.002	0.359	1.746									
Roughness	0.005	0.359	1.323	2.458								
Tetrises	0.001	0.359	1.129	1.722	3.009							
PitHolePercent	0.012	0.359	1.633	1.863	2.458	3.167						
ClearableLine	0.001	0.527	1.714	1.621	2.458	3.030	3.253					
DeepestWell	0.002	0.369	1.412	1.876	2.211	3.009	3.167	3.536				
Blocks	0.002	0.359	1.507	1.768	2.458	3.009	3.063	3.138	3.558			
ColHoles	0.028	0.666	1.129	1.907	2.475	3.009	3.219	3.27	3.540	3.628		
Concavity	0.001	0.359	1.129	1.746	2.458	3.009	3.207	3.037	3.224	3.558	3.628	
ColumnVariance	0.027	0.416	1.163	1.772	1.902	3.009	3.202	3.432	3.536	3.32	3.483	3.628
HeightDiff	0.012	0.481	1.226	1.746	2.286	3.009	3.009	3.087	3.536	3.558	3.483	3.628
HeightSum	0.183	0.939	0.951	1.746	2.38	3.009	3.140	3.487	3.552	3.552	3.628	3.628
Holes	0.042	0.933	1.129	1.746	2.458	3.009	3.167	3.263	3.487	3.558	3.628	3.628
LowestClearableLine	0.001	0.579	1.273	1.746	2.352	3.009	2.994	3.360	3.518	3.391	3.628	3.628
MaxHeight	0.102	0.359	1.129	1.474	1.954	3.009	3.188	2.990	3.432	3.558	3.628	3.628
One	0.001	0.359	1.129	1.746	2.458	3.009	3.167	3.253	3.536	3.558	3.628	3.628
Pit	0.002	0.359	1.293	1.899	2.146	3.058	3.058	3.145	3.187	3.558	3.628	3.628
Score	0.002	0.359	1.153	0.959	2.897	2.897	2.853	3.487	3.536	3.503	2.789	3.628
SideStackQuality	0.003	0.359	1.129	1.898	2.458	3.009	3.243	3.466	3.432	3.558	3.628	3.628
WeightedBlocksLog	0.297	0.297	1.218	2.001	2.015	3.009	3.167	2.849	3.067	3.382	3.628	3.628
Wells	0.023	0.684	1.129	1.791	1.935	2.322	2.815	3.466	2.683	3.506	3.628	3.628

Figure 4: Feature Set

Terms used to explain the feature:

- **block** – a filled space on the board
- **hole** – an unfilled space that is located below a filled space
- **pit** – an unfilled space that is not a hole and has a filled space (or the board edge) on its left and right
- **column height** – the row where a column's highest block is located

7 Top 10 Selected Features

1. WeightedBlocks – Weighted sum of blocks by their row. Where weight is the row it is on.
2. ConnectedHoles – Number of vertically connected holes.
3. LinesCleared – Lines cleared so far.
4. Roughness/Bumpiness – Difference between adjacent column heights.
5. Tetrises – Number of 4-line clears.

6. PitHolePercent – Ratio of pits to total unfilled spaces.
7. ClearableLine – Max lines clearable by an "I" piece.
8. DeepestWell – Row of the lowest unfilled space.
9. Blocks – Total number of blocks.
10. ColHoles – Columns containing at least one hole.

After 10 features, no new feature improved results (efficiency plateaued at 3.628).

8 Algorithm Results

After determining the optimal feature set of size $M = 10$, the Genetic Algorithm was executed for 30 generations with a population size of $N = 1000$ candidates.

Setup:

- Feature Set Size (M): 10
- Population Size (N): 1000
- Number of Generations: 30
- Candidate Selection Process:
  - Selection: 20% of the candidates were directly carried forward to the next generation without modification.
  - Mutation: 40% of the candidates were altered slightly to introduce random variations and maintain diversity in the population.
  - Crossover: 40% of the candidates were generated by combining pairs of parent solutions to produce new offspring.

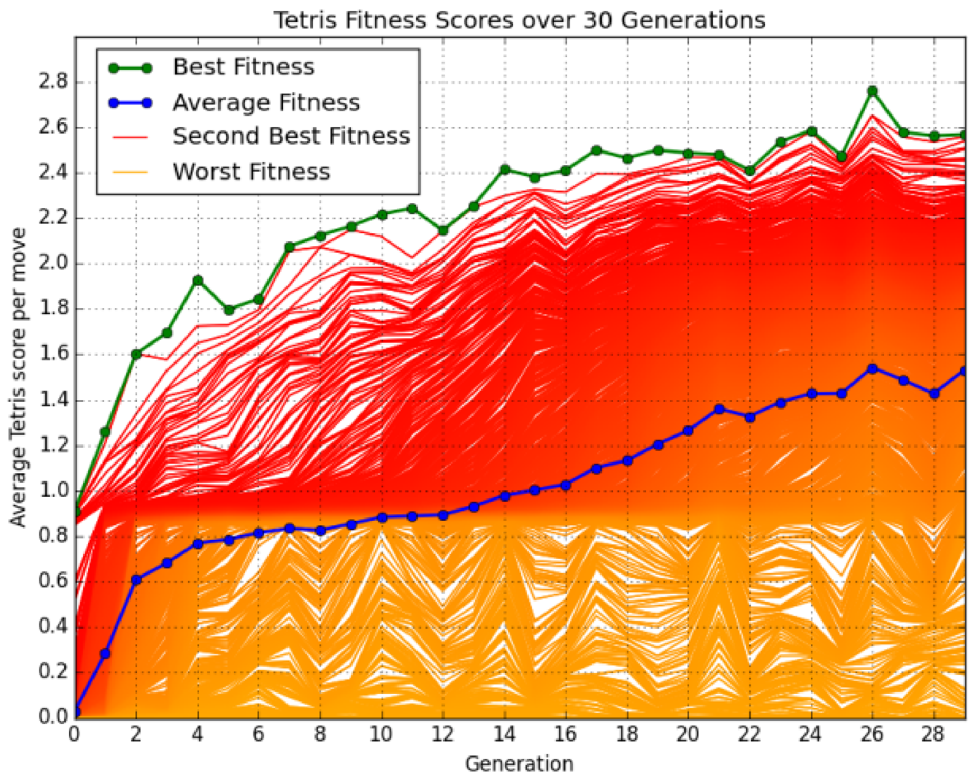


Figure 5: fitness evaluation

The graph in Figure 5 illustrates the progression of fitness values across all 30 generations of the algorithm. It visualizes the performance of the 1,000 candidates evaluated in each generation, highlighting the best fitness, average fitness, second-best fitness, and worst fitness scores. The color gradient, ranging from red (second-best) to bright orange (worst fitness), effectively demonstrates the overall trend of improvement throughout the evolutionary process. As observed, both the best and average fitness values increase steadily with each generation, indicating that the population, as a whole, evolves toward higher-performing solutions over time.

9 Conclusion

This work demonstrated that the game of Tetris can be effectively modeled as an optimization problem and solved using Genetic Algorithms. By evolving the weights of a set of carefully selected state features, the algorithm successfully produced an autonomous player capable of sustained and efficient gameplay. Over 30 generations of training, the evolved players consistently improved their fitness, culminating in an average performance of over 179,000 moves per game.

The results highlight the importance of feature selection and balanced exploration–exploitation in evolutionary search. Even with a relatively small feature set, the GA learned a robust strategy that generalizes well to unseen game states. While the current approach does not account for long-term planning beyond two moves or dynamic adaptation during play, the strong performance demonstrates the viability of GAs in complex, stochastic environments like Tetris.

ORIGINAL PAPER: `https://cs229.stanford.edu/proj2015/238_report.pdf`