Playing Tetris with Genetic Algorithms

DSE106: Soft Computing

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

MSC 1st Year 2025

November 11, 2025

Outline

What is Tetris?

- Classic tile-matching video game (1985)
- Stack falling tetrominoes (7 shapes)
- Clear horizontal lines
- Simple rules, complex strategy

Computational Problem:

- Given: board size ($W \times H$), piece sequence
- Goal: Maximize lines cleared

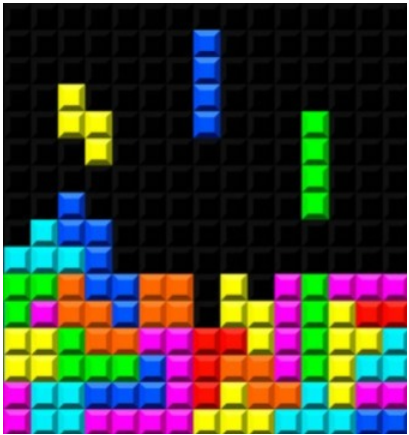
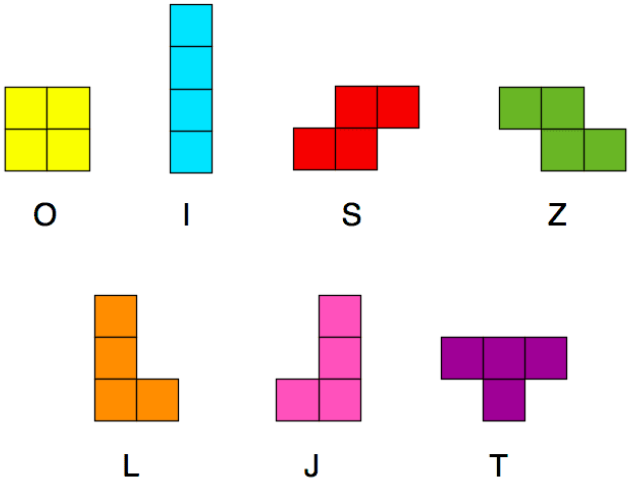


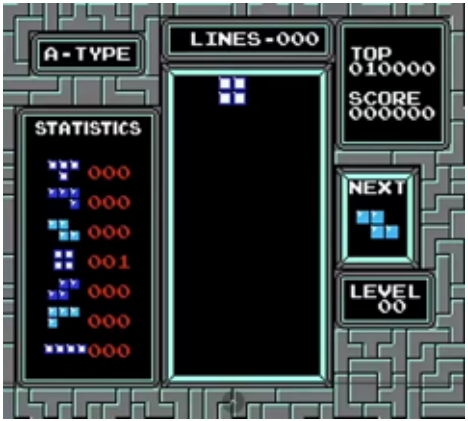
Figure: Tetris gameplay

Tetris Pieces (Tetrominoes)



Standard Tetris Tetrominoes

Tetris Gameplay



Animated demonstration of Tetris gameplay.

Tetris Optimization Problem

In the Tetris optimization problem, each candidate Tetris player is represented by a fixed weight vector in $\mathbb{R}^M$, where M denotes the number of features describing the current game state.

Given the current configuration (active and next tetromino), the player evaluates all possible future states after the next two moves. For each possible state, the player computes the dot product between its weight vector and the corresponding feature vector:

$$\text{Score}(\text{State}) = \mathbf{W} \cdot \mathbf{F}$$

The move selected is the one that results in the highest score.

The performance of a Tetris player is measured based on two factors:

- **Longevity:** Ability to make a large number of moves before losing.
- **Efficiency:** Ability to score effectively by clearing multiple lines, ideally achieving tetrises.

Fitness Function and Metrics

Both aspects are captured by the **fitness function**, defined as the average number of points earned per move:

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

where:

- F – Fitness value of the player
- S – Total score achieved
- N_{max} – Maximum allowed number of moves

If a player loses early, the score S is still divided by N_{max}, penalizing premature losses.

Feature Set Design

Feature design is crucial for optimization performance. The author proposed 23 features and used a forward search to identify the 10 most effective.

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: Feature Selection Process

Feature Selection (Part 1)

Terminology:

- **Block:** Filled cell on the board
- **Hole:** Empty cell below a filled block
- **Pit:** Empty cell enclosed on both sides by filled blocks
- **Column Height:** Row of the highest filled cell in a column

Top 10 Selected Features (Part 1):

1. **WeightedBlocks** – Weighted sum of blocks by their row, where weight is the row index.
2. **ConnectedHoles** – Number of vertically connected holes.
3. **LinesCleared** – Total lines cleared so far.
4. **Roughness/Bumpiness** – Sum of absolute differences between adjacent column heights.
5. **Tetrises** – Number of 4-line clears achieved.

Feature Selection (Part 2)

Remaining Selected Features (Part 2):

6. **PitHolePercent** – Ratio of pits to total unfilled spaces.
7. **ClearableLine** – Maximum number of lines that can be cleared by an "I" (straight) piece.
8. **DeepestWell** – Row of the lowest unfilled space that is not a hole.
9. **Blocks** – Total number of blocks on the board.
10. **ColHoles** – Number of columns containing at least one hole.

After identifying these 10 features, further additions showed no improvement, with efficiency plateauing at 3.628.

GA process (from paper)

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

Experimental Setup:

- Feature Set Size (M): 10
- Population Size (N): 1000
- Number of Generations: 30

Candidate Selection Process:

- 20% Selection – Top candidates carried forward unchanged
- 40% Mutation – Random variation introduced to preserve diversity
- 40% Crossover – New offspring formed from pairs of parent solutions

This setup ensures a balanced exploration and exploitation strategy for effective evolution.

Fitness Graph Interpretation

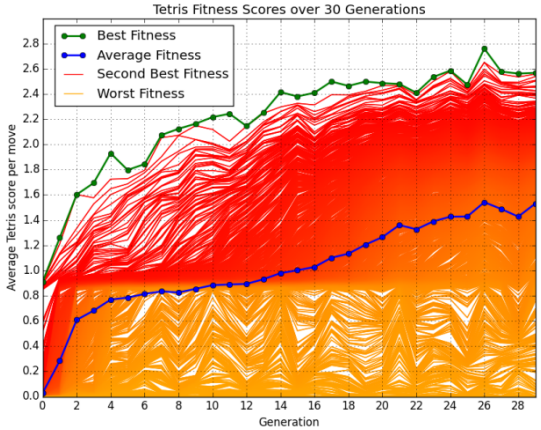


Figure: Evolution of Fitness Values across 30 Generations

Interpretation:

- The graph illustrates the evolution of fitness values across 30 generations for 1,000 candidates.
- It highlights the **best**, **average**, **second-best**, and **worst** fitness scores in each generation.
- A color gradient—ranging from red (second-best) to bright orange (worst)—shows the population's overall trend of improvement.
- Both best and average fitness values rise consistently, indicating that the algorithm successfully evolves higher-performing players over successive generations.

Overview of Our Genetic Algorithm Setup

Board & Pieces (same as paper):

- Board size: $H \times W = 20 \times 10$
- Tetrominoes used: `I, O, T, J, L` (each with all possible rotations)
- Realistic spawn rule – piece cannot appear if blocked at the top

Key Difference from the Paper:

- Instead of 10 features, we use a smaller and simpler **4-feature model**.
- Each player is represented by a **weight vector** $\mathbf{w} = [w_1, w_2, w_3, w_4]$.
- Each board state is represented by a **feature vector** $\mathbf{f} = [f_1, f_2, f_3, f_4]$.

Decision Making (Greedy Selection):

$$\text{Score} = \mathbf{w} \cdot \mathbf{f} = w_1 f_1 + w_2 f_2 + w_3 f_3 + w_4 f_4$$

- For every possible move, the player calculates this score.
- The move with the **highest score** is chosen.

Our Feature Set (4-D Features)

The Genetic Algorithm learns the best combination of weights for these 4 features:

1. **Aggregate Height:** Total of all column heights on the board. $\Rightarrow$ Lower is better, as tall stacks lead to game over.
2. **Move Score:** NES-style points gained for clearing lines in a move. $\Rightarrow$ Higher is better.
3. **Holes:** Empty spaces that have blocks above them. $\Rightarrow$ Lower is better, as holes block future clears.
4. **Bumpiness:** Difference in heights between neighboring columns. $\Rightarrow$ Lower is better, smoother surfaces are easier to fill.

Linear Evaluation Formula:

$$\text{Value(state)} = w_1 \cdot \text{Height} + w_2 \cdot \text{Score} + w_3 \cdot \text{Holes} + w_4 \cdot \text{Bumpiness}$$

The better the combination of weights, the smarter the move decisions.

Fitness Function and Evaluation Metric

Goal: To evolve weights that maximize the player's average score per move.

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

Where:

- S = total score earned in a game
- N_{max} = maximum number of allowed moves (piece budget)

NES-like Scoring Rule:

$$\text{Lines Cleared: } 1 \to 2, \quad 2 \to 5, \quad 3 \to 15, \quad 4 \to 60$$

Even if the player loses early, the total score is divided by N_{max}, so early losses are automatically penalized.

Training Configuration (Step-by-Step)

Evaluation per Genome:

- Each weight vector (genome) plays multiple games to measure performance.
- `num_games = 10` (10 games per genome)
- `n_pieces = 100` (each game uses 100 pieces)

Genetic Algorithm Parameters:

- **Population Size:** 20 Number of weight vectors in each generation.
- **Generations:** 10 Number of times the population evolves.
- **Offspring Fraction:** 30% New children generated each generation through crossover.
- **Mutation Probability:** 5% Small random change applied to one weight to explore new possibilities.

Evolution Loop Explained

The GA runs in the following loop:

1. **Initialize:** Start with 20 random weight vectors on a unit sphere.
2. **Evaluate:** Each weight vector plays 10 games. Calculate its average fitness $F = S/N_{\text{max}}$.
3. **Selection:** Pick the top-performing individuals to act as parents.
4. **Crossover:** Combine two parent weights (weighted by fitness):

$$\text{child} = \text{normalize}(p_1 \cdot f_1 + p_2 \cdot f_2)$$

5. **Mutation:** With 5% probability, slightly modify one random weight to add diversity.
6. **Replacement:** Keep the best 70% of old population and replace the rest with new children.

Generational Evolution Process

Across 10 Generations:

- Each generation learns slightly better weight combinations.
- The best individuals survive longer and dominate the population.
- Over time, average fitness improves steadily.

Final Evaluation:

- After 10 generations, all individuals are evaluated again.
- The best weight vector is saved as the final learned player.
- This vector defines how strongly each feature affects the move choice.

Our Fitness Graph

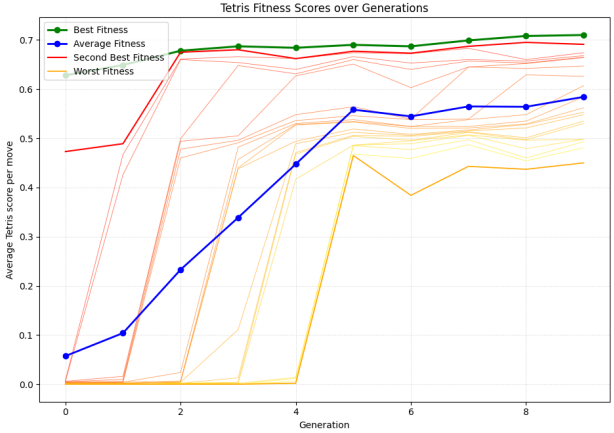


Figure: Fitness over generations (our GA)

How to read the plot

- **Green** = Best fitness in each generation.
- **Blue** = Average fitness.
- **Red** = Second-best fitness.
- **Orange** = Worst fitness.
- Faint lines show other individuals ("spaghetti" trend).

Takeaways

- Best and average scores rise steadily.
- Population spreads early, then stabilizes.
- Our 4-feature GA learns a reasonable policy.

Implementation Link

GitHub Links:

`https://github.com/hritikjena/Playing-Tetris-with-Genetic-Algorithms`

or

`https://github.com/Ankitchawla3123/SOFT-COMPUTING-PROJECT`

Conclusion

- A simple **4-feature** model + **GA** can learn effective Tetris play.
- Using the paper's fitness $F = S/N_{\text{max}}$ lets us compare fairly.
- Results show steady improvement in best and average fitness over generations.
- **Practical benefits:** fast greedy play, interpretable weights, easy to extend.

Future Work

- Try larger populations / more generations for stronger policies.
- Add More Features.
- Reduce variance (more evaluation games) and track seed-controlled runs.

References

Jason Lewis (2015). *Playing Tetris with Genetic Algorithms.* Stanford CS229 Project Report. https://cs229.stanford.edu/proj2015/238_report.pdf

Codemyroad Blog (2013). *Tetris AI: The Near-Perfect Player.* https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/