Algorithm Explanation:

Initialization:
The algorithm starts by initializing a population of chromosomes, where each chromosome represents a set of weights used to evaluate potential moves in the Tetris game.

Evaluation:
Each chromosome in the population is evaluated by running the Tetris game with the corresponding set of weights. The fitness of each chromosome is determined based on the score achieved in the game.

Selection:
Parents for the next generation are selected from the current population. In this implementation, roulette wheel selection is used, where chromosomes with higher fitness have a higher probability of being selected as parents.

Crossover:
Crossover is performed on selected parents to generate offspring for the next generation. Arithmetic crossover is applied, where new chromosomes are created by combining the genes (weights) of the selected parents.

Mutation:
Mutation is applied to introduce diversity in the population. Random mutation is performed on the offspring chromosomes by randomly altering some of their weights.

Replacement:
The new offspring population replaces the old population, ensuring that the best-performing individuals are retained for the next generation.

Iteration:
The above steps are repeated for a certain number of iterations, with each iteration consisting of multiple generations.

Improvement Strategy:

The algorithm aims to find optimal weights that can guide the Tetris game-playing agent to make better decisions, leading to higher scores in the game.
By iteratively evaluating and evolving populations of chromosomes, the algorithm explores the search space of possible weight combinations and converges towards solutions that maximize the game score.

Additional Findings:

Seed Selection:
The choice of random seed can significantly impact the performance and convergence behavior of the genetic algorithm.
Experimenting with different seeds may lead to variations in the results obtained.

Algorithm Parameters:
Parameters such as population size, number of generations, crossover rate, and mutation rate play crucial roles in determining the algorithm's performance.
Fine-tuning these parameters through experimentation and analysis can lead to improved results.

Convergence Analysis:
Analyzing the convergence behavior of the algorithm, such as the rate of improvement in the best scores over iterations, can provide insights into its effectiveness and efficiency.

Optimal Factor:
After completing the training iterations, the algorithm saves the optimal chromosome values, representing the best-performing set of weights found during the training process.
Testing this optimal factor in a separate final test run can provide a measure of its performance in comparison to other factors.

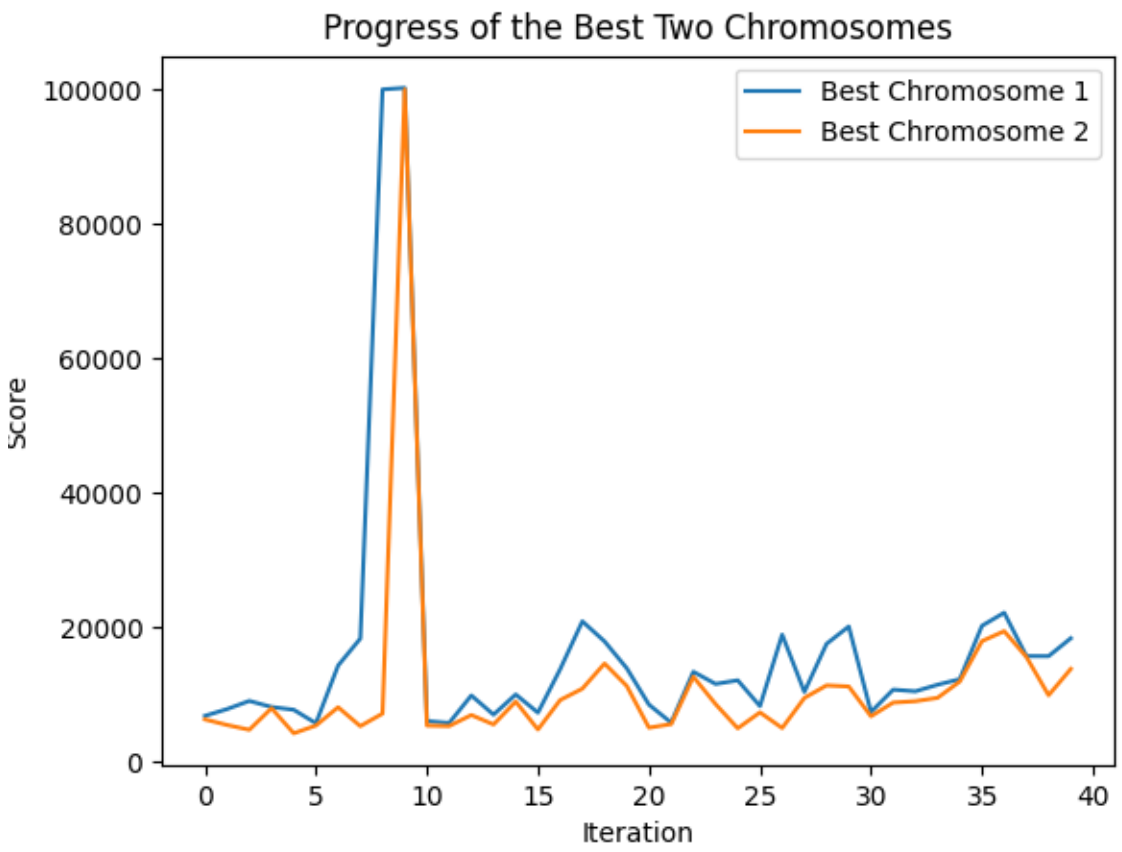
Result Summary:

Random Seed: The random seed we used is 42.

Best Chromosomes:
Chromosome 1: [-0.08933281 -3.71478873 -2.81787783 0.76258522 2.84972008 -0.12998827 2.20642401] , Score: 100023
Chromosome 2: [-0.08933281 -3.71478873 -2.81787783 0.86994324 2.84972008 -0.12998827 2.20642401] , Score: 100245

Optimal Factor Score: 100299

Graph of Best Chromosomes' Progress:



Conclusion:

The genetic algorithm presented here demonstrates an iterative optimization approach to finding effective weight combinations for guiding Tetris gameplay.

By iteratively evaluating, selecting, and evolving populations of chromosomes, the algorithm aims to improve gameplay performance over time.

Fine-tuning algorithm parameters and analyzing convergence behavior can further enhance its effectiveness in finding optimal solutions.