Cairo University, Faculty of Artificial Intelligence

FACULTY OF COMPUTERS AND ARTIFICIAL INTELLIGENCE, CAIRO UNIVERSITY

AI332: Computational Cognitive Science

2023-2024

Second Semester

Course Project, Task 1 Report

Course Instructors:

Dr. Kamal El-Damerdash

Dr. Waleed Mohamed Azmy

Authors

Basma Mahmoud Hashem	20210091
Nourhan Mahmoud Eldesouky	20211107
Selsabeel Asim Ali Elbagir	20210714
Abdelrahman Mohamed Ali	20210518
Roaa Fathi Nada	20210140
Hesham Mahmoud Ahmed	20211110

AI332: Computational Cognitive Science
Course Project, Task 1 Report

Cairo University, Faculty of Artificial Intelligence

Genetic Algorithm (GA) is a nature-inspired optimization algorithm (NIOA) that mimics the process of natural selection to find the optimal solution to a problem. In our case, the problem is to find the best combination of factors or chromosomes to improve gameplay.

Algorithm Overview:

The Tetris AI algorithm uses a genetic algorithm to evolve a set of heuristic weights that guide the AI player's decisions. These weights evaluate the game state and determine the best move to make. The algorithm iteratively evaluates different chromosomes (sets of weights), selects the top performers, and evolves the population towards better gameplay.

Chromosome Representation:

Each chromosome consists of 8 floating-point values representing heuristic weights:

1. **Removed Lines**: Tracks the number of complete lines removed from the game board, with corresponding score bonuses.
2. **New Holes**: Represents the increase in empty spaces created on the board after piece placements.
3. **Lowest Clearable Line**: Identifies the lowest row on the board that can be cleared by removing complete lines.
4. **New Blocking Blocks**: Tracks the obstructive block increase that limits player moves.
5. **Piece Sides**: Refers to the sides of the falling Tetris piece in contact with other blocks or edges.
6. **Floor Sides**: Represents the sides of the piece in contact with the bottom floor of the board.
7. **Wall Sides**: Indicates the sides of the piece in contact with the walls of the game board.
8. **Weighted Blocks Log**: Calculates a value based on block distribution and concentration on the board.

AI332: Computational Cognitive Science
Course Project, Task 1 Report

Cairo University, Faculty of Artificial Intelligence

Algorithm steps and key components:

1. Initialization: We randomly initialize a population of chromosomes with weights within the range of “-10 to 10” with population size = 14, and number of generations = 50. Each chromosome represents a potential solution, with each gene representing a factor influencing gameplay represented by the function “initialize_population”.

2. Evaluation: We evaluate each chromosome's fitness by running simulations of the gameplay with the corresponding factors to measure how well the gameplay performs “Scores” with the given factors. The `calc_best_move()` function evaluates different moves based on the current game state and the chromosome's weights to determine the best move.

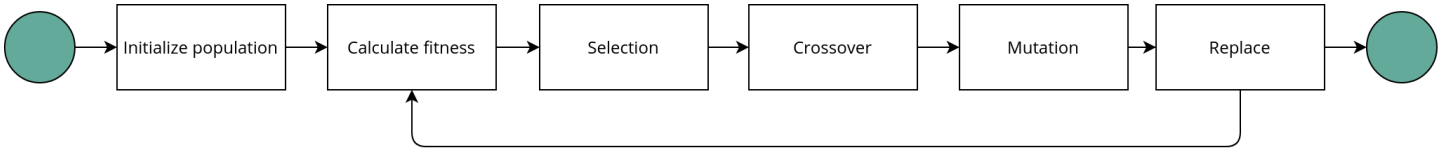
3. Selection: The `select_parents()` function selects the top-performing chromosomes based on their scores for the next generation. This selection process is biased towards selecting chromosomes with higher fitness scores.

4. Crossover: We perform crossover operations on selected chromosomes to create offspring for the next generation using the “crossover function”. This involves combining genetic material from two parent chromosomes to create new solutions.

5. Mutation: To introduce diversity into the population, we apply mutation operators to some offspring chromosomes. This helps prevent premature convergence to a suboptimal solution. We used mutation rate = 0.2, which indicates a 20% probability for each chromosome to undergo mutation. The mutation involves assigning a value from -10 to 10. Done using the “mutate” function.

6. Replacement: The new generation replaces the old one, and the process iterates until a termination condition is met, such as reaching a maximum number of generations or achieving a satisfactory score level.

7. Game Mechanics: The algorithm handles piece movement, rotation, falling, collision detection, line completion, scoring, and level calculation.



Cairo University, Faculty of Artificial Intelligence

The changes we've made (added) to the game code for better performance:

1. calc_move_info:

- This function sets the rotation and position of the piece based on the input parameters.
- Checks if the new position of the piece is valid on the board.
- Iterates the piece down while it remains in a valid position.
- Creates a hypothetical board to simulate the move and evaluate the impact.
- Returns a list of values that provide information about the move, such as whether it's a valid position.

2. draw_status:

- This function draws the score and level information on the screen.
- Optionally displays the generation number and maximum score.
- Renders the score, level, and additional information on the game screen for the player to view.

3. X_MARGIN Variable:

- It's used to determine the horizontal margin for the game board. It calculates the distance between the left edge of the game window and the left edge of the game board. It ensures that the game board is centered horizontally within the game window. It is used in various functions to position game elements correctly on the screen and aesthetically within the game window.

4. *AI_game_mode*:

- This function is responsible for running the Tetris game automatically using a given chromosome for the AI player.
- It initializes the game environment, sets up the game board, and controls the flow of the game. The function initializes the Pygame environment, sets the display mode, and defines fonts for text rendering. It sets the frames per second (FPS) for the game loop and creates a blank game board.

- The function enters a continuous game loop to manage the gameplay. It handles events such as quitting the game and updates the game state accordingly. The game loop continues until one of the following conditions is met:
    1. The score exceeds the specified max_score (default is 200,000)
    2. The game is over (i.e., a new piece cannot be placed on the board)

  Inside the game loop, the function manages the falling piece and the next piece to be played in the game. It checks if the falling piece has landed and updates the game board accordingly. If the falling piece has landed, it checks for complete lines and removes them, updating the score based on the number of lines removed.

- The function calls the calc_best_move function to determine the best move for the AI player based on the provided chromosome. The AI player automatically plays the game by calculating the best move using the chromosome. It updates the game state, including the score, level, and falling pieces. and it draws the game board, the next piece, and the falling piece on the screen using Pygame. It also updates the game status, such as the score and level, on the screen.

AI332: Computational Cognitive Science
Course Project, Task 1 Report

Cairo University, Faculty of Artificial Intelligence

Progress of Best Two Chromosomes:

Seed and Best Chromosomes:

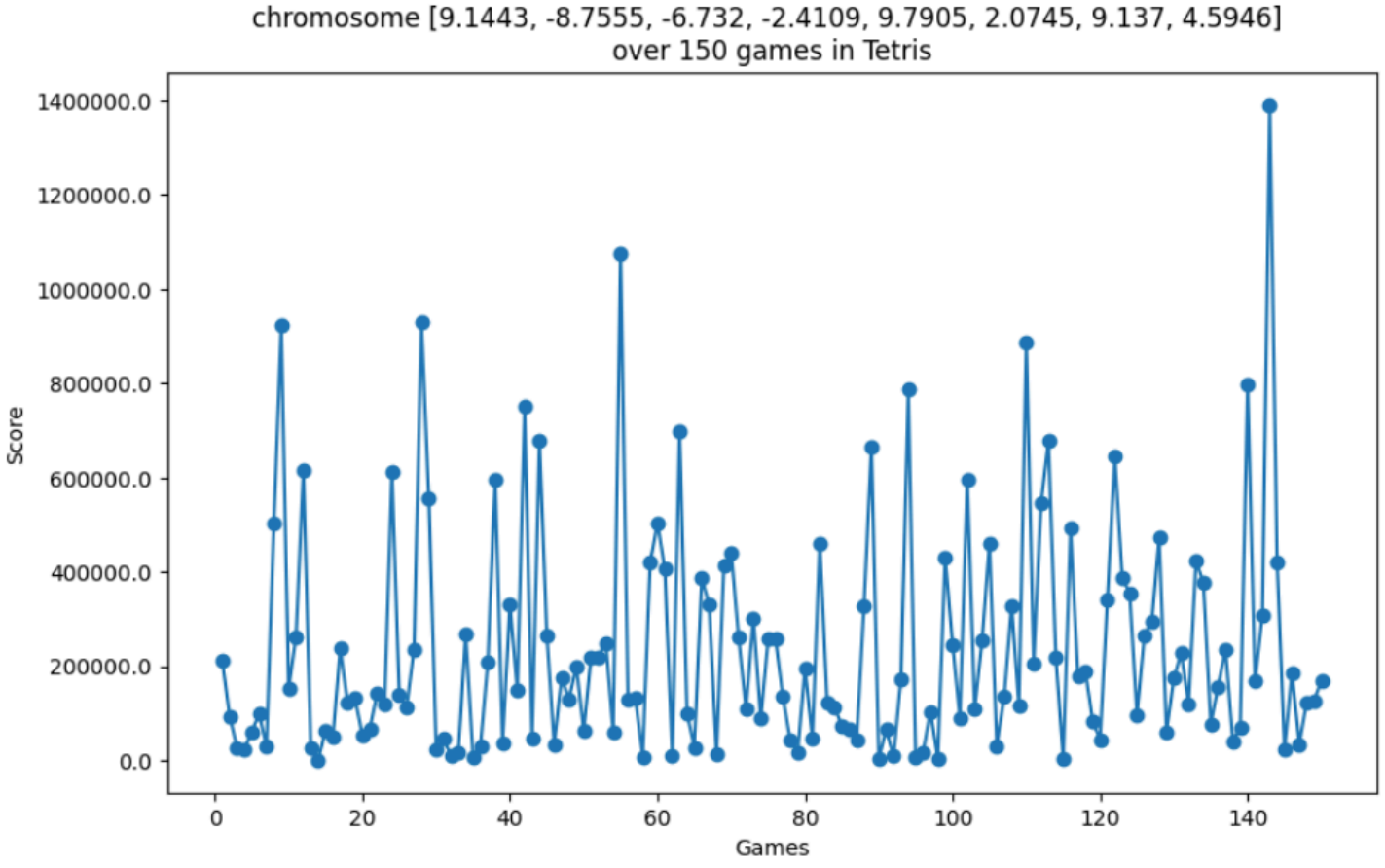
Training:

- Seed: 42

Best Two Chromosomes that achieved the goal of 200000 score: [9.1443, -8.7555, -6.732, -2.4109, 9.7905, 2.0745, 9.137, 4.5946], [9.1443, -4.7405, -8.1451, -0.5974, 6.9499, 2.0745, 6.1426, -4.7322]

Testing (tried different seeds):

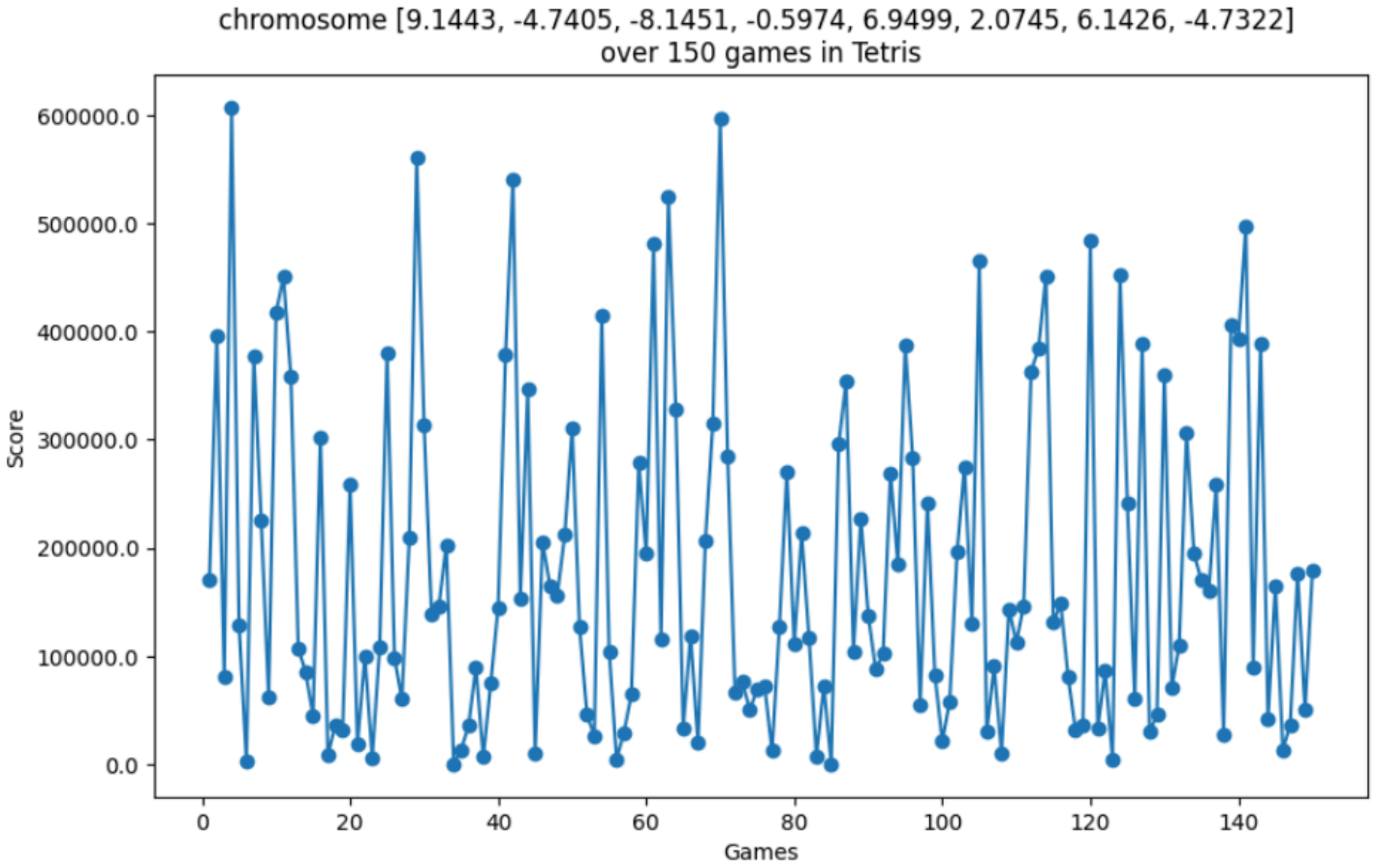
- Best Chromosome 1: [Factors =9.1443, -8.7555, -6.732, -2.4109, 9.7905, 2.0745, 9.137, 4.5946] - Score: [1391029] - Seed: 42



AI332: Computational Cognitive Science
Course Project, Task 1 Report

Cairo University, Faculty of Artificial Intelligence

- Best Chromosome 2: [Factors: = 9.1443, -4.7405, -8.1451, -0.5974, 6.9499, 2.0745, 6.1426, -4.7322] - Score: [596475] - Seed: 333



AI332: Computational Cognitive Science
Course Project, Task 1 Report

Cairo University, Faculty of Artificial Intelligence

Optimal Factor Score:

After conducting the final test run with more iterations, the optimal factor [agent with weights [9.1443, -8.7555, -6.732, -2.4109, 9.7905, 2.0745, 9.137, 4.5946] achieved a score of [Score = 1391029]. This serves as a benchmark for evaluating the effectiveness of our genetic algorithm in gameplay improvement.

Additional Findings:

Throughout the optimization process, several interesting observations and insights were noted:

- The impact of individual factors on gameplay performance varied significantly, highlighting the complexity of optimizing gameplay.

- Mutation played a crucial role in maintaining diversity within the population, preventing premature convergence, and promoting exploration of the solution space. The fixed mutation rate encourages exploration by introducing random changes in genetic material, potentially uncovering hidden patterns or solutions that might not be apparent with deterministic selection methods like the roulette wheel.

- Crossover operations facilitated the exchange of beneficial genetic material between chromosomes, leading to the emergence of novel and potentially superior solutions.

- With smaller population sizes, the algorithm converges faster.

In conclusion, our genetic algorithm demonstrates promise in optimizing gameplay by efficiently exploring the solution space and identifying configurations that enhance performance. Further refinements and experiments can lead to even better results and insights into gameplay optimization strategies.

AI332: Computational Cognitive Science
Course Project, Task 1 Report

Cairo University, Faculty of Artificial Intelligence