Computational Cognitive Science
Course Project
Second Semester

Tetris Game

Team members

- Maria Mohamed Bassam ID: 20210786
- Omar Gaafar ID: 20210263
- Marya Zuhair ID: 20210785
- Mostafa Gamal ID: 20210391
- Nour Eldeen Ahmed ID: 20210431
- Abdelrahman Yasser Ali ID: 20210564


Tetris AI Menu
Welcome to FCAI Tetris Game!
Play Tetris Normally
Start The Genetic Algorithm
Start Testing (Beast Mode)

Contents

Getting Started

Important Jargons

1. Population:
    - In a genetic algorithm, the population refers to a group of individual solutions (often represented as chromosomes) that are used to explore the solution space of a given problem. Each individual in the population represents a potential solution to the problem being solved.

2. Chromosome:
    - A chromosome is a data structure that represents a potential solution to the problem being solved by the genetic algorithm. It is typically encoded as a string of binary digits (genes) or other data types, where each gene corresponds to a specific attribute or parameter of the solution.

3. Iteration:
    - An iteration, also known as a generation or epoch, represents a single cycle or step in the genetic algorithm's iterative process. During each iteration, the algorithm evaluates the fitness of individuals in the population, selects individuals for reproduction based on their fitness, applies genetic operators such as crossover and mutation to produce offspring, and updates the population for the next iteration.

4. Crossover:
    - Crossover, also known as recombination, is a genetic operator used to create new offspring by combining genetic material from two parent chromosomes. It mimics the process of reproduction in nature by exchanging genetic information between parent chromosomes to produce offspring with characteristics inherited from both parents. Crossover promotes genetic diversity within the population and facilitates the exploration of the solution space.

5. Mutation:
    - Mutation is a genetic operator that introduces random variations or alterations to individual chromosomes within the population. It serves as a mechanism for introducing diversity and preventing premature convergence to suboptimal solutions. Mutation involves randomly changing one or more genes in a chromosome, typically by flipping the value of a gene or making small adjustments to its value. This helps to explore new regions of the solution space and potentially uncover better solutions

Launching the Application

Upon launching our Tetris Game Application, users are greeted with an inviting interface that sets the stage for an immersive gaming experience.
Giving the Player a choice of 3:

- **Play Tetris Normally** : This mode offers the traditional Tetris experience without any algorithmic interference. Players can enjoy the game at their own pace, utilizing their strategic skills to achieve high scores.

- **Start the Genetic Algorithm** : Experience the power of genetic algorithms in Tetris gameplay. This mode provides insight into the workings of our genetic algorithm, showcasing the current evaluation, chromosome utilization, scores, moves, and relevant statistics for each run. Witness the evolutionary process firsthand as the algorithm refines its strategies to optimize performance.

- **Start Testing (Beast Mode)** : Step into the ultimate challenge with Beast Mode. Wondering why it's called "Beast Mode"? Let us explain.In Beast Mode, our smart algorithm takes the lead, showing off its skills by playing Tetris without **ever losing**.The algorithm doesn't just play; it plays smart, achieving scores that go beyond imagination. It's unusual to see scores in the millions, all while making over 50,000+ moves. But bear with it, as you might find it slower. Beast Mode involves massive calculations to make those incredible moves

Algorithm Overview

In the 10x20 Tetris game board, players receive Tetris pieces to strategically position on the board. With each piece placement, we calculate some attributes which are called “Fitness metrics”. which assists in determining the best move.

Fitness metrics

- **Max Height**: The maximum height of the Tetris stack after placing the piece.
- **Number of Removed Lines**: The number of lines cleared by placing the piece.
- **New Holes**: The number of empty spaces created beneath placed blocks.
- **Piece Slides**: Describes the number of sides of the Tetris piece that are in contact with other occupied cells
- **Floor Slides**: Indicates the number of sides of the Tetris piece that are in contact with the bottom floor of the board.
- **New Blocking Blocks**: Blocking blocks are occupied cells that block potential moves or create difficult-to-fill gaps
- **Wall Slides**: Represents the number of sides of the Tetris piece that are in contact with the walls of the board

How we are using the Fitness metrics

Our algorithm aims to identify the optimal values (represented as a chromosome with 7 values, e.g., [a,b,c,d,e,f,g]) to be multiplied by these factors

For each Piece , you will have a lot of moves available to play with different rotations , how will you choose between them ?

Here, the fitness function plays a crucial role. After calculating the contribution factors for each dropped piece, we proceed to evaluate the fitness of each move. This involves multiplying the values of the chromosomes with the corresponding contribution factors for each move. Subsequently, we select the move with the highest score to be played

The below image shows the Fitness Function

$$Fitness = a * MaxHeight + b * LinesRemloved + c * NewHoles + d * newBlockingBlocks + e * PieceSides + F * FloorSides + g * WallSides$$

Algorithm Starting

To start our algorithm, we begin by initializing a population consisting of 12 random chromosomes, each comprising seven values randomly selected from the range [-10, 10]. These chromosomes represent potential solutions for optimizing gameplay.

Next, we execute the game using the current chromosomes and record the score achieved by each chromosome. This process represents a single iteration of the algorithm.

Following the completion of an iteration, we proceed to select the top 50% of chromosomes based on their overall scores. These top-performing chromosomes are chosen to undergo crossover, resulting in the creation of a new population.

With a mutation rate of 10%, we introduce random variations to the selected chromosomes. This involves selecting a value within a chromosome, dividing it by 2, and adding a random value between [0, 1]. Additionally, we incorporate a random number between [-2, 2] to further diversify the chromosome's genetic makeup.

Upon completion of these genetic operations, we obtain a new population comprising 12 chromosomes, ready to undergo gameplay in the subsequent iteration. This iterative process continues, allowing the algorithm to evolve and refine its strategies over multiple generations.

What makes our algorithm unique

The Idea and how it works

We have noticed that the next Piece (the piece that will be played after the falling piece falls) is known prior. So we have used it in order to optimize the choice of the move to be considered. How is that ?

For each possible move with the current falling piece, we simulate placing it in every position on the game board. This includes rotations and translations to explore all potential placements.

With the current piece in various positions on the board, we then simulate all possible moves for the next piece, considering its different orientations and placements. This comprehensive analysis allows us to evaluate the potential outcomes of each move.

By combining the scores obtained from simulating both the current and next pieces, we determine the optimal value for the current move to be played. This involves assessing the potential consequences of each move in light of the subsequent piece and selecting the move that maximizes overall gameplay performance

Intuition

Considering the algorithm has no knowledge of the next piece and it evaluates three possible moves with fitness scores of [10, 20, 30]. Naturally, it selects the move with the highest score (the 30), assuming it to be optimal. However, there's a potential drawback: What if this seemingly optimal move results in a suboptimal position (bad position) for the next piece?

Now, let's consider the same scenario, but with the algorithm equipped with knowledge of the next piece. With this foresight, the algorithm conducts calculations for each possible move of the current piece, factoring in the next piece's potential positions and their corresponding scores.

For instance:

- Current piece moves: [10, 20, 30]
- Next piece moves (based on moves of the current piece): [80, 20, 10]

This insightful analysis reveals a crucial insight: While the move with a score of 30 may appear optimal in isolation, it leads to a suboptimal position for the next piece, resulting in an overall score of only 40 (30 + 10). Conversely, selecting the move with a score of 10, though initially deemed less favorable, places the subsequent piece in a strategically advantageous position, resulting in an overall score of 90 (10 + 80).

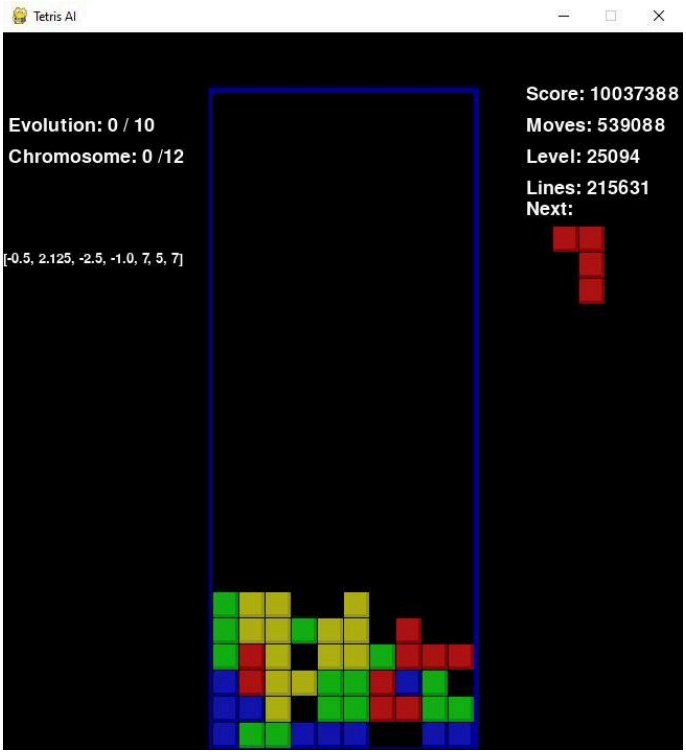
Does it really make a difference ?

Despite being slower - as it make massive calculation for each move- it has increased the score by about 3233% and number of moves played by 3110% and lines removed by 3116%

I've gathered the following statistics by executing both algorithms using the best chromosomes obtained during training sessions until a loss occurs. Here are the results:

Algorithm Without considering next Piece	Algorithm considering next piece
• Does it lose: Yes • Score Gained: 309,422 • Moves Played: 16802 • Lines Cleared: 6701	• Does it lose: No • Score Gained: 10,0373388 • Moves Played: 539088 • Lines Cleared: 215631 *Does not lose means that the algorithm has ran without loosing for about +24 hr and that it was closed intentionally .i.e it would run forever

Here is a screenshot during the game, demonstrating the scores I have mentioned



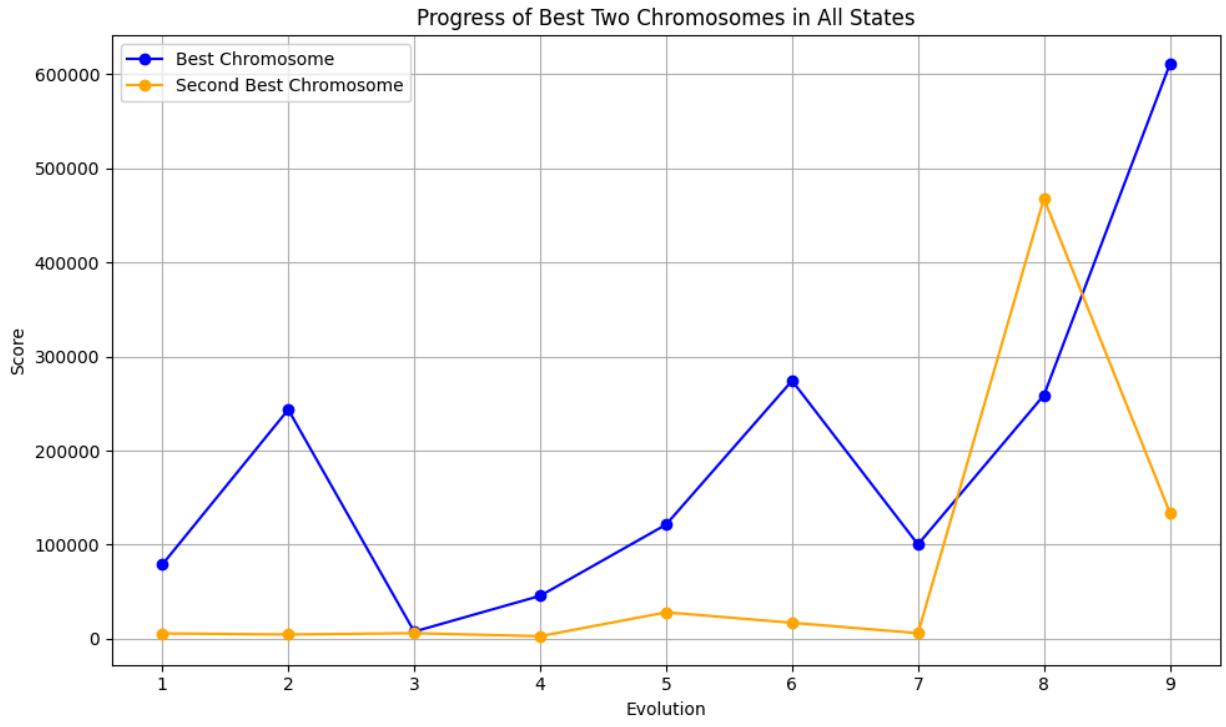
Training Insights

Seed

We have chosen a random seed of 42 during our algorithm

Best two chromosomes

- First Chromosome :[-1.42 , 7 ,-8 , -2.41 , 8,6,8]
- Second Chromosome:[-1, 4,-5.5, -2, 8, 6, 8]



Final Run Score

Winning Condition

The winning criteria for our algorithm were either achieving a score above 100,000 points or completing the game within 2000 moves, without encountering a loss condition

Importantly, our algorithm possesses the capability to run forever without encountering a loss condition

Final Score

- Score Gained: 10,0373388
- Moves Played: 539088
- Lines Cleared: 215631