Survival of the Tetris

CS 4701

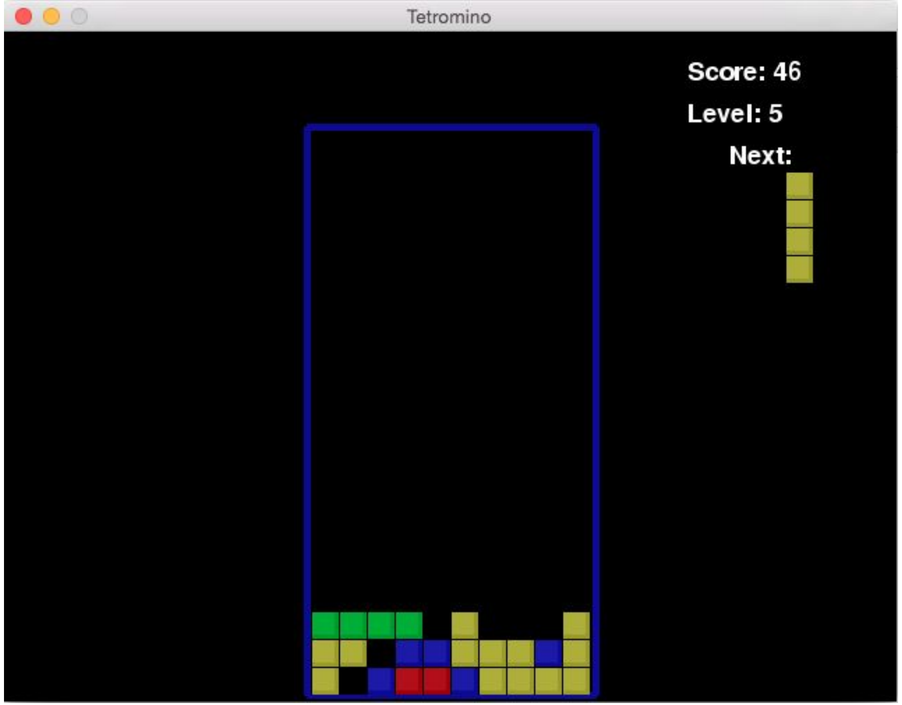
Rohit Biswas (rb625)
Tao Marvin Liu (tl474)

Introduction

- Learned about Genetic Algorithms in class, saw its application on Tetris
- Goal is to create a genetic algorithm that will identify the best move to make for each turn in a game of Tetris

Framework - Game

- Pygame
- Started with open source Tetris Implementation
- Removed all user input and timers since they are not needed
- Implemented functions *getAllMoves()* and *evaluateBoard()*
- AI runs *evaluateBoard()* on all the boards returned by *getAllMoves()* and chooses the best one

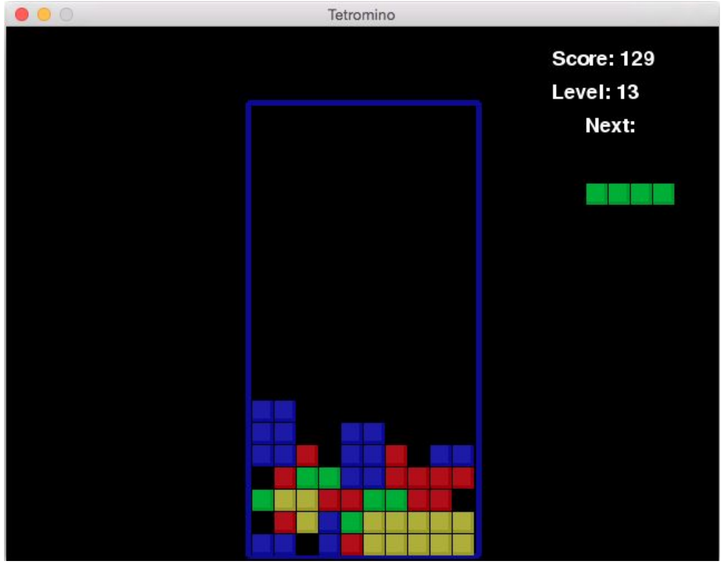


Framework - Trainer

- Extension of game implementation described in last slide
- Remove all GUI to improve performance
- Used DEAP evolutionary algorithm framework to learn parameters

Heuristics

- Total Height
  - Sum of height of each column
  - Lower height means we are further away from losing
- Lines Completed
  - Objective we are trying to maximize
- Holes
  - Empty space with piece above it in same column
  - Want to minimize these since they are hard to clear
- Height Variance
  - Total difference in height between adjacent columns
  - Uneven heights make it harder to form a complete line



"I" Block needs to be placed
Initial Total Height = 7+7+5+4+6+6+5+4+5+5 = 54
Initial Holes = 4
Initial Height Variance = 0+2+1+2+0+1+1+1+0 = 8
Next Potential Boards Calculated with new values for Heuristics with "I" Block

Genetic Representation of Solution

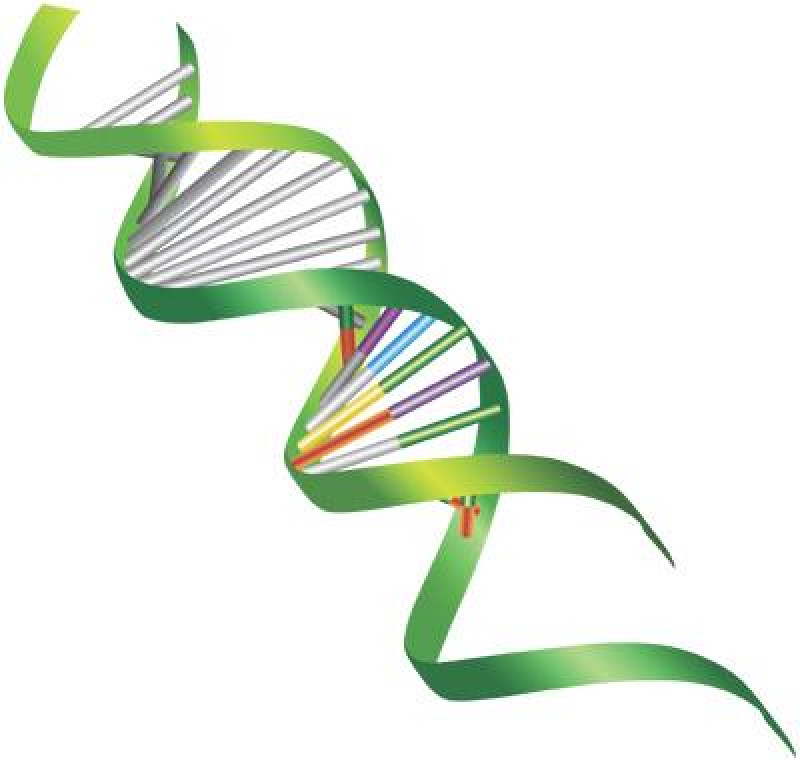
- Evaluate each move by linear combination of the 4 heuristics
- Solution represented by vector $w = \langle a, b, c, d \rangle$
- Score = a × Total Height + b × Lines Completed + c × Holes + d × Height Variance
- Looking for a vector *w* which performs the best

Evaluating Fitness

- Play 10 games for each weight vector and sum up the total number of lines cleared
  - Playing multiple games reduces variance and effects of high bias
- Limit number of tetrominoes per game to prevent training from taking an extremely long time

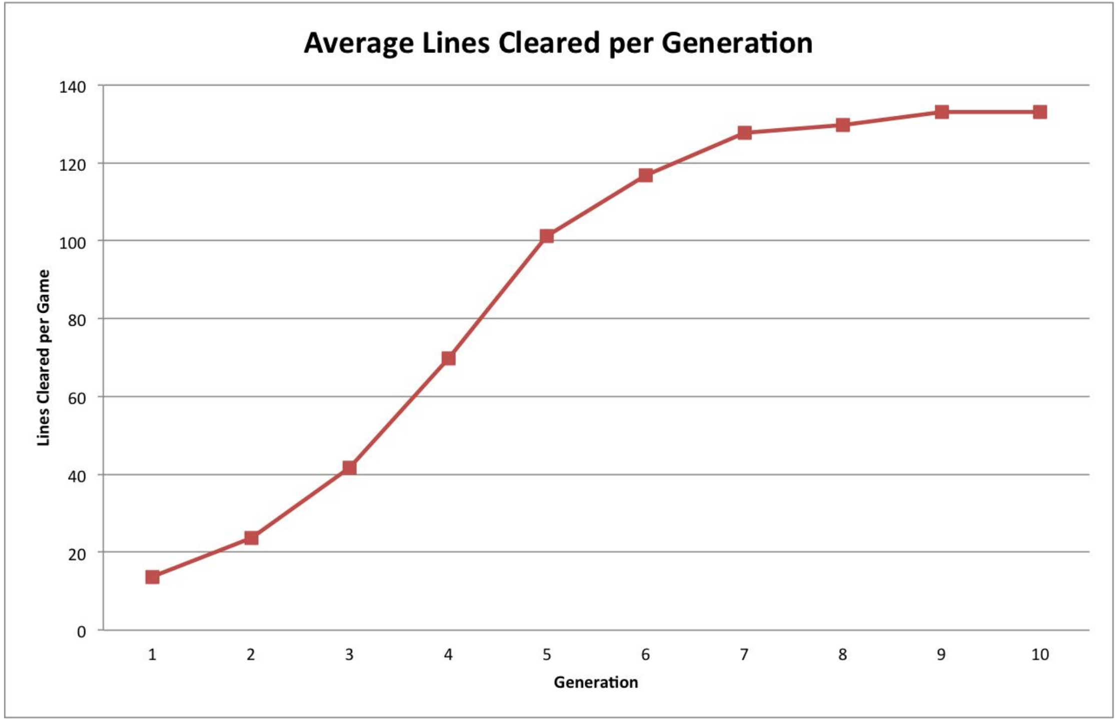
Genetic Algorithm

- Randomly selected initial population of 100
- For each generation:
  - Evaluate fitness of entire population
  - Tournament selection to find parents
  - One point crossover, gaussian mutation, and reproduction to create 100 children
  - Tournament selection on 200 (parent and children) vectors to pick 100 to survive to next generation
- Run for 30 generations (not enough time or computing power for more)



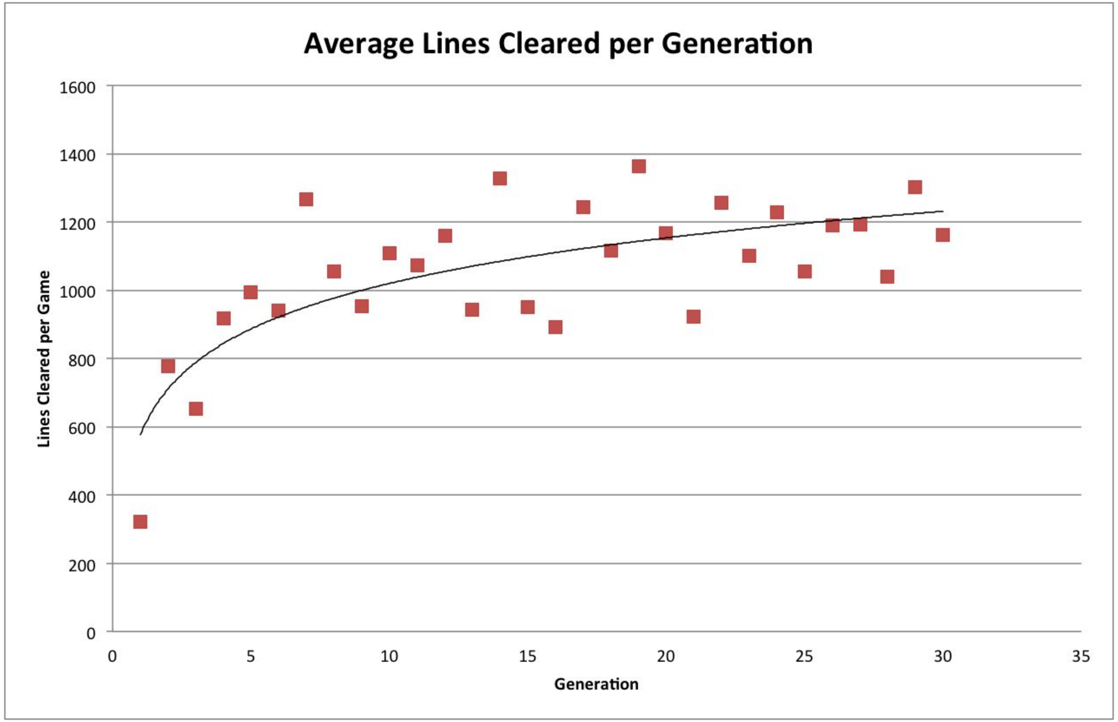
Initial Results

- 10 generations only (due to bad performance)
- Average lines cleared reaches a maximum because we limited each game to 400 tetrominoes



Final Results

- Population size 100
- 30 generations
- Increase limit to 5000 tetrominoes per game



Future Improvements

- Increasing search space to deal with overhang
- Looking ahead to the next move
- Tuning genetic algorithm hyper-parameters
- Improving performance via GPU processing or clusters