Build a Game Playing Agent

Opening Book using Genetic Algorithm

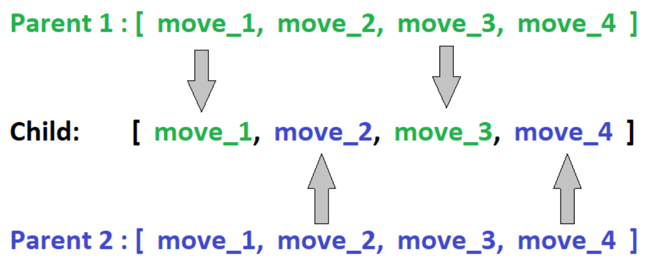
1. **Implement advanced search technique.**
   I decided to develop an opening book using a genetic algorithm. The custom agent was also augmented to use this opening book.

2. **Describe your process for collecting statistics to build your opening book, and the rollout procedure.**
   The opening book I constructed attempted to find the best moves for every possible game state from an empty board. Notation clarification: from here onwards let player_0 represent the player that starts the game with an empty board.
   A genetic (evolutionary) algorithm used was:
   i) For each starting board position, construct a large pool of *genomes*. A genome is a sequence of actions [move_1, move_2, …, move_n], where each move_i corresponds to one ply for player_0. Thus, a genome of length n corresponds to n rounds in the Isolation game.
      - Note: Each genome is initially generated as a sequence of random <u>legal</u> actions on an empty starting board with no opponent. Also, move_1 in each genome is the starting board position.

   ii) for each starting position:
       for generation in range(number_of_generations):
         for genome_i in genome_pool:
      - Evaluate the fitness of genome_i by simulation a game against an opponent (player_1) that chooses a random initial move and then follows the minimax algorithm with search depth of 3 thereafter.
      - In the simulated games, player_0 chooses moves sequentially from the elements of genome_i. Once the moves from genome_i have been exhausted, player_0 follows the minimax algorithm with search depth of 3 thereafter.
      - If the next move suggested by genome_i is invalid for the current board state, player_0 substitutes it for the move suggested by the minimax algorithm.
      - The fitness of genome_i is tuple: (player_0's utility at final game state, number of turns to complete the game)

      Sort the genomes by their fitness (prioritizing winning in fewest turns) and discard the worst k_percentile (where k is based on learning_rate)
      Generate k_percentile new genomes by splicing:

- The splicing strategy chooses two parents based on a weighted random sample of genome_pool, where the weights give preference to genomes with higher fitness.
- The spliced child genome contains moves from the two parent genomes in alternating order (as illustrated below).


Parent 1 : [move_1, move_2, move_3, move_4]
Child: [move_1, move_2, move_3, move_4]
Parent 2 : [move_1, move_2, move_3, move_4]

- Check if the new child genome is a legal sequence of moves (starting on an empty board, with no opponent). If the new genome is not legal, the splicing process is repeated with a different sample of parents.

The new spliced genomes are added to the genome_pool.

Lastly, a small percentage of the genomes from the genome_pool are mutated by replacing a randomly (uniform) chosen move with another legal move. Note: the mutation of a move may affect the legality of subsequent moves in the genome's sequence. So, more than one (random) mutation per genome may be necessary to preserve the genome's legality.

I decided to run this algorithm using 2 sets of hyperparameters:

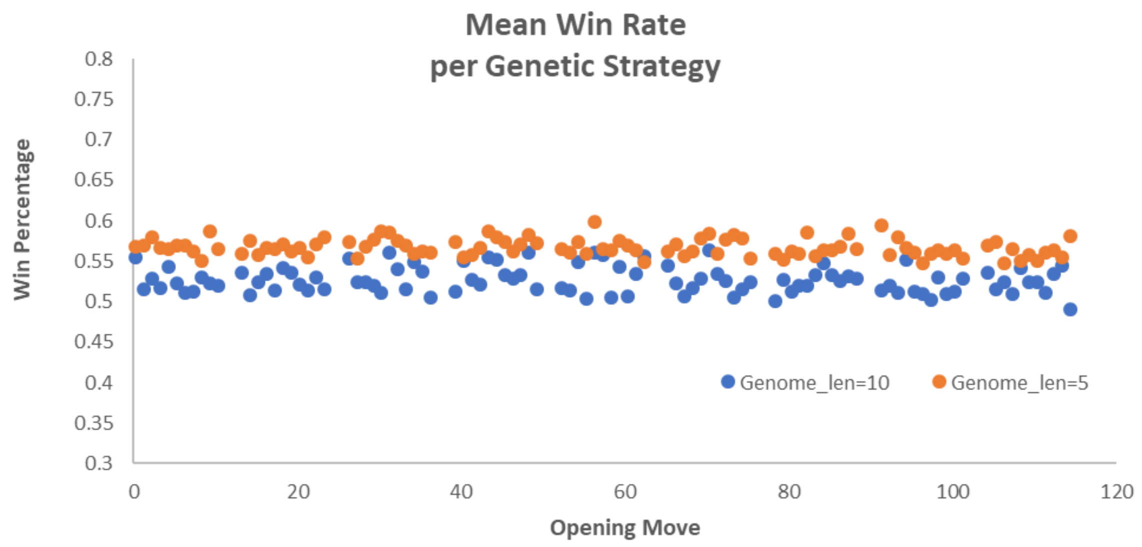
Strategy 1:	**Strategy 2**:
Genome length = **5**	Genome length = **10**
Genome Pool Size = 2000	Genome Pool Size = 2000
Number of Generations = 50	Number of Generations = 50
Learning Rate = 0.25	Learning Rate = 0.25
Mutation Rate = 0.05	Mutation Rate = 0.05

The heuristic used with minimax algorithm for all experimentation is the (#my_moves - #opponent_moves) heuristic.

3. Experimental Results

For each starting position, each genome from the final genome_pool for the two strategies above was tested against 100 opponents. The 100 opponents (player_1) each chose a random initial move and followed the minimax algorithm with search depth of 3 thereafter. The starting player (player_0) followed the actions in the genome until exhausted; and used the minimax algorithm with search depth of 3 thereafter.

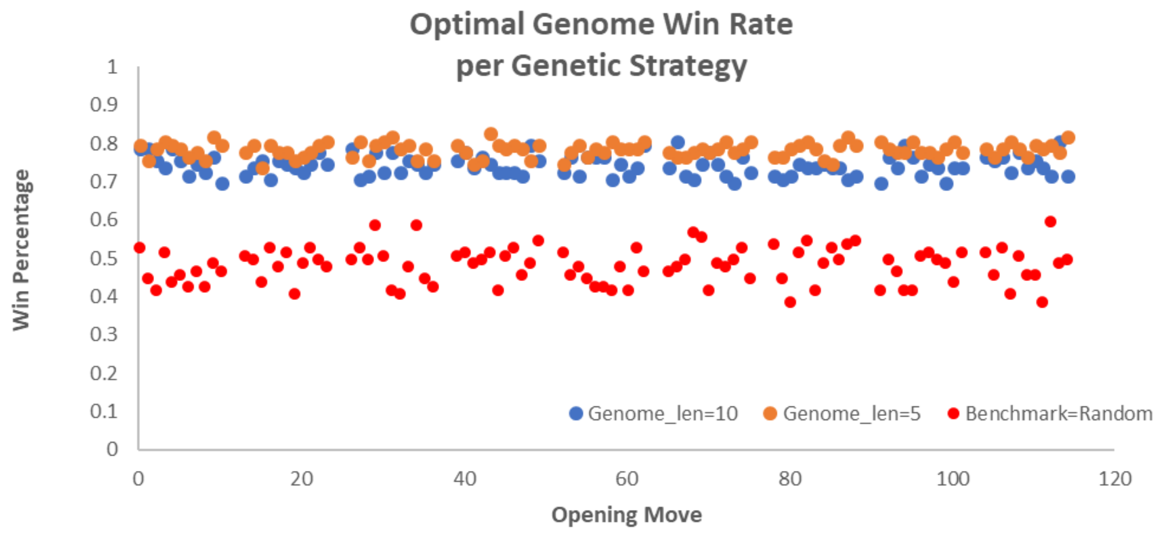
The chart below shows the average win rate across all genomes for each starting position.



We can see that Strategy 1 consistently outperforms Strategy 2. I suspect longer genomes require a greater number of generations to improve performance.

The chart below shows the win rates for the best performing genome for each starting position.

For the benchmark, we set player_0 to randomly select from legal actions for the first 5 plies; and use the minimax algorithm with search depth of 3 thereafter. The benchmark for each starting position was also tested against 100 opponents (as above).



The average win rate (across all starting positions) for the benchmark is 49.38%. The average win rate for the best genomes from Strategy 1 is 78.75%. The chart above also shows that Strategy 1 consistently out performs the random benchmark against the 100 opponents (as defined previously).

So, the 5-ply opening book I created is made up of these best performing genomes from the genetic Strategy 1, for each starting position.

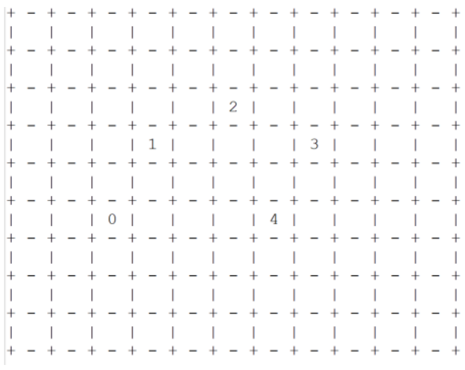
4. What opening moves does your book suggest? What is player 2's best reply?

According to my experiments, some examples of opening strategies that were able to achieve close to 80% win-rates are:

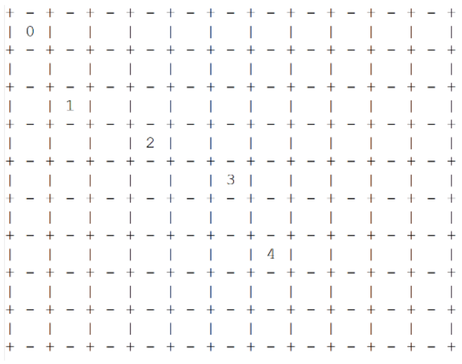
- Opening Move: 47 -> Action.NNE -> Action.ENE -> Action.ESE -> Action.SSW (win-rate 81%)
- Opening Move: 114 -> Action.SSE -> Action.ESE -> Action.ESE -> Action.SSE (win-rate 80%)
- Opening Move: 43 -> Action.NNW -> Action.NNE -> Action.ESE -> Action.WSW (win-rate 80%)
- Opening Move: 56 -> Action.NNE -> Action.WSW -> Action.WSW -> Action.SSW (win-rate 79%)

Below are the board representations of the moves above

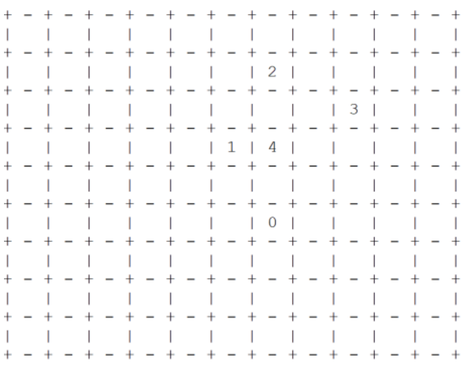
Opening Move: 47



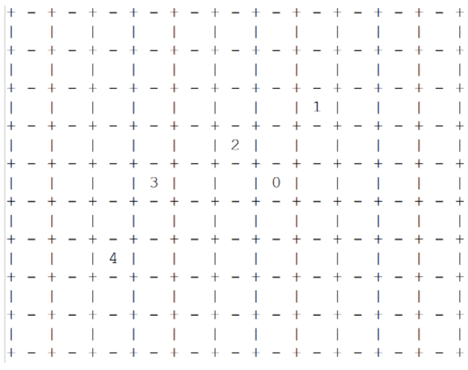
Opening Move: 114



Opening Move: 43



Opening Move: 56



As expected, one successful strategy for replies by player_1 is to visit one (or more) of the cells in player_0's opening move sequence before player_0. In my current implementation, this forces player_0 to resort to the mini-max algorithm (same as player_1), and thus drives player_0's win-rate closer to 50%.

Below are some other move sequences that defeat the opening moves discussed above:

- Player 1 = Initial Move: 47
  Player 2 = Move Sequence: 60 -> NNE -> ENE -> ENE -> SSW
- Player 1 = Initial Move: 114
  Player 2 = Move Sequence: 3 -> NNE -> NNE -> NNE -> WSW
- Player 1 = Initial Move: 43
  Player 2 = Move Sequence: 78 -> SSW -> SSE -> WSE -> NNE
- Player 1 = Initial Move: 56
  Player 2 = Move Sequence: 66 -> NNE -> WSW -> SSE -> WSW

I suspect more robust performance can be achieved from this opening book strategy by increasing the number of generations in the genetic algorithm (keeping the other model hyperparameters same).