Playing Tetris with Genetic Algorithms

Tetris

Tetris is the classic falling blocks video game invented by Russian programmer Alexey Pajitnov in 1984.

This immensely popular and deceptively simple game is well suited for machine learning applications.

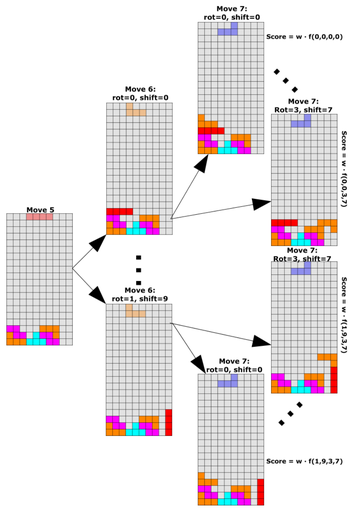
Optimization of Tetris

For my Tetris optimization problem, a candidate player has a **weight vector** and a state of Tetris has a **feature vector**. The dot product of these vectors ($w \cdot f$) produces a **score**.

In my Tetris game, a **move** is picking one of the four rotations of the **tetromino** (Tetris piece) and which column to drop it from.

When a player clears lines, the player earns:

- **2 points** for **1 line**
- **5 points** for **2 lines**
- **15 points** for **3 lines**
- **60 points** for **4 lines** (known as a **tetris**)



When picking a move, the player knows the next two tetrominos and generates every possible future state over two turns. The player picks the state that produces the best score.

Goals

For my purposes, a good player should accomplish two things:

- Make as many moves as possible without losing
- Efficiently score points by making more tetrises.

The **efficiency** of a player is the **average score per move** divided by 6 (the best average possible).

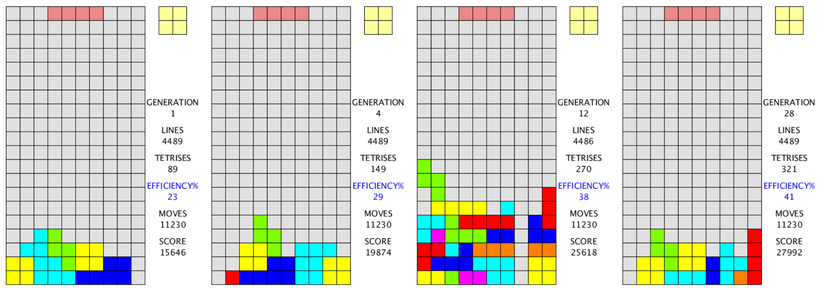
Genetic Algorithms

Optimization Problems involve finding the optimal input to maximize or minimize a function.

Genetic Algorithms are an approach to optimization by simulating natural selection. A **population** of **candidate** inputs is maintained. The better a candidate performs, the more likely it will contribute to the next **generation**, resulting in better candidates each generation.

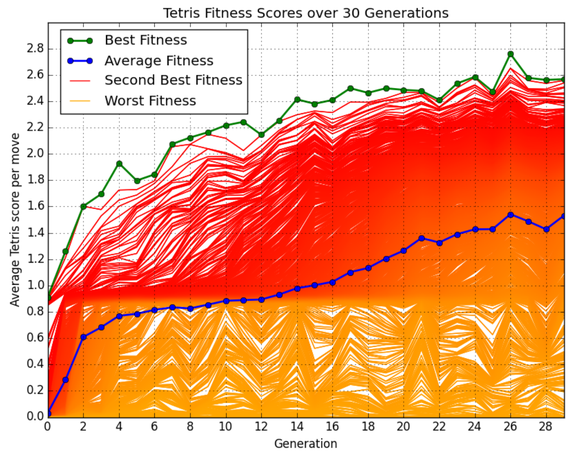
Results

The genetic algorithm was run for **30 generations**, with a population of **1000 players**. The fitness function is the **average score over 10,000 moves**.



Screenshot of the best players of generations 1, 4, 12, and 28 playing the same game. The higher generation players are achieving higher efficiency scores.

The best resulting Tetris players can play games for hundreds of thousands of moves and achieve an average scores above 2.4 per move (a score efficiency of 40%).



Genetic Operators

Gen 0 is initialized randomly. Future generations are generated using three genetic operators:

- **Selection** – some candidates are selected and added unchanged
- **Mutation** – some candidates are changed slightly before being added
- **Crossover** – some pairs of candidates are combined into new candidates

Feature Selection

Forward search was used to pick the best subset (F) of features:

- Start with F = $\emptyset$
- Repeat:
  - For each feature i not in F, evaluate $F_i = F \cup i$
  - Determine F_i with best score and set $F := F_i$

Each column below shows one iteration of forward search, where the best feature is added to the final feature set. In the iterations not shown, all values are 3.628.

The top half contains the final feature set. The bottom half contains features that failed to improve the score.

WeightedBlocks	0.359										
ConnectedHoles	0.028	1.129									
LinesCleared	0.002	0.359	1.746								
Roughness	0.005	0.359	1.323	2.458							
Tetrises	0.001	0.359	1.129	1.722	3.009						
PitHolePercent	0.012	0.359	1.633	1.863	2.458	3.167					
ClearableLine	0.001	0.527	1.714	1.621	2.458	3.030	3.253				
DeepestWell	0.002	0.369	1.412	1.876	2.211	3.009	3.167	3.536			
Blocks	0.002	0.359	1.507	1.768	2.458	3.009	3.063	3.138	3.558		
ColHoles	0.028	0.666	1.129	1.907	2.475	3.009	3.219	3.27	3.540	3.628	

Concavity	0.001	0.359	1.129	1.746	2.458	3.009	3.207	3.037	3.224	3.558	3.628
ColumnVariance	0.027	0.416	1.163	1.772	1.902	3.009	3.202	3.432	3.536	3.32	3.483	3.628
HeightDiff	0.012	0.481	1.226	1.746	2.286	3.009	3.009	3.087	3.536	3.558	3.483	3.628
HeightSum	0.183	0.939	0.951	1.746	2.38	3.009	3.140	3.487	3.552	3.552	3.628	3.628
Holes	0.042	0.933	1.129	1.746	2.458	3.009	3.167	3.263	3.487	3.558	3.628	3.628
LowestClearableLine	0.001	0.579	1.273	1.746	2.352	3.009	2.994	3.360	3.518	3.391	3.628	3.628
MaxHeight	0.102	0.359	1.129	1.474	1.954	3.009	3.188	2.990	3.432	3.558	3.628	3.628
One	0.001	0.359	1.129	1.746	2.458	3.009	3.167	3.253	3.536	3.558	3.628	3.628
Pit	0.002	0.359	1.293	1.899	2.146	3.058	3.058	3.145	3.187	3.558	3.628	3.628
Score	0.002	0.359	1.153	0.959	2.897	2.897	2.853	3.487	3.536	3.503	2.789	3.628
SideStackQuality	0.003	0.359	1.129	1.898	2.458	3.009	3.243	3.466	3.432	3.558	3.628	3.628
WeightedBlocksLog	0.297	0.297	1.218	2.001	2.015	3.009	3.167	2.849	3.067	3.382	3.628	3.628
Wells	0.023	0.684	1.129	1.791	1.935	2.322	2.815	3.466	2.683	3.506	3.628	3.628

My poster was accompanied by a live demo.
I have uploaded a short sample video of the final demo.

https://vimeo.com/148293176