Term Project: Let's Play Tetris!

Group Number: 8
Team Members: Jeyavithushan Jeyaloganathan A0106876, Joanne Leong A0106624,
Tan Jun Zhong U0900042, Sing Keng Hua A0067387, Bathini Yatish A0091545

Agent Strategy

Strategy Overview

The objective for this project was to complete a utility-based agent to play the game of Tetris. The Tetris player we have implemented (titled the JL Tetris AI) uses a heuristic function that comprises a linear weighted sum of features and is based on based on El-Tetris [1]. Since the state space is very large, a heuristic function is used in lieu of determining a utility score for each possible board state. Given the next piece, the player evaluates and assigns a corresponding heuristic-score for each legal move available. The player plays the legal move that achieves the highest total heuristic score. Several features factor into the overall desirability of that move; the features and weights that were used are outlined below.

Characteristic Features

The heuristic function of the implemented Tetris player evaluates the legal moves available by tracking several characteristic features of its resulting playing field. These features are the number of rows cleared, the landing height of a piece, the number of holes, the number of row and column transitions, whether the move is a losing move, and the size of wells.

Number of Rows Cleared

This feature counts the number of rows that would be cleared by making a move. The greater the number of rows the move clears, the better it is.

Landing Height of a Piece

This feature is calculated by determining the height at which the piece will be placed. Note that the height is taken from the middle of the placed piece, and not the top edge. A move that results in a higher landing height is worse than a move that has a lower landing height.

Number of Holes

A hole is defined as an empty slot that exists beneath a filled slot in a column. This feature counts the total number of holes that exist in the resulting playing field. The fewer the number of holes in the resulting playing field, the better the move.

The number of Row and Column Transitions

A transition is a characteristic defined for either a column or row in the playing field. A transition is counted every time there is a change from a filled block to an unfilled block, and vice versa. The number of transitions, the less irregularities exist in the playing field, and the better the move.

Losing the Game

Losing the game is the least desirable outcome for the Tetris player. A move that costs the Tetris player the game should be avoided completely and is thus assigned a maximum penalty.

Size of Wells

A well is defined as an empty slot between two filled slots. A well can be several units wide and several units deep. For this feature, the total number of slots that comprise a well is counted. A move that creates a new well or increases the size of an existing well is unfavorable.

Feature Weights

The weighting of each feature in the linear evaluation function is the most important part of the Tetris AI. For this specific AI, the weights were determined using Particle Swarm Optimization in combination with stochastic hill climbing to prevent entrapment in a local maxima. Note that JSwarm-PSO (an open source swarm optimization package) was used for optimization.

The idea is to move a candidate solution composed of particles into a hyperspace of parameters to improve it with reference to some evaluation function. The swarm (these particles) traverse the search space with different initial velocities and positions. The movements of the particles are guided by the "best" known position of a particle in the entire swarm. A larger number of iterations lead to a more significant optimization [3][4][6].

The weights of each feature were optimized using a swarm of 20 particles with the weights representing the position and the velocities of each particle represented by the evaluation delta between two iterations. Of the 20 particles, the 5 performing the worst were replaced with randomly generated weights with random velocities in order to prevent entrapment in a local maxima.

The evaluation function for each particle was the total number of rows cleared. The positions and velocities were randomly generated for the first iteration. At all subsequent iterations, the algorithm finds the best position (weights) and updates the global best position. The velocities of the other particles are then adjusted based on the global best position and velocity. Eventually the swarm should settle on an approximate global optimal position. The weights used in this Tetris AI was the position of the global best position after 5 iterations of PSO.

Table 1. Weightings used for each Tetris AI.

Features	El-Tetris Weights[1]	Pierre Dellacherie Weights [2]	JL Weights
Rows Cleared	3.4181268	4	3.2139831
Landing Height	-4.5001588	-1	-4.7329915
Row Transitions	-3.2178882	-1	-3.0148912
Column Transitions	-9.3486953	-1	-8.5341347
Size of Wells	-3.3855972	-1	-3.2390876
Total Holes	-7.8992654	-1	-7.1114569

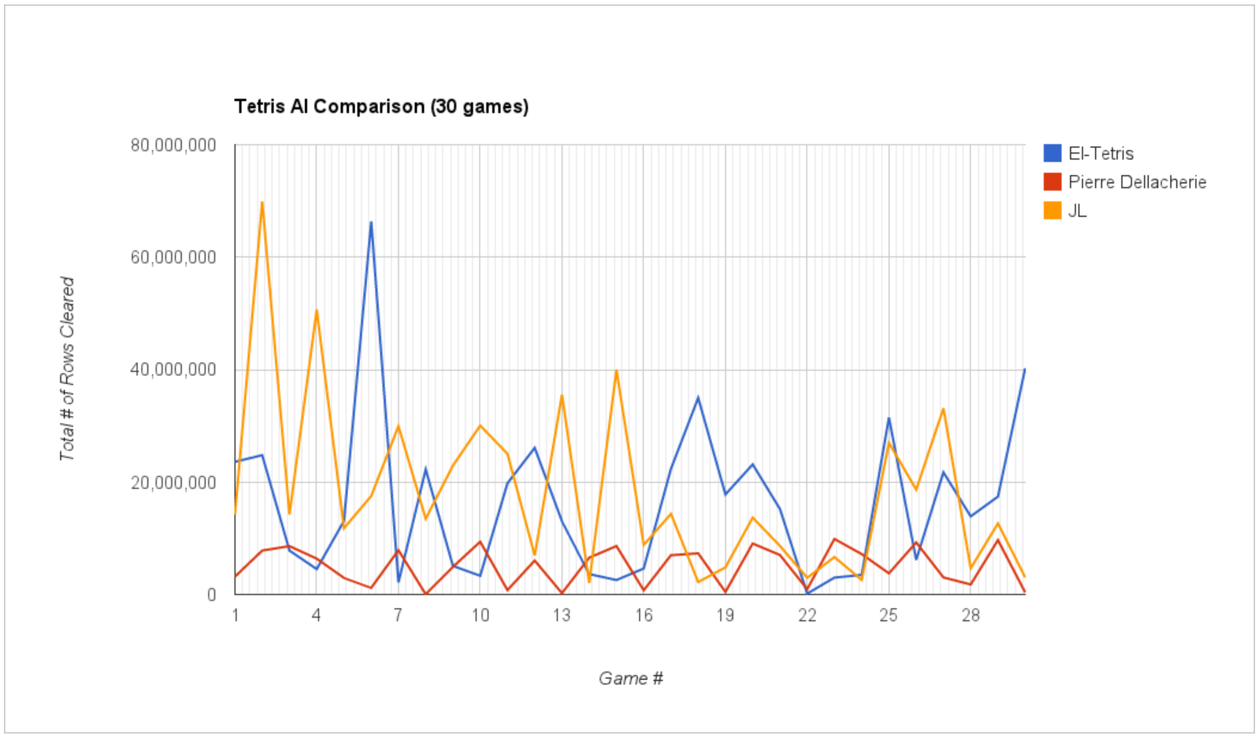
Experimental Results

The results of the Tetris AI were recorded for 30 games and analyzed against the El-Tetris and PD AIs. The following are the results.

Table 2. Comparison between El-Tetris, Pierre Dellacherie and JL on number of rows cleared.

	El-Tetris	Pierre Dellacherie	JL
Mean (Rows Cleared)	16,454,879	4,341,276	18,268,412
Standard Deviation (Rows Cleared)	13,208,708	3,447,601	15,903,642

Figure 1. Graph illustrating the number of rows cleared by the El-Tetris, Pierre Dellacherie and JL Tetris



The experimental results indicate that JL is the superior Tetris player. While its performance does not always exceed the performance of the other two implementations (based on a 30 game sample size) it does on average clear more rows.

Discussion and Analysis

Over 30 games based on the results, the JL Tetris AI performs the best. The PD algorithm performs the worst. This is expected as the weights are not optimized. It is important to note that this is not a completely fair comparison because the environment is not strictly controlled. In order to analyze the three AI's objectively, the tetromino sequence for each game should be consistent. This could be done by adjusting the random seed for the tetromino generator to be the same.

Moving forward, there are a few steps one can take to improve the JL AI. The first is to add more features to the algorithm such as the max height of the pile after the piece is played or the average pile height. The addition of more features should improve the performance of the AI by providing it more percepts. Secondly, one can use an exponential heuristic function instead of a linear one. Intuitively this makes sense - as the Tetris AI nears a losing state, the importance of certain features such as the number of rows cleared and the pile height increases dramatically. Finally one may improve the algorithm by adjusting the evaluation function for the position of the swarm particles during PSO to be more comprehensive such as penalizing on total number of holes generated or on total number of lines cleared vs. total number of tetromino encountered.

References

[1] El-Ashi, I.(2011). "El-Tetris", URL http://ielashi.com/el-tetris-an-improvementon-pierre-dellacheries-algorithm/
[2] Fahey, C. (2003). "Tetris AI", URL http://colinfahey.com/tetris/tetris.html
[3] Carr, D. (2005)." Applying reinforcement learning to Tetris; Technical report". Rhodes University, 15 pages.
[4] Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. IVICNN.1995.488968.
[5] Nick Parlante, (2001) Stanford Tetris Project http://cslibrary.stanford.edu/112/
[6] Cingolani, P. (2006) "JSwarm-PSO" http://jswarm-pso.sourceforge.net/
[7] InriaForge (2009). "Mdp Tetris documentation" http://mdptetris.gforge.inria.fr/