Growing with Big Data, A Tetris Player: Project Report by Group 22

Loh Han Tong, Victor
A0135808B

Ryan Tan Wen Jun
AxxxxxxxZ

Tan Yu Wei
AxxxxxxxZ

Li Jiaxin Cassandra
AxxxxxxxZ

April 7, 2018

1 Introduction

The purpose of this project is to create a utility based agent to maximise the number of rows removed in a game of Tetris. This tetris playing agent uses a heuristic function to estimate the utility of each state.

In this report, we discuss how this agent was designed and the features used to evaluate the utility of the board. We will also look at how we have implemented and used genetic algorithm to train a tetris agent that could play Tetris decently well, averaging about 19,700,000 lines cleared.

2 Strategy

The agent's heuristic function sums the linear weights $w(k)$ of features $\varphi_k(s)$ (As stated in subsection 2.1) for a given state of the board, s, where n is the number of features as shown below:

$$\hat{V}(s) = \sum_{k=0}^{n} w(k)\varphi_k(s)$$

Where at every turn, the agent evaluates, among all possible moves using the heuristic function, the move that gives the best utility.

2.1 Features Selected

This is the list of 11 features that we have selected. They allow us to evaluate each state s based on certain characteristics of the board.

- **NUM_ROWS_REMOVED** – Number of rows removed
- **MAX_HEIGHT**
- **TOTAL_HEIGHT**
- **TOTAL_DIFF_HEIGHT** – Sum of all difference in height of all columns
- **LANDING_HEIGHT** – Height of where the next piece lands
- **NUM_HOLES** – Number of empty cells with at least one filled cell above
- **COL_TRANSITION** – Number of filled cells adjacent to empty cells, summed over all columns
- **ROW_TRANSITION** – Same as the above, but applied to rows
- **COVERED_GAPS** – Number of empty cells with a filled cell anywhere above them
- **TOTAL_WELL_DEPTH** – Sum of the depth of all wells
- **HAS_LOST** – Gives a penalty of -10000 if move result in loss, else give 100

2.2 Genetic Algorithm

For our implementation of the genetic algorithm, Each chromosome has a weight vector where each gene (weight value) corresponds to one of the 11 features stated in subsection 2.1, and a fitness score.

The fitness score of each chromosome is defined as the mean score of playing 50 games using that individual's chromosome weight.

This is our implementation of the genetic algorithm:

1. Start out with 1000 individuals with random weights. Initially calculate their fitness score.
2. Select 40% of population via Stochastic Universal Sampling to be potential parents
3. Generate 40% of population as offspring by the process below:
   (a) Randomly select 2 parents from the pool generated above
   (b) Crossover with 80% chance, by taking weighted average of genes
   (c) Mutate these 2 offsprings with 8% chance by adding 1/10 times the random gaussian value.
   (d) calculate fitness score for the 2 offsprings
   (e) Add to offspring pool
4. Cull bottom 40% and replace with offsprings in offspring pool
5. Repeat steps 2 to 4 for each generation, till convergence

Convergence is determined by the score of the best individual in the population. If this score has not improved for 50 generations, we terminate the algortihm.

2.3 Parallelisation and Speedup

Each generation of the algorithm required running games to evaluate fitness. This meant that as the weights progressively got better, each generation started taking a longer time to evaulate.

We decided to parallelise the games by running each game on its own thread. Playing 100 games, with a set of weights that give around ¡SCORE¿ lines cleared, the speedup from sequential to parallelised is ¡TIMES¿.

Another way that we have tried speeding up the learning algorithm was to simply reduce the number of rows that the board has. Our team ran 2 instances of the learning algorithm with a smaller board of 9 rows and 13 rows. The instance running with 9 rows, even at later generations, took an average of 30 minutes per generation, while the latter, took an average of 4 hours per generation.

3 Results

The following results are from the weights shown in Table 1. These weights were derived from the instance running GA on a board with 13 rows at generation 132.

Features	Weights
NUM_ROWS_REMOVED	-0.10994115458466136
MAX_HEIGHT	-0.1154697834187254
TOTAL_HEIGHT	-0.04390525258236673
TOTAL_DIFF_HEIGHT	0.017912908135268947
LANDING_HEIGHT	-0.3044476707923254
NUM_HOLES	-0.38617473506172584
COL_TRANSITION	-0.12518629866820255
ROW_TRANSITION	-0.22806177833393343
COVERED_GAPS	-0.7696058904564755
TOTAL_WELL_DEPTH	-0.19377750577164388
HAS_LOST	0.13672271498097804

Table 1: Respective Weights for Features

The result of running 600 games can be seen in Figure 1, while some common metrics of the 600 games can be seen on Table 2.

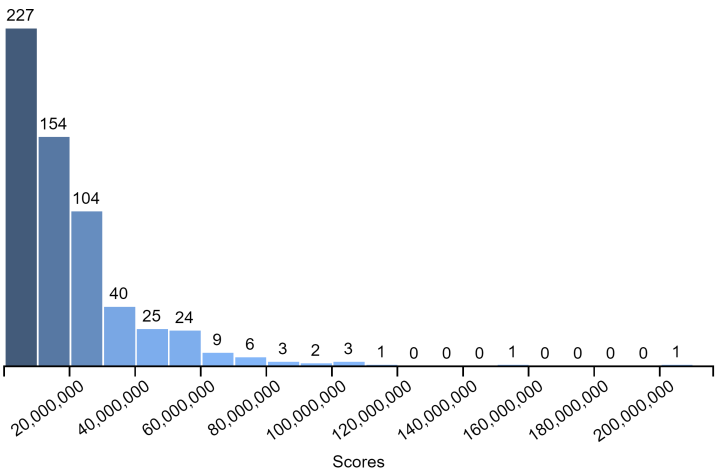

Figure 1: Results from 600 games

Features	Weights
Q1 (25th Percentile)	6,307,657.5
Median	13,655,622.0
Q3 (75th Percentile)	25,716,898.5
Mean	19,793,958.2
Max	216,319,742.0
Min	5125.0

Table 2: Common Metrics for the Scores

As we can see from the results above,