CS3243 Assignment: Learning to play Tetris with Big Data

Members: Luo Yuyang (A0147980U), Wang Si Qi (A0124863A), Edmund Mok (A0093960X), Li Heng (A0135739W), Yong Lin Han (A0139498J) **Group**: Tetris Agent 02

Introduction

The goal of this project is to develop a competent utility-based agent to play the Tetris game. The agent's aim is to maximise the number of rows cleared up to the termination of the game.

In Tetris, the blocks, or tetrominoes, fall from the top of the board. One row will be cleared if there is no any hole in it.

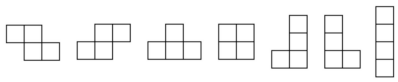

Figure 1: The 7 Tetris object shapes (tetrominoes)

Our agent uses a heuristic function to evaluate the utility of a state. The agent is trained by *Genetic Algorithm* to obtain the optimum weights for the heuristic function. This report explains how the agent is trained using *Genetic Algorithm*, and discusses the improvements we made to speed up the learning process for the agent to make use of big data.

Strategy of the agent

Given the current state of the game, for each tetromino, the agent simulates and evaluates all possible resulting states from all the possible moves: the orientations and positions of the tetromino. It then chooses the action that leads to the state with the highest evaluation value. The evaluation function used to evaluate the state of the game is given by the heuristic function:

$$V(i) = \sum_{k=1}^{n} w_k f_k(i)$$

where i is the state, w_k are the weights to be trained and f_k are the feature functions.

When the agent is evaluating a legal move and the resulting state from making that move results in a column reaching a height beyond a specified threshold value, the agent does another round of evaluation to determine the best move to make next. Given the current state of the game, the agent simulates all possible resulting states from all the possible moves. For each resulting state whose maximum height column is bigger than the threshold value, the agent does another round of simulation from that state onwards by looking at each of the tetriminos, and perform all possible placements and orientations to find the highest heuristic value of all the possible resulting states of the particular piece. Then the agent finds the average max heuristic value across all tetriminos to be the final evaluation value of the state. Finally the agent chooses the action that leads to the state with the highest evaluation value. In general, given a state i, the agent's strategy μ for playing the tetris game is defined as follows:

$$\mu(i,a) \triangleq \operatorname*{argmax}_{a \in A} H(f(i,a))$$

$$H(f(i,a)) = V(f(i,a)) \quad (1)$$

$$H(f(i,a)) = \frac{1}{m} \sum_{y=1}^{m} \max_{b \in B_y} V(f(f(i,a),b)) \quad (2)$$

1. JGAP Library, a Genetic Algorithms and Genetic Programming component provided as a Java framework. http://jgap.sourceforge.net

Where A is the set of all legal moves, $f(i,a)$ is the resulting new state by taking action a, m is the number of tetriminos, y is the current piece, B_y is the set of legal moves of the current piece y and $H(i)$ is the evaluation function. $H(f(i,a))$ is (1) if max height of $f(i,a)$ is less than or equal to the specified threshold and (2) if max height of $f(i,a)$ is more than the threshold.

Feature Functions

We use the following features for our heuristic function:

Number of holes: sum of all the empty cells that are below the top of each column

Complete lines: number of rows cleared

Height variation: sum of all the differences of adjacent column heights

Max height: the height of the tallest column

Pit depth: sum of all the pit depths. A pit is defined as the difference in height between a column and its two adjacent columns, with a minimum difference of 3

Mean height difference: the average of all height differences of each column with the mean height.

Lost Weight: whether the game is lost from making the move. A lost game is assigned a negative weight.

Our Learning Algorithm

In order to determine the weights to be given to each of the above features, we run the agent using a trainer which implements *Genetic Algorithm*. Each *Gene* represents one of the features and the seven genes are encapsulated as a *Chromosome*. A *Population* of 100 *Chromosomes* is set up and left to evolve. The fitness function evaluates the *Chromosome* and returns the number of lines cleared in one game as the fitness value. Standard crossing over and mutation rate from the JGAP package was used. After many evolutions the *Chromosome* with the highest fitness value is recorded to obtain the optimum weights for the features.

Handling Big Data

We use multi-threading to speed up the learning process so that the agent can make use of big data. Multi-threading is done in two processes: during the training of weights in the genetic algorithm, and during the evaluation of the best possible moves out of all legal moves. By running the learning algorithm on multiple threads, we are able to train with a larger population of chromosomes compared to a single-threaded approach since we can now do evaluation on multiple threads and we set each thread to have a population equal to the original population for single-thread (new total population = original population * number of threads). This allows the algorithm to have a wider pool of candidates to evaluate from and move to an optimum set of weights more easily, instead of converging to local optimas. For the player, we also apply a multi-threaded approach to evaluate the best possible move. Since the number of possible legal moves per step is quite small (at most 4 possible rotations * 10 columns = 40), this might not seem like much. However, when running the player over many moves, it tremendously speeds up the playing and the learning (since the fitness evaluation of genetic algorithm requires the playing). From our results, this approach has allowed us to achieve around 2x speedup as shown by the table below:

Rows cleared (intentionally stopped)	Single-Thread Player	Multi-thread Player (4 threads)
100,000	52 - 54s	24 - 28s
1,000,000	530 - 540s	230 - 240s

1. JGAP Library, a Genetic Algorithms and Genetic Programming component provided as a Java framework. http://jgap.sourceforge.net

Furthermore, we also apply multi-threading when we perform the second layer lookahead if the state has a column that exceeds some threshold value set. In the second layer we have to check 4 possible next pieces since we do not know which next piece we will get, so this results in a large state space. Using multi-threading, we are able to divide the work for multiple threads, allowing us to conquer the lookahead faster. This approach also yields around 2x speedup, and allows us to lower our threshold value to increase agent performance while still maintaining a decent speed for the agent.

Results and Analysis

The graph below shows performance of our agent running on the heuristic weights trained by *Genetic Algorithm*, with maximum height threshold of 8 and 10 respectively.

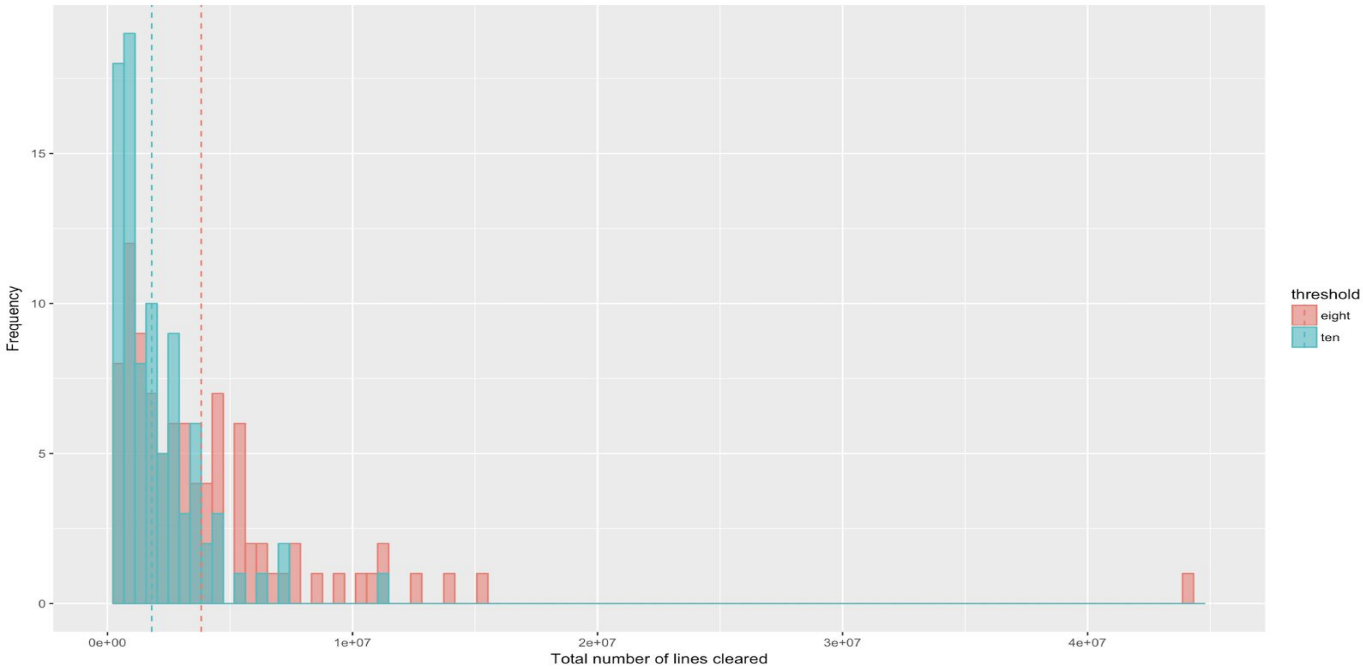

Figure 2: Total number of lines cleared vs frequency graph for 100 games played (dotted line shows the mean)

More detailed statistics are below:

Minimum	25th percentile	Median	Mean	75th percentile	Maximum
42,045	1,030,845	2,521,171	3,836.323	4,676,154	44,064,722

Table 2: Statistics of total number of lines cleared per game for 100 games played, setting the max height threshold to 8

Minimum	25th percentile	Median	Mean	75th percentile	Maximum
40,495	467,234	1,174,287	1,805,113	2,756,693	11,350,796

Table 3: Statistics of total number of lines cleared per game for 100 games played, setting the max height threshold to 10

Table 2 and Table 3 suggest that, as the maximum height threshold decreases, the performance of the agent improves. This can be further supported by the following graph.

1. JGAP Library, a Genetic Algorithms and Genetic Programming component provided as a Java framework. http://jgap.sourceforge.net

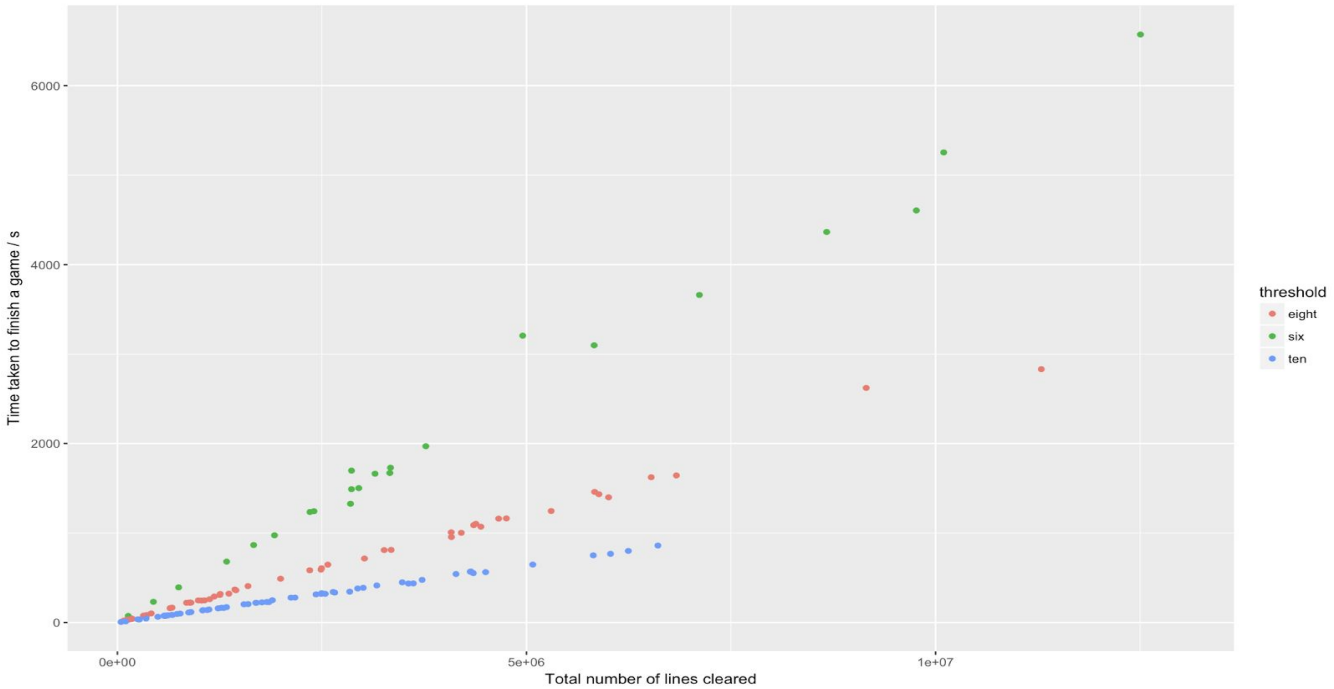

Figure 3: Total number of lines cleared vs Time taken to play the game graph for different max height threshold

From Figure 3, there is a clear trend between the threshold for lookahead and number of rows cleared by the agent - the lower the maximum height threshold, the more rows the agent is able to clear. However, it also takes a longer time for the agent to clear the same number of rows as those run with higher maximum height threshold. This is the performance-time trade-off.

Further Improvements in Future

The fitness value is defined to be the number of rows cleared by the candidate before the termination of the game. However, as the population evolves, candidates are able to clear more lines, resulting in longer fitness computation time and the learning process slows down. Furthermore, the learner could be trapped in a local optima during the process. To tackle this, we plan to apply the ***Least-Squares Policy Iteration*** (LSPI) algorithm to generate a decent set of weights first to avoid being trapped in the local optima and reduce the training time due to the fast convergence of LSPI. After that, we can apply genetic algorithm on the weights generated by LSPI as the initial weights to further improve it as LSPI does not guarantee the optimal policy. Two other possible methods to speed up the training process are reducing the size of the tetris for training and using the number of rows cleared within a certain number of iterations rather than till the termination of the game as the fitness value.

Conclusion

Our agent plays with a 2-layer look-ahead strategy that allows a threshold to be adjusted for time and performance trade-off. Our agent learns by the genetic algorithm. Multi-threading is used to speed up the learning process. We tried to implement our own genetic algorithm, but the result is not as good as JGAP Library[1]. Also, JGAP genetic algorithm optimises learning to allow multi-threading.

The learning process can be improved by LSPI as mentioned above. A network of computers may also be deployed to carry out the learning, instead of just multi-threading.

1. JGAP Library, a Genetic Algorithms and Genetic Programming component provided as a Java framework. http://jgap.sourceforge.net