CS3243 Project Report

Team 36
Tan Mingrui (A0158278N), Wang Debang (A0162479U), Yu Pei, Henry (A0156206L)
Zhang Hui (A0160463L)

Abstract

A linear heuristics model trained by Harmony Search algorithm was used to build a Tetris agent. To evaluate the learning algorithm, different possible algorithms were compared. Ways to accelerate the learning process were explored. Some of our key findings are shown in this paper.

1 Introduction

Tetris is a game where the player is given a two-dimensional grid and a random shape sequence are dropped top down. The player manipulates current piece before dropping it into the playing field. The goal of the player is to survive as long as possible by clearing rows that are completely filled. In this project, a grid of size 21 rows by 10 columns is used.

2 Heuristics

We adopted features 2 - 6 from Dellacherie's heuristics [1] and feature 7 from Boumaza [2].

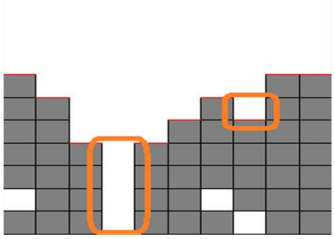

Fig 1. Example of wells

1. **Rows eliminated:** The number of rows eliminated. Rows need to be cleared to survive in the game. Intuitively, moves that clear more rows tend to be better.
2. **Landing Height:** The height where the piece is put (= the height of the column + (the height of the piece / 2)). Putting a piece at a lower height compared to higher heights will often lead to more homogenous block arrangements.
3. **Row Transitions:** The total number of row transitions. A row transition occurs when an empty cell is adjacent to a filled cell on the same row and vice versa.
4. **Column Transitions:** The total number of column transitions. A column transition occurs when an empty cell is adjacent to a filled cell on the same column and vice versa. Row and Column Transitions encourage holes to be grouped together to be cleared in fewer steps with higher probability.
5. **Number of Holes:** A hole is an empty cell that has at least one filled cell above it in the same column. In order to maximize the number of rows cleared and to minimize the total height, one key strategy is to avoid creating holes.
6. **Well Sums:** Sum of wells. A well is a succession of empty cells such that their left cells and right cells are both filled (The two orange circles in Fig.1).Wells are undesirable because the surface would be bumpier which may also restrict the shape of piece to fill the well.
7. **Row with Holes:** Number of rows that has a hole in it. It measures the number of rows that could not be cleared within a move.

Learning Algorithm:

Harmony Search

begin

Objective function $f(\mathbf{x})$, $\mathbf{x}=(x_1,x_2,\ldots,x_d)^T$

Generate initial harmonics (real number arrays)

Define pitch adjusting rate (r_{pa}), pitch limits and bandwidth

Define harmony memory accepting rate (r_{accept})

while *(t<Max number of iterations)*

Generate new harmonics by accepting best harmonics

Adjust pitch to get new harmonics (solutions)

if *(rand>r_{accept}), choose an existing harmonic randomly*

else if *(rand>r_{pa}), adjust the pitch randomly within limits*

else *generate new harmonics via randomization*

end if

Accept the new harmonics (solutions) if better

end while

Find the current best solutions

end

Fig 2. Pseudo code of Harmonic Search

Harmony Search algorithm is inspired by the improvisation technique of jazz musicians and the suggested relationship between music and mathematics. When musicians improvise, they use a combination of the following three options: (1) play a known piece of music exactly from memory; (2) play a slightly modified version of the original piece; (3) compose a new piece by playing random notes. These correspond to the essential parts of Harmony Search algorithm: (1) selecting from harmony memory (2) pitch adjustment and (3) randomization. Simplicity and search efficiency are the two distinctive characteristics of Harmony Search algorithm.

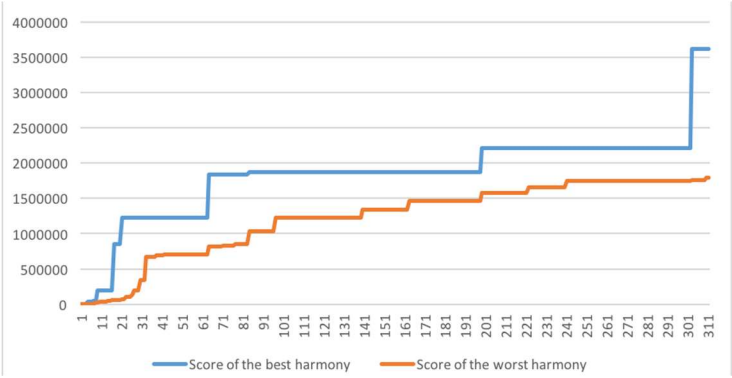

Fig 3. Trend of Learning

Harmony memory contains the five best performing sets of weights it has seen so far. New harmonies are generated based on the current harmony memory as shown in the pseudo code. A new harmony will replace the worst harmony inside the harmony memory if it is able to score better. During training, the performance of harmony memory should be strictly improving before converging. We trained our heuristics using an accept rate of 0.95 and adjust rate of 0.99. The acceptance rate is the chance of an existing weight of being accepted, and the adjust rate is the change that an existing weight is taken with an added constant of 0.05. The learning process is terminated manually as the learning speed goes slow with the increase of performance.

Performance:

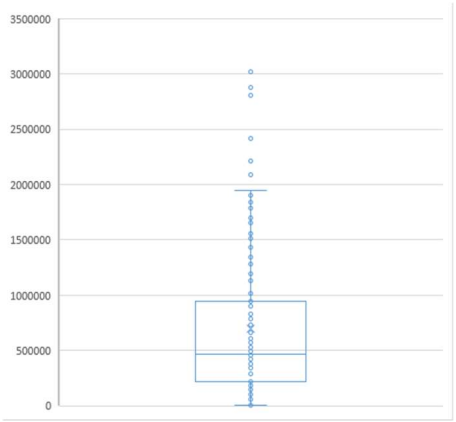

Fig 4. Box plot of test run

A Tetris agent's performance depends not only on its heuristics but also largely on the sequence in which the tetrominoes appear in the game. According to Burgiel, it is impossible to play Tetris infinitely, as there exists a sequence of tetrominoes, which will make the player lose within finite time [3]. Although we cannot test on all sequences, we have tested 120 random sequences, which will produce a relatively objective result. The box plot and Pareto diagram are shown (Fig.4 and Fig.5). The agent cleared 69.8 thousand rows on average, with 1.5 IQR reaching nearly 2 million rows.

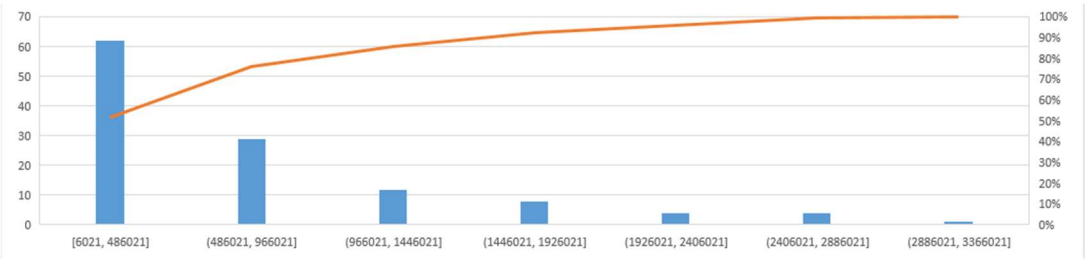

Fig 5. Pareto plot of test run

Discussion:

1. Boost up the speed of learning and scale up to large data sets

Our method: A probabilistic way in line with multithreading.

Multithreading is an effective way to make use of computational resources. But it depends on the extent that a program can be paralleled. For example, the repeated runs of one game set can be executed concurrently. However, as the game is sequential, it is difficult to accurately execute a single game across multiple threads. The game itself is extremely time-consuming, especially when the agent is capable of clearing more than 10k rows after some training. Initially, we tried to cut the height of the Tetris grid into half from 21 to 11, which could save up to thousands of execution time. However, due to the inaccuracy and inflexibility incurred, we introduced exponential distribution of landing height as a fitness function. The method is based on the probabilistic nature of the game where the probability of losing increases with the number of tetromino pieces placed. Let A_i be the event that a brick lands on the *ith* row. It is estimated that for a sensible player, $P(A_i) = exp(\lambda i+t)$, where $P(A_i)$ is the probability of a brick landing on row i, and t is a constant. Therefore, we can calculate λ by $exp(\lambda) = P(A_i+1)/P(A_i) = n(A_i+1)/n(Ai)$, where $n(Ai)$ denotes number of

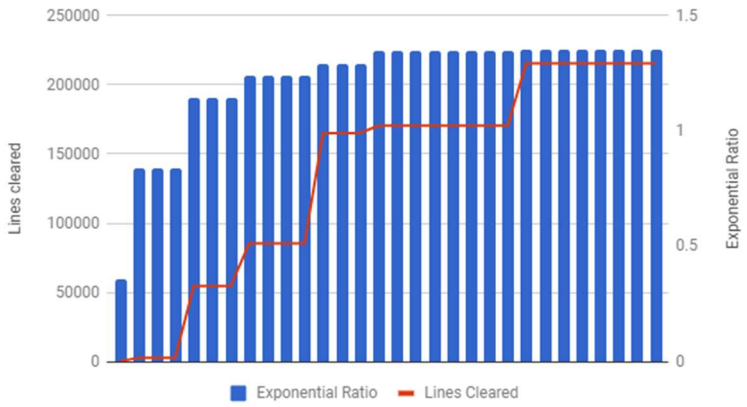

Fig 6. Positive relation between distribution and lines cleared

times a brick has landed on row i. We set a limit on the **maximum turn numbers** before the game is lost, and calculate λ by averaging $n(A_i+1)/n(A_i)$ for columns 1 to 20. By setting different maximum turn numbers, it is simple to adjust the accuracy and runtime limit per game. Empirically, a limit of 75 to 100 thousand pieces is accurate enough most of the time. Our data (Fig 6) shows that there is a strong positive correlation between the exponential ratio and the lines cleared

2. Heuristic function or learning algorithm

At the initial stage of the project, we focused more on developing a good learning algorithm. Little emphasis was placed on the heuristics. Because the common features used in Tetris AI are similar, and we assumed the performance of the agent is more limited by the learning algorithm. However, we discovered that even with a good learning algorithm, simple heuristics may cause it to converge at mediocre results. After some research and experiments, we found out that using a good set of heuristics could improve the performance significantly. The conclusion that heuristics are important is accountable as the linear model proposed is an extremely weak one. There is another possibility that the heuristics might be so powerful to conceal the effect of the learning algorithm. In other words, if the learning result does not perform much better than a set of random coefficients, the effect of the learning algorithm would be debatable. Actually, the majority of research paper online, regardless of their performance, lacks such comparison. In Fig. 7, comparing our adopted learning algorithm with random guess of coefficients, we concluded that Harmony Search algorithm has dominated Random algorithm in our scenario. Although Random algorithm has reached a noticeably good result as well.

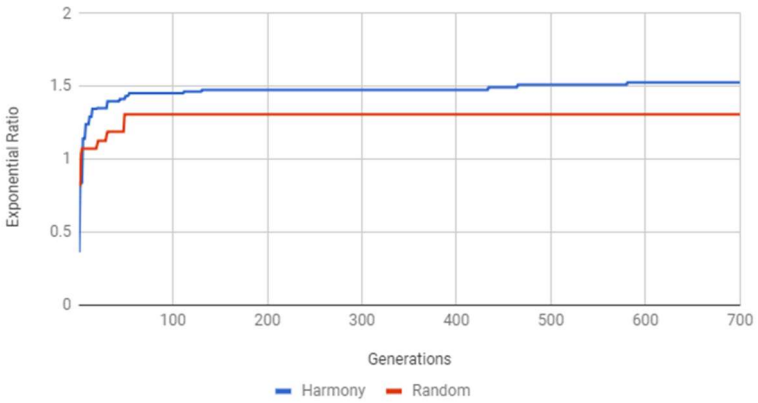

Fig 7. Harmony Search algorithm vs Random algorithm

Conclusion:

Although we stopped the learning algorithm before converging due to time and resources constraints, an acceptable result is generated. By comparing our learning algorithm with random guesses of coefficients, we prove that the learning process is effective. We have also shown some feasibility of using an exponential distribution of landing heights to measure the fitness of a Tetris agent.

References:

[1] Fahey, C. (2013). Tetris AI, Computer plays Tetris. http://colinfahey.com/tetris/tetris.html.

[2] Boumaza, A. (2013). How to design good Tetris players.

[3] H. Burgiel. (1997). How to lose at Tetris. Mathematical Gazette, 81:194–200, 1997.