**Tetris AI**

Academic work for obtaining the degree

“Master of Machine Learning”

University of Konstanz

Department of Economics

Author: Silke Husse,

Sebastian Staab

Time period: 01.08.2020 – 31.08.2020

Konstanz, 28th of August 2020

Contents

[1. Introduction 1](#_Toc47442353)

[2. Genetic Approach 1](#_Toc47442354)

[2.1. Literature Review 1](#_Toc47442355)

[2.2. Own Approach 1](#_Toc47442356)

[2.2.1. A 1](#_Toc47442357)

[2.2.2. B 1](#_Toc47442358)

[2.2.3. C 1](#_Toc47442359)

[3. Dominance of Bitcoin in cryptocurrency market 1](#_Toc47442360)

[3.1. Literature Review 1](#_Toc47442361)

[3.2. Own Approach 1](#_Toc47442362)

[3.2.1. A 1](#_Toc47442363)

[3.2.2. B 1](#_Toc47442364)

[4. Conclusion 1](#_Toc47442365)

[Appendix 2](#_Toc47442366)

[References 2](#_Toc47442367)

# Introduction

# Genetic Algorithm

## Literature Review

There have already been a number of approaches to create a Tetris AI with Genetic Algorithms. However it is difficult to compare the results between different approaches due to different rulesets for the game, different scoring methods and the time they gave their algorithm. Results in the millions might have taken months to obtain.

Thiery and Scherer (2009) achieved average scores of around 35,000,000 cleared lines with a modified evolutionary approach. Their cross-entropy method uses a gaussian distribution to generate new generations. The mean and variance of this gaussian distribution is obtained from the best candidates in the generation before. Additionally, the noise term works similarly to the idea of the mutation rate.

Lee (2013) developed a classical genetic algorithm with only 4 features that had already cleared over 2,000,000 lines by the time he had to stop it (running time ~ 2 weeks).

Young (2018) shows that with the number of features the results get less of an increase per new generation but take longer to converge, resulting overall in higher scores.

## Our Approach

### General Idea

The general functionality of a Genetic Algorithm is derived from Biology. At the core of it, we have the optimization problem: Where should the current tetromino be placed? An evaluation function computes a score of the tetromino on every possible position (potentially also with regards to the next following tetrominos) based on the dot product of a feature vector and a weight vector.

The feature vector describes the state of the game (e.g. altitude/height of filled tiles or number of lines cleared). It is a design choice and it is stable throughout the game.

The weight vector is the objective of the optimization. We want our weights to produce the optimal evaluation function to achieve the best performance in our Tetris AI. To get the optimal weights we use a process derived from evolution:

We start with Generation 0, where all weights are randomized. We have a population (N) of different candidates - different weight vectors but same feature vectors, forming N different evaluation functions that are able to play the game. The overall performance of a candidate (or “chromosome”) is evaluated via a fitness function.

After each iteration we simulate natural selection. The better a candidate performs, the more likely it will contribute to the next generation. This is done with different operators:

* Selection: Candidates are added to the next generation without changes.
* Mutation: The candidate is added with some small changes to the weights.
* Crossover: A pair of candidates is combined into a new candidate.

The population size will stay the same after each iteration. The goal is to evolve the weights to their optimal values by increasing the performance with each generation.

### Design Choices

The design choices include:

* Population size N
* Number of games each candidate plays in each generation to receive its score
* Feature selection
* Randomized values
* Allocation of natural selection operators (Crossover-rate, Mutation-rate, Selection-rate)
* Intensity of mutation
* Form of crossover
* Tetris filed height & width
* Move definition
* Move Bounds (how long is the game allowed to be)
* …

### C

# Reinforcement Learning

## Literature Review

## Our Approach

### A

### B

# Conclusion

# Appendix

# References

Lee, Yiyuan (April 14, 2013), Tetris AI – The (Near) Perfect Bot. Code My Road:

Programming Projects and Articles: <https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/> (Accessed 09.08.2020)

Lewis, Jason (2015), Playing Tetris with Genetic Algorithms.

<https://www.semanticscholar.org/paper/Playing-Tetris-with-Genetic-Algorithms-Lewis/3e40a48f6fa2c17e933dd6b1c707fd43aae9d176> (Accessed 09.08.2020)

Cheung, Joshua and Karpur, Ajay and Lafrance, Frederic and Meeke, Iain (April 18,

2015), Dead Man’s Tetris. <https://raw.githubusercontent.com/ajaykarpur/tetris-ai/master/report.pdf>. (Accessed 09.08.2020)

Fahey, Colin (July, 2012), Tetris. <https://www.colinfahey.com/tetris/> (Accessed

09.08.2020)

Algorta, Simon and Simsek, Özgür (August 2017), The Game of Tetris in Machine

Learning. <https://www.researchgate.net/publication/332683605_The_Game_of_Tetris_in_Machine_Learning> (Accessed 09.08.2020)

Young, Thomas (December 2018), Genetic Tetris.

<https://github.com/thyo9470/Genetic-Tetris> (Accessed 09.08.2020)

Thiery, Christophe and Scherrer, Bruno (2009), Improvements on learning tetris with

cross entropy. Icga Journal, 32(1): 23–33, <https://hal.inria.fr/inria-00418930>

(Accessed 09.08.2020)