NANYANG TECHNOLOGICAL UNIVERSITY

REINFORCEMENT LEARNING FOR TETRIS

Tan Sheng Hock

School of Computer Science and Engineering

2019

NANYANG TECHNOLOGICAL UNIVERSITY

SCSE18-0161

REINFORCEMENT LEARNING FOR TETRIS

Interim Report

28 January 2019

Tan Sheng Hock

U1621606C

Supervised by Assoc Prof Xavier Bresson

School of Computer Science and Engineering

2019

Abstract

This project presents the development and implementation of an agent that can play Tetris, which learns how to maximize along a specific dimension over many steps using reinforcement learning; in this case, maximize the points won in a game of Tetris over many moves. Like how humans learn to achieve a better score, through multiple interactions with the Tetris game environment, reinforcement learning is just a computational approach to learning from action.

This project includes the development of the Tetris game environment, the development of the self-playing Tetris agent and the implementation of reinforcement learning.

The source code of the project can be found on https://jcodesh.github.io/.

Table of Contents

Abstract i

Table of Contents ii

1. Introduction 1

1.1. Background 1

1.2. Purpose and Scope 1

2. Model Implemented 2

2.1. Genetic Algorithm 2

2.1.1. The Notion behind Genetic Algorithm 2

2.1.2. Feature Selection 4

3. Future Work 6

1. Introduction

1.1. Background

Machine Learning is an application of Artificial Intelligence (AI) that provides computers with the ability to learn from examples and experience without being explicitly programmed. Motivated by how human beings learn from examples, experience, machine learning focuses on the development of computer programs that can teach themselves to grow from data and change when exposed to new data.

Games are more than just only entertainment, it is commonly used as a testbed for AI, where agents are trained to outperform human players by optimizing its score. The standard Tetris is a stochastic, open-ended board game is an example of such games that is a suitable environment for such development. First developed by Alexey Pajitnov in 1984, had become popular ever since. The game being a very well-known game is now widely available on most gaming consoles and computer systems.

The game field is made up of 10 by 20 cells. Players control the current falling pieces randomly chosen from 7 pre-defined shapes of blocks and try to build fully occupied rows, which are removed in each turn. In addition to the current piece, the shape of the next piece is also made known to the player. The objective of the game is to place the pieces in the optimal position in the game field to best accommodate subsequent pieces and clear as many rows as possible. Wrong placement of pieces may often result in an undesirable situation, which may lead to spending additional time to manage the game field. The game ends when the top of the game field is filled, and no pieces can be placed onto the field.

1.2. Purpose and Scope

The aim of this project is to explore and implement the self-playing Tetris agent to maximize the points won in a game of Tetris over many moves. Using reinforcement learning-based algorithm, the agent is trained and will learn the optimal policy to

perform the optimal set of actions to eliminate as many rows as possible in order to perform as many moves as possible to achieve a high score.

2. Model Implemented

2.1. Genetic Algorithm

Genetic Algorithm (GA) is an adaptive heuristic search algorithm which is based on the evolutionary ideas of natural selection and genetics. The algorithm is used to solve optimization problems by utilizing historical information to direct the search into the region of better performance within the search space. The basic concept of GA is to simulate the processes in the natural systems for evolution, following the principle of "survival of the fittest" where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

2.1.1. The Notion behind Genetic Algorithm

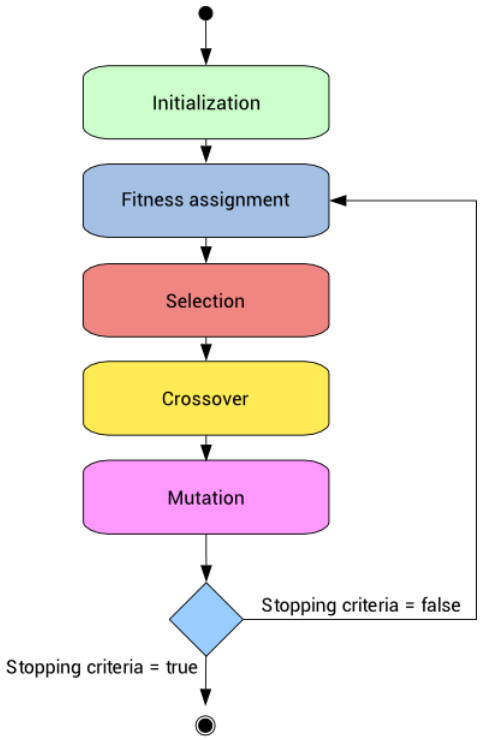

Figure 1 – Genetic Algorithm Sequence

Initialization

The initial step is to create and initialize all the individual in the entire population. The genes of each individual are initialized at random as the genetic algorithm is a stochastic optimization method. The genes also are known as the parameters of the game state will be discussed later.

Fitness Assignment

After the population has been generated and initialize, a fitness value is assigned to each individual. Individuals with great fitness will have a higher probability of being selected for recombination. The aim is to make the child of this generation obtain the best properties of any two individuals from this generation.

Selection

After performing the fitness assignment, the selection operator, in this case, individuals with the top 5 fitness values, will be chosen to be recombined for the next generation.

Crossover

After selection, two individuals from a generation are selected and their features are randomly chosen to combine. This provides us with a slight chance of producing an individual in the next generation which is better than the current generation.

Mutation

Performing crossover may generate individual in the next generation with features similar to their predecessor. Issue such as low diversity might arise in the new generation, and to prevent the solution from converging to a solution which is not universally optimal, a mutation in some of the values of certain features are done at random.

With the new population, each generation is likely to be more adapted to the environment than the old one, and the whole process is performed repeatedly until a stopping criterion is satisfied.

2.1.2. Feature Selection

Completed Lines

This feature represents the weight of each row cleared by the given move. With more rows cleared, the more this weight increases as we will want to maximize the number of complete lines because clearing lines are the goal of the agent and it will give us more space for more subsequent pieces.

Maximum Height

This feature represents the height of the highest column to a certain scalar. We will want to minimize the number so that the algorithm can detect if the blocks are stacking too high

Aggregate Height

This feature represents the sum the height of all columns, which is the distance from the highest tile in each column to the bottom of the grid. We would like to minimize the number because a lower value means more piece can be added onto the game field before hitting the top of the grid.

Relative Height

This feature represents the difference in the height of the highest column and the height of the lowest column. We would want to minimize this value to ensure that the pieces are not constantly placed on one side of the game field.

Holes

This feature represents the sum of all the empty cells that have a block above them, which is those cells that are unable to be filled. We would have to minimize this number as holes are harder to clear as we would have cleared the row above it before reaching the hole before filling up the hole.

Roughness

This feature represents the sum of the absolute differences between the height of between two adjacent columns. The value tells us the variation of the column height, and we would like to minimize this value to ensure the top of the grid is as flat as possible.

3. Future Work

Over the next two months of the project, the following points will be covered:

1. **Experimenting with human boosted data**: Comparison of time-wise progression and improvement between the agent that learn with and without human boosted data. This is to determine if providing information to the agent would be able to achieve similar results in a shorter time or possibly produce an agent that can perform significantly better.
2. **More user-friendly application**: Currently all the parts of the project is in separate projects. A plan on compilation into a single application that can perform all the requirements.
3. **Experimenting with different weight of the different features**: Plans to perform more experiments on changing how each feature affects the weight used to calculate the best action.
4. **Optimization of search when finding the next best action**: Currently, the project uses brute-force to search for the next best action, looking through every possible action. Plans to investigate how the search is performed to prune redundant nodes to reduce the time taken for the agent to learn.