Tetris AI with Genetic Algorithm

Machine Learning and Data Mining

Group 8

University of Science And Technology of Hanoi

ICT Department

October 18, 2018

Introduction

Tetris

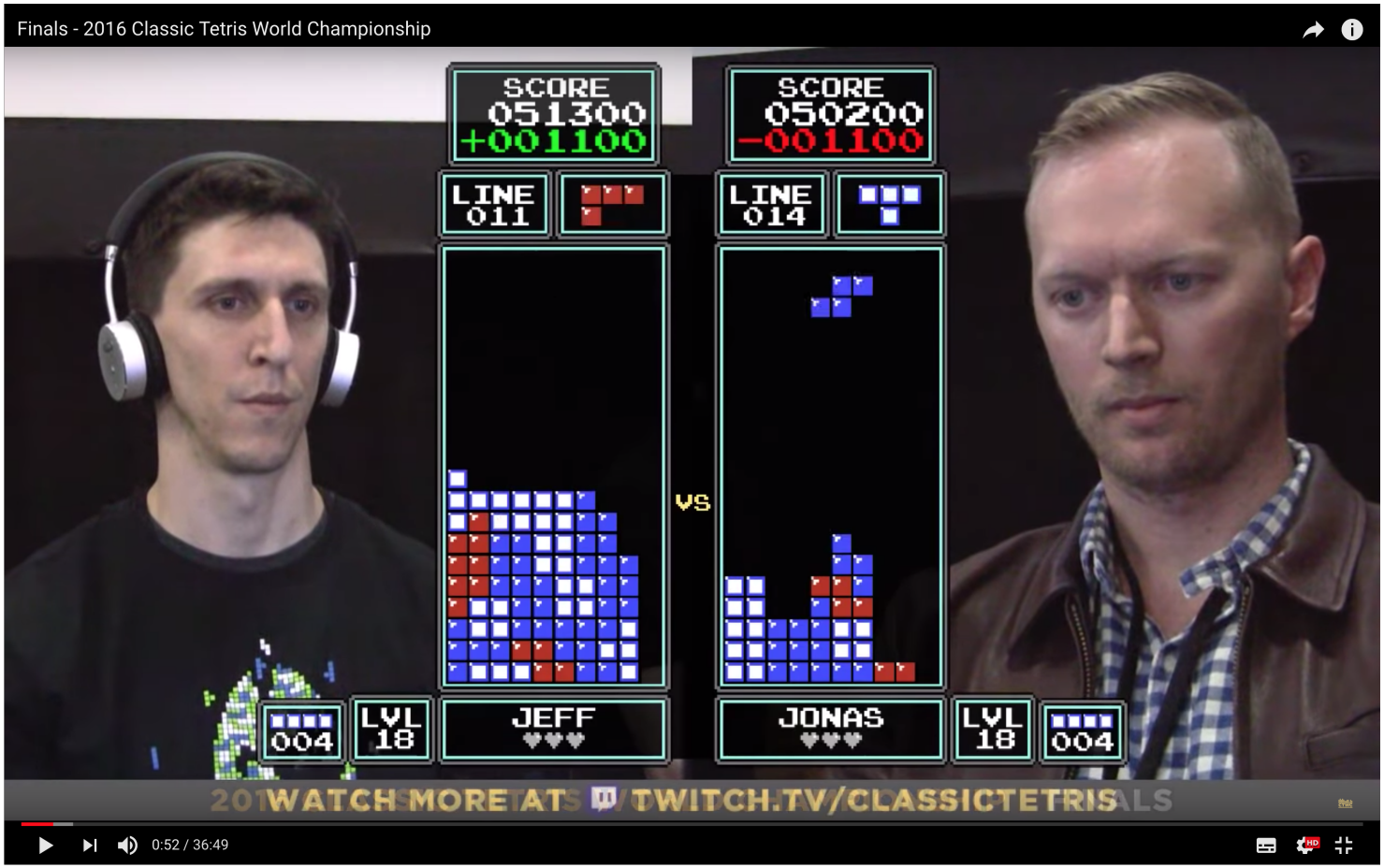


Figure 1: Classic Tetris World Championship

Tetris

- Tetris is one of the most amazing games since 1984 so far.
- Tetris is an interesting game to play with because it combines both luck and skill.
- Tetris is hard, even to approximate
- Additionally, Tetris has been a common topic in artificial intelligence.

→ Our team want to create an AI to play tetris using genetic algorithm.

Goal

- Make as many moves as possible without losing.
- Efficiently score points by making more tetrises.

Tetris

Tetris

- Grid: 22x10 cells.
- There are 7 Tetrominoes: “I”, “O”, “J”, “L”, “S”, “Z”,“T”.
- The Rotation System is used for all rotations.

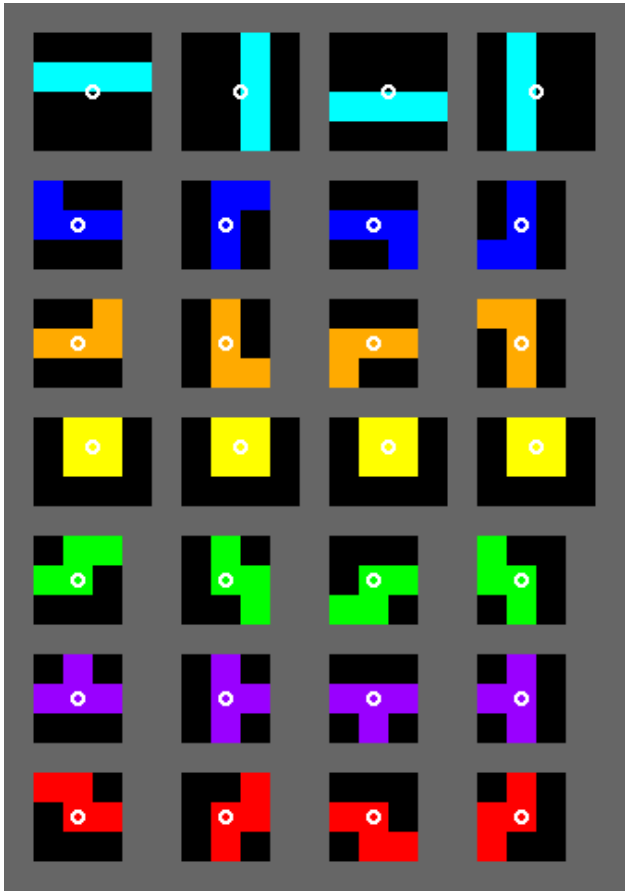


Figure 2: All rotation states of all seven tetrominoes

Heuristics

- The score for each move is computed by assessing the grid the move would result in.
- 4 heuristics which will estimate the utility of a given Tetris board:
  1. Aggregate Height.
  2. Complete Lines.
  3. Holes.
  4. Bumpiness.

Aggregate height

- This heuristic tells us how "high" a grid is.

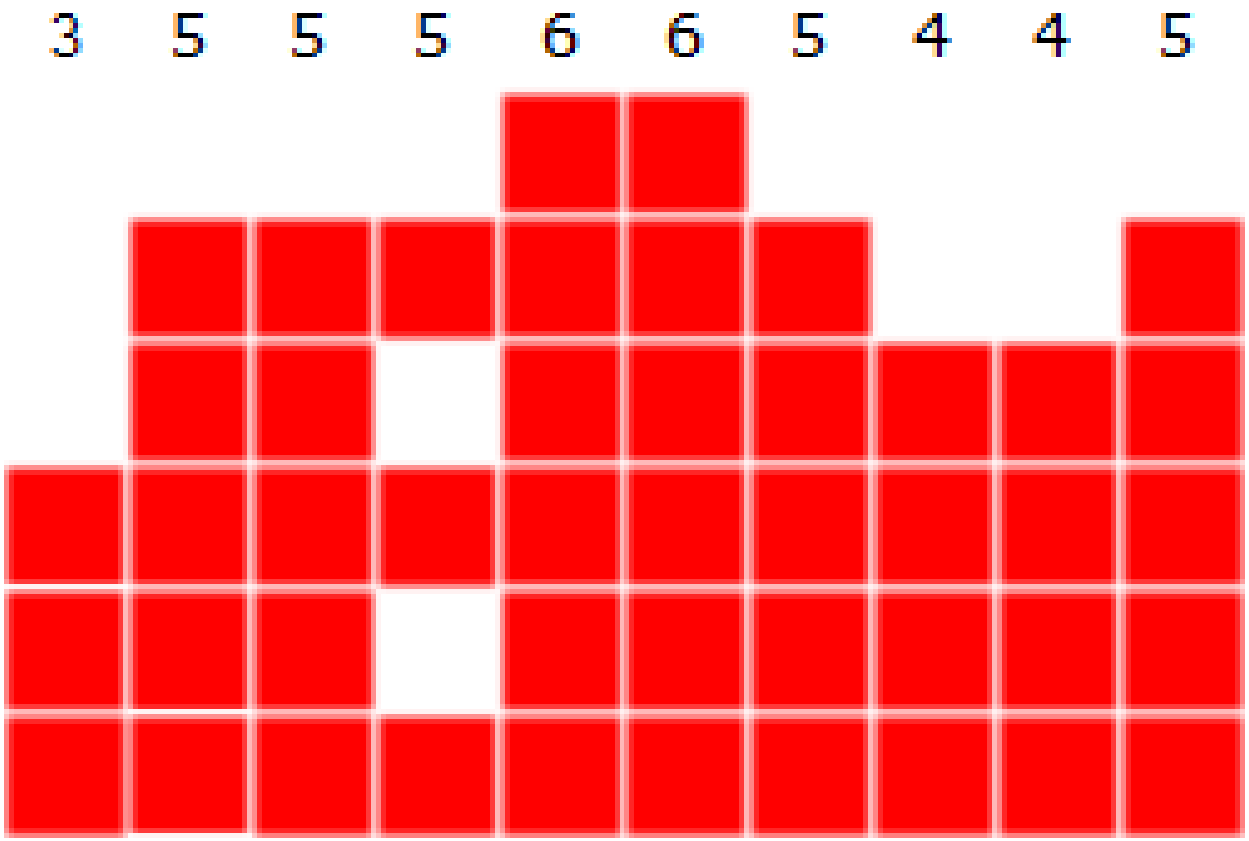


Figure 3: Aggregate Height = 48

Complete lines

- It is simply the the number of complete lines in a grid.

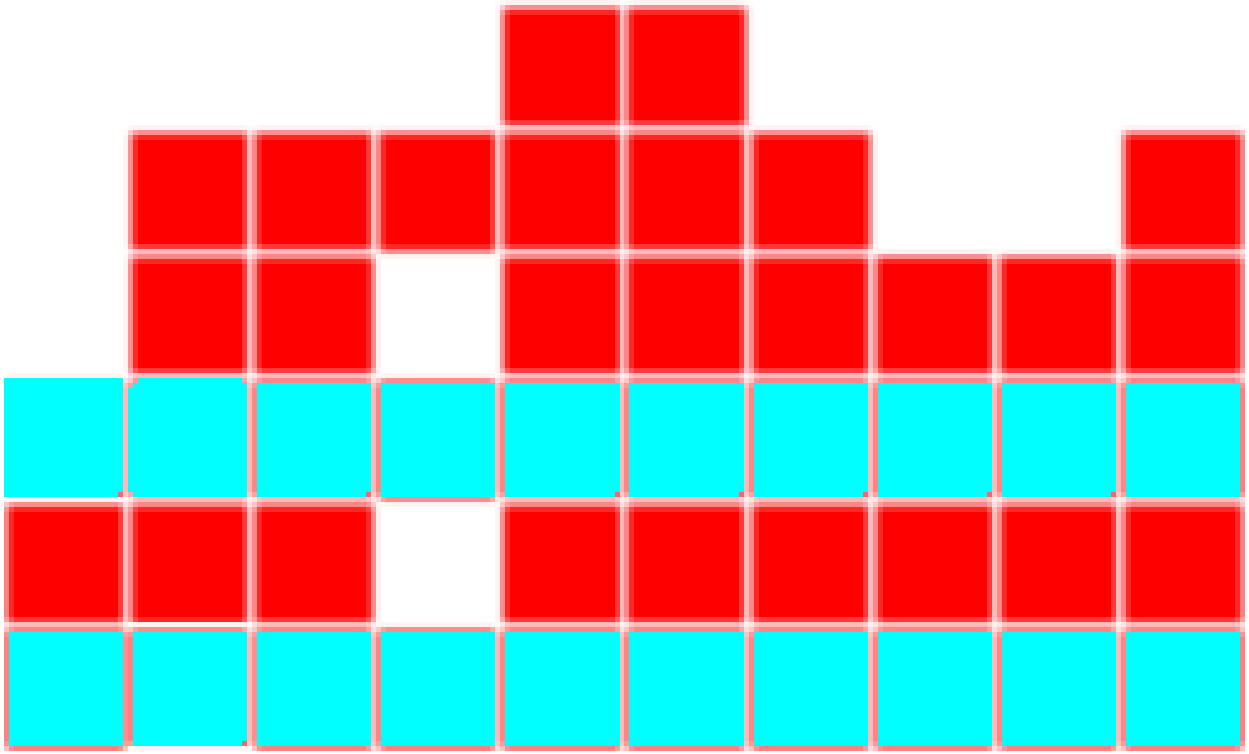


Figure 4: Complete lines = 2

Holes

- A hole is defined as an empty space such that there is at least one tile in the same column above it.

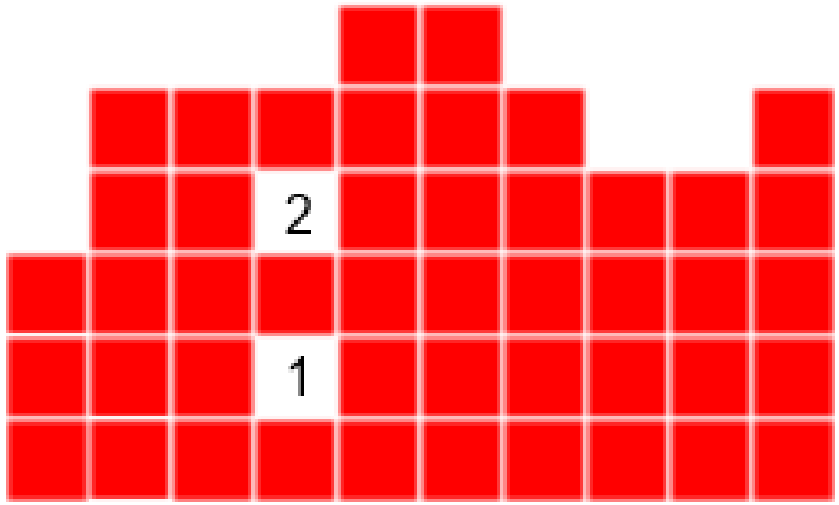


Figure 5: Number of holes = 2

Bumpiness

- The bumpiness of a grid tells us the variation of its column heights. It is computed by summing up the absolute differences between all two adjacent columns.

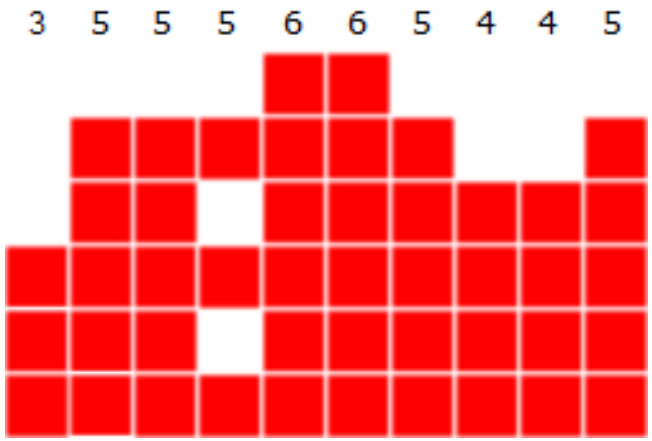


Figure 6: Bumpiness = 6

- bumpiness = 6 = |3 - 5| + |5 - 5| +. . . +|4 - 4| + |4 - 5|

Score Function

```
Score = A * Aggregate height
      + B * Complete lines
      + C * Number of Holes
      + D * Number of Bumpiness
```

Where A, B, C, and D are weights that we decide.
Check better move: if move 1 better than move 2 (Score1 $>$ Score2).

Genetic Algorithms

Genetic Algorithms

- A search technique.
- Use techniques inspired by evolutionary.
- Decide how important each heuristic is, and whether each measures a positive or negative aspect of the board.
- Find true or approximate solutions to optimization problems.

Genetic Algorithms

- Population
- Chromosomes.
- Fitness.
- Selection.
- Crossover.
- Mutation.

Chromosomes

- Subset of solutions in the current generation(set of Chromosome).
- Each “chromosome” is a dictionary of heuristics and their weights.
- How much each chromosome can influence the direction of the search is based on how long they survive, and how many offspring they produce.

```
def utility(self, board):
return sum([fun(board)*weight for (fun, weight)
    in self.heuristics.items()])
```

Population

- Subset of solutions in the current generation(set of Chromosome).

Fitness

- Indicates how well the solution to the problem

Selection

- Method of selecting an individual from a population.

Crossover

- Analogous to reproduction and biological crossover.
- This program currently defines two crossover methods:
  - Random attributes: Randomly take all required attributes from either parent.
  - Average attributes: Average each of the attributes of each parent.

Mutation

- Small random tweak in the chromosome, to get a new solution.

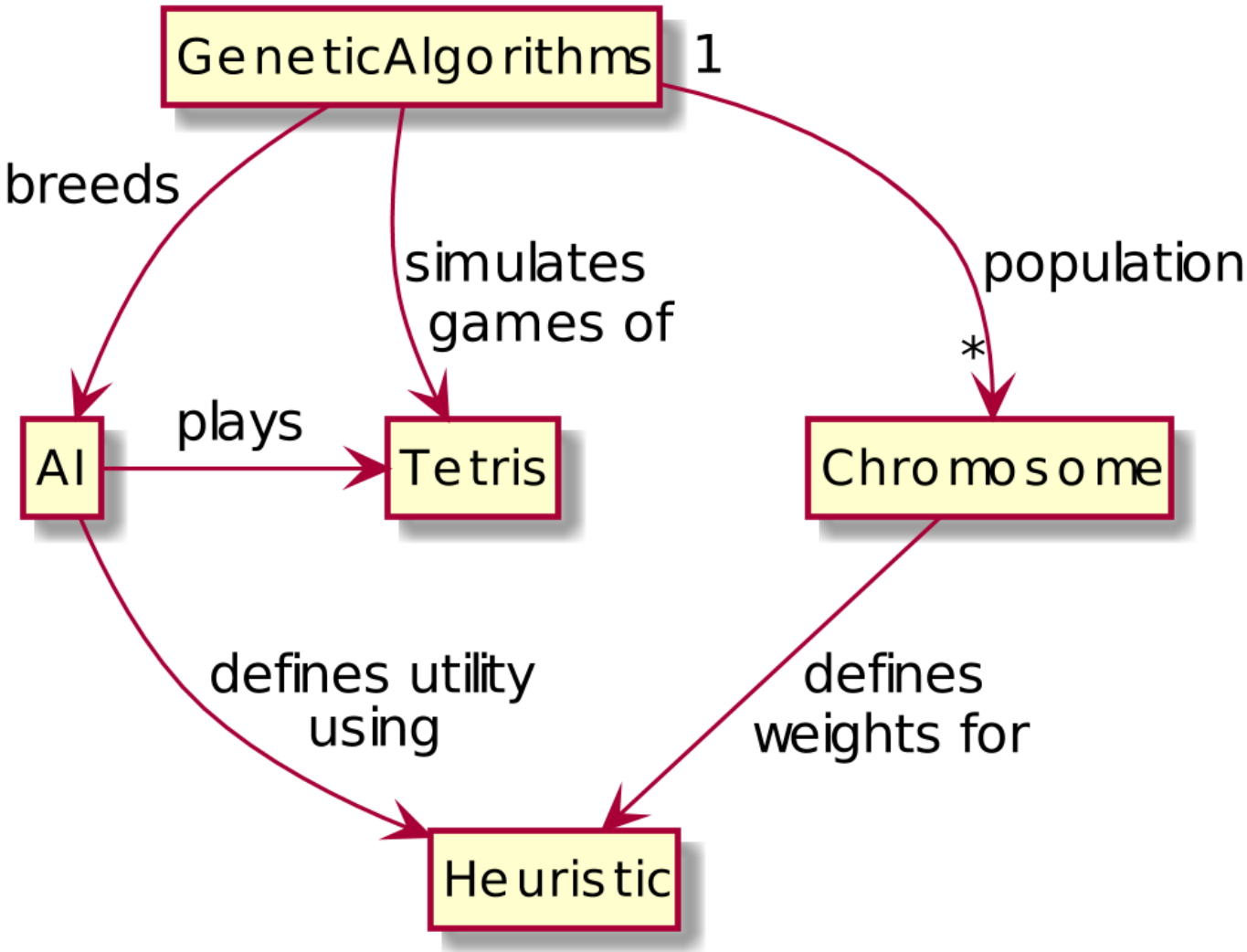
Domain model

Domain model

- A chromosome is simply a set of heuristics and their weights.
- A genetic algorithms class will generate many random chromosomes to create an initial population, and simulate many games of Tetris by swapping new chromosomes into the AI.
- Using the final score as a "fitness function".

$\rightarrow$ Determine which chromosomes were the most successful and create new chromosomes based off those chromosomes

Domain model



Algorithm

The genetic algorithm starts by creating a population of random chromosomes.

```
def __init__(self):
  self.population = [self.random_chromosome()
    for _ in range(POPULATION_SIZE)]

def random_chromosome(self):
  return Chromosome({fun: randrange(-1000, 1000)
    for fun, weight in self.ai.heuristics.items()})
```

Algorithm

- The size of the population heavily influences the performance of the program.
- Populations that are too small won't cover as much of the search space, and are more likely to get stuck in local minimums.

Genetic Algorithm

- Step 1: **Initialize** population
- Step 2: **Selection**
- Step 3: Reproduction:
  - Pick 2 parents
  - Crossover
  - Mutation
- Step 4: New population

Summary

Summary

- The AI was implemented in Python and tested with both Linux/Windown
- Make as many moves as possible without losing.
- Efficiently score points by making more tetrises.

Problem

- A “tetris” is when you clear four lines at once. The easiest way to do this is to build up four solid lines, and leave a one-block column clear on one side.
- But the AI always tries to make a best possible move (highest score).