2019-11-13 Meeting Notes

In this session, we discussed your *Tetris* coursework for ENGF0002, and some general approaches.

Basic Idea

You are required to write an auto-player for the standard Tetris. You have access to three pieces of information:

1. Current tile layout (currently "placed" tiles).
2. Current falling block (tile type, tile rotation).
3. Next falling block (tile type, tile rotation).

Approaches

You want to derive some sort of scoring function, based on heuristics, to score the current move (and the next move, with both moves combined) to determine which possible move (or moves) is the best one to take.

- You are performing breadth first search on the possibility of tile moves (because depth first search is not possible and impractical as your Tetris game could theoretically continue forever and your complexity might explode).

Some possible approaches exist:

- Brute-force breadth-first search (for each horizontal move, for each rotation) of the current move and the next move, find the best combination based on your scoring function.
- Genetic algorithms to optimize your scoring function, which optimizes the brute-force breadth-first search.

Note that these are *possible* approaches but are not the definitive solutions. Experiment on your own!

Block Types and Rotation

You have access to the **block type** and **rotation** of the current move's block and the next move's block.

Block Types

https://tetris.wiki/Tetromino

1. I-type (long stick)
2. O-type (2x2 square)
3. T-type (T-shaped)
4. S-type (right snake)
5. Z-type (left snake)
6. J-type (inverse L-shaped)
7. L-type (L-shaped)

Heurstics

Some heuristics that you may use include (*possibly* taking into account block type and rotation):

- **Holes**. For some fallen tile *t*, you check if it is surrounded by other fallen tiles (top, down, left, right). Holes are not desirable as they waste space and is difficult to eliminate.
- **Wells**. Almost like holes, except with no top fallen tiles covering. Wells may be more than one block deep. You may not want deep wells as they may be hard to eliminate apart form using I-type blocks.
- **Aggregate height**. Sum of the heights of each column. Might want to minimize, or you might want to optimize it to some height.
- **Jaggedness**. Disparity between neighbourcolumns. May want to minimize it to reduce number of wells.

And other heuristics, such as closeness-to-screen-edge, optimizing for vertical I-type row eliminations, etc.

Scoring Function

Basically, your scoring function s for move m, $s(m)$, should come up with some "rating" for your current move (and possibly the next move, combined). It can basically be the weighted sum of your individual features or heuristic, such as holes.

Let h_i denote a **heuristic**, where $h_i \in [0, 1]$ be the scoring given by the heuristic.

Let w_i denote the **weighting** corresponding to the heuristic h_i, where $w_i \in [-1, 1]$. A *negative* weighting may denote that the heuristic is undesirable, e.g., holes. The more negative, the more you intend the auto-player to avoid, etc. The weighting may denote that the heuristic is desirable.

Then the scoring function $s(m)$ is the weight sum of your heuristics for the move m.

$$s(m) = \sum_{i=1}^{n} w(i) \cdot f(i) \tag{1}$$

Note: you may want your scoring function *normalized*, i.e. for $s(x) \in [0, 1]$ so you can compare different moves more easily, such as by min-max feature scaling.

You may also want to use a composite scoring function $s'(m, m')$ where m is your current move and m' is the next move, which may give better results as your have two "lookahead" moves to aggregate information from.

$$s'(m, m') = s(m) + s'(m) \tag{2}$$

Again, you may want to normalize $s'(m, m') \in [0, 1]$ for easier comparisons.

You could also assign different weightings for the current move vs the next move, i.e. you may value the current move more, for example.

$$s'(m, m') = v_{\text{current}} \cdot s(m) + v_{\text{next}} \cdot s(m') \tag{3}$$

And possibly check that

$$v_{\text{current}} + v_{\text{next}} \equiv 1 \tag{4}$$

For a weight sum.

Genetic Algorithms

Should you choose to use a genetic algorithm, you would be able to optimize each of your weightings $w(i)$, and possibly v_{current} and v_{next}. Basically, your genetic algorithm tries with some initial default weightings, and changes that (in some direction) to yield a new generation. If that new generation is worse-off than before, we can backtrack to a previously higher-scored generation, and its corresponding weightings (we can call it some set of best weightings $\mathbf{W}_{\text{best}}$). If

the new generation is better-off with a new set of weightings $\mathbf{W}'$, than we can update $\mathbf{W}_{\text{best}} = \mathbf{W}'$.

Note that such genetic algorithm suffers from a gradient-descent/gradient-ascent deficit, where you may be able to find some local maximum (or minimum) but it is not guaranteed to be the global maximum (or minimum). Specifically, the initial weightings you start your gradient-descent/gradient-ascent or genetic algorithms could matter significantly. Additionally, how you "step" or "evolve" new generations are important as well (i.e. your rate / step size of your gradient-descent/gradient-ascent) - stepping or evolving too fast may mean you miss local maximums (or minimums).

Resources

You probably will want to know some basic concepts for *Tetris* before you start coding. Terms and phrases such as *grid*, *column*, *tile*, *block*, *block type*, etc. are good for understanding.

Tetris

https://tetris.fandom.com/wiki/Tetris_Wiki

Autoplayer

https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/

https://luckytoilet.wordpress.com/2011/05/27/coding-a-tetris-ai-using-a-genetic-algorithm/

Plagiarism Warning

Do **not** copy code from the internet. If you use code snippets or ideas from the internet, you *must* cite them. Your code will get checked for plagerism. Variable and function name and other obfuscation techniques are *not* effective because they do AST similarity checking which mitigates your obfuscation attempts.

Do **not** redistribute, print or reproduce this document.