CS182 Project Proposal

Nikhil Suri and Soumil Singh

November 2018

1 Statement

The topic of our final project is applying reinforcement learning to train an agent for playing Tetris. The problem that we intend to investigate is how effectively reinforcement learning (Q-learning, in particular) can be applied to playing Tetris. The aim of the project will be to effectively train Tetris playing agents which use different learning parameters and methods. For example, we would like to compare a tabular/sampling based Q-learning agent with an approximate Q-learning agent (also possibly multiple approximate Q-learning agents) which uses a certain heuristic and certain features to update Q-values for state and action pairs. We intend to make these comparisons between the different types of learning agents on different versions of the tetris game; that is, we will vary the size of the tetris board and the transition model we use for the sequence of pieces we present to our agent, and investigate how well each of the different Q-learning models we use work in these different situations.

How does Tetris work? The game tetris consists of a gridded game board of dimesion x by y (for some arbitrary x and y), and pieces of various non-curved shapes. The player is presented a new piece every fixed time interval. The player in practice has a limited time to decide where to place the piece; when placing the piece it must be placed on top of other piece(s), or touching the base of the board (this means pieces cannot be placed in 'mid-air'). It is the goal of the player to line pieces up in such a way the player 'completes a row', that is, arranges pieces so pieces are lined up next to each other without gaps for the entire width of the board in a single row (i.e. a complete row). When this happens, the pieces in the complete row are deleted, and all pieces above this now deleted row shift down the board by the 'height' of the row that was just deleted (height of 1 unit grid square). The player receives a large score bonus when completing a row. The player will lose the game as soon as a piece breaches the top of the board; the player has to keep completing rows so that the height of stacked pieces does not breach the top of the board. For a more detailed description of Tetris' rules, please refer to the following link: $https://www.tetrisfriends.com/help/tips_beginner.php$

2 Approach

The game of Tetris can be modeled in two parts. The first is how we place pieces on the board and plan for scoring points in the future. The second is how we receive pieces to place in every move. Below is a more in depth look at both of these aspects.

1. 'Placing pieces on the board and planning for the future' is what we want to learn an optimal policy for. Our states would be defined as the current board configuration plus the current/'newly presented' piece to place on the board. Our actions can be modeled as all valid positions in which we can place this newly presented piece, where 'valid' means a placement that doesn't violate the dimensions of the board, and doesn't overlap/collide with other pieces already present on the board. If there are no valid positions in which to place a piece, then we have lost and the game is over. In Tetris, the typical board size is 10 blocks wide and 20 blocks high. We use 7 different pieces to play. Using the above model, the upper-bound on the number of (state, action) pairs will be the number of pieces multiplied by the number of possible valid board configurations. Let us calculate an upper bound on the number of possible valid board configurations. Using the 10x20 board, we have

   $$\sum_{k=1}^{9} \binom{10}{k} = 1022$$

   ways to order a single line, assuming that the only pieces we are arranging on the board are 'unit square' pieces (the smallest possible piece). we don't count $\binom{10}{10}$ because we assume that a full line (a complete row extending from one side of the board to the other) will be cleared by the game representation/rules. We know that we can only have blocks at a higher height by stacking blocks atop each other (Blocks can't float in space without any blocks beneath them connected to them). We therefore have 20 ways to stack blocks in just a single column (stack of 1, stack of 2, etc). In two columns, there are 20^2 ways to stack blocks. Therefore, an upper bound on the number of valid configurations in our game will be

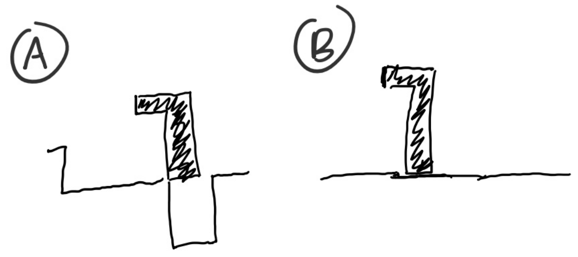
   $$\sum_{k=1}^{9} 20^k \cdot \binom{10}{k} = 6439880978200$$

   The upper bound on our state space would be this multiplied by 7 (the total number of pieces the game is played with, because a state is defined as a board configuration plus a newly assigned/current piece). This is clearly a huge state space. We have a few options to consider to fix this. If we want to continue to use an exact Q-learning agent, we can try using a smaller state space. For example, if the dimensions were halved, our number of board configurations would be

$$\sum_{k=1}^{4} 10^k \cdot \binom{5}{k} = 61050$$

, for (61050)(7) = 427350 total states. This is exponentially more manageable. We would also try using approximate Q-learning. This would require coming up with a few features for which we could learn optimal weights for. Possible features we might consider could be: the average height of the columns on the board (we prefer not to have a high average height of the columns/stacks because this might suggest we are closer to losing), and the average number of blocks per row (we might prefer a higher number of blocks per row as this might mean they are closer to being cleared).

Let us now obtain an upper bound on the number of actions for any piece. First, we should notice that, in any board configuration, we will have the same number of actions for every piece unless we are near the very top of the board and cannot fit the piece in a certain orientation. Consider the following example:



The same exact set of moves for this piece is available to us in scenario A as in scenario B. The varying depth of the board configuration in scenario A does not have an effect on the actions that we can take for this piece. Therefore, the upper bound on the number of actions for a certain piece is the maximum number of ways we can orient any piece in a completely clear board configuration. This number is given by the T-shaped, L-shaped, and J-shaped pieces. Each of these pieces have 4 orientations. On one axis, their length is bounded by 2 blocks and on the other axis, their length is bounded by 3 blocks. Therefore, there are at most $9 + 9 + 8 + 8 = 34$ actions we can take (in a board which has a width of 10 squares).

To perform Q-learning, we must have a reward function that models taking an action a from state s and moving into state s'. In Tetris, the goal of the game is to maximize our score before the board fills up and we lose. We increase our score by clearing lines of blocks. We believe that one possible way to model the rewards function is to give a reward of 0 for losing, set a living penalty of -1 (have a reward of -1 when we survive in

the next iteration), and give very large rewards for clearing lines (so the reward when we clear a line would be the set large reward for clearing a line plus the living penalty).
One topic to further explore is how we can best set these reward values to properly align incentives so that the agent learns how to evaluate planning for clearing multiple lines at once/with one action, possibly delaying clearing a single row/line when this becomes possible (which might be more optimal than clearing many single lines separately), rather than immediately choosing to clear a single line with the next action.

2. We can model receiving pieces as random, or as conforming to some sort of probability distribution or even some underlying MDP. The reception of pieces would represent our transition probabilities. Given how we receive pieces in the game, our agent should be able to learn this transition model and plan out moves which will maximize its score.

3 Goals

By the end of our project, we want to have answered the following questions:

1. How effective are reinforcement algorithms when applied to the game of Tetris?

2. For approximate Q-learning, using which features/heuristics did we get the best results?

3. How much can we extend upon this project? For example, in real Tetris we can hold a single piece in 'reserve'. Given a state which is a board configuration and a piece to place on the board, one of our actions is to swap out the current piece for our reserve piece. Can we train an effective Q-learning agent for this extended problem as well?

4. How effective are different reinforcement learning methods (exact Q-learning, approximate Q-learning, using different learning parameters, etc) when applied to different sized game boards? How scalable is this approach to different board sizes?

4 Work Process

We intend to divide up the work required for this project but continue to work closely together on the research and planning phase. Once we get to the implementation phase, we will divide up the work by each implementing different parts of the overall codebase. For example, while one of us works more closely with getting the Tetris game itself to run, the other will begin coding one of the Q-learning agents whilst in communication with the game-implementer. Once the game is implemented, the work remaining in implementing the Q-learning agents will be further divided up. Throughout, we will consult each other to ensure our code is consistent with each others' implementations.

5 Resources

We might choose to base our game off an existing Tetris game coded for the pygame platform, like the one found here: http://inventwithpython.com/blog/2010/11/18/code-comments-tutorial-tetromino/

Some other resources based to assist our development of our Q-learning agents are cited below:

Littman, Michael L. "Markov Games as a Framework for Multi-Agent Reinforcement Learning." Department of Computer Science, Brown University, 2009.

Sutton, Richard S, and Andrew G Barto. 2016.Reinforcement Learning: An Introduction 2ndEdition (Draft). MIT Press.

Bailis, Panagiotis, et al. "Learning to Play Monopoly: A Reinforcement Learning Approach." National and Kapodistrian University of Athens - Dept of Informatics and Telecommunications .