National University of Singapore
School of Computing
CS3243 Introduction to AI

Term Project: Learning to Play Tetris with Big Data!

Issued: February 7, 2018 Due: April 21, 2018

Important Instructions

1. *Your project report must be TYPE-WRITTEN using* 12*-point font. It should NOT be more than* 4 *pages (inclusive of diagrams and team details described in bullet point* 5*).*
2. *Submit a zipped file of your (a) project report in electronic format (***in PDF format***) and (b) completed "PlayerSkeleton.java" code to the workbin folder named "Project Report Submission" in IVLE by* **April 21, 2018, 11:59pm Singapore time***. Late submissions will NOT be accepted.*
3. *You should submit ONE zipped file per team.*
4. *Each team should have four to five (4-5) members; different group sizes will not be entertained.*
5. *Indicate clearly your team number (corresponding to the team number on IVLE) and all names and matriculation numbers of team members on page 1 of the project report.*
6. *Note that the TAs may call any one member of a team to explain the submitted report and demo your program during the marking process. You are not advised to free-ride.*

1 The Game of Tetris

Tetris is a popular video game played on a two-dimensional grid, which we assume to be of size $h = 20$ rows by $w = 10$ columns in this project. Each square in the grid can be full or empty, making up a "wall of bricks" with "holes". The squares fill up as objects of different shapes fall from the top of the grid and are added to the top of the wall, giving rise to a "jagged top". Each falling object can be moved horizontally and can be rotated by the player in all possible ways, subject to the constraints imposed by the sides of the grid. There is a finite set of standard shapes for the falling objects, which we assume comprises of 7 possible object shapes in this project as shown in Figure 1. The game starts with an empty grid and ends when a square in the top row becomes full and the top of the wall reaches the top of the grid. However, when a row of full squares

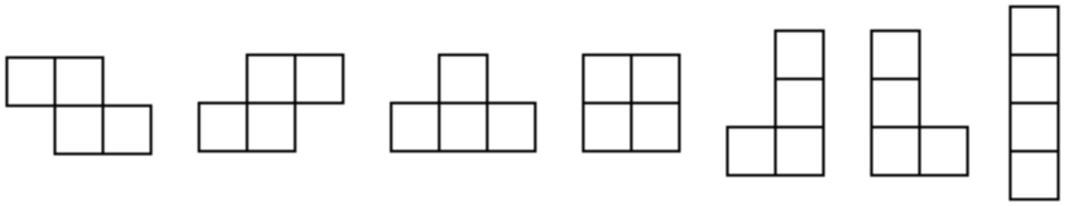

Figure 1: The 7 Tetris object shapes, each of which is an arrangement of 4 connected squares.

is created, this row is removed, the bricks lying above this row move one row downward, and the player scores 1 point. More than one row can be created and removed in a single step, in which case the number of points scored by the player is equal to the number of rows removed. The player's objective is to maximize the score attained (total number of rows removed) up to termination of the game. You can play the game yourself at `https://tetris.com/play-tetris`.

2 Heuristics for Designing a Fast, Proficient Tetris Player

The goal of this project is to develop a competent AI Tetris playing agent. The agent's action, denoted by a, is the horizontal positioning and rotation applied to the falling object. Each state of the game consists of two components:

1. The board status; this is represented by an 20×10 binary matrix $B = (b_{ij})$, where

$$b_{ij} = \begin{cases} 1 & \text{if the } ij\text{-th square is full} \\ 0 & \text{otherwise.} \end{cases}$$

2. Let $\mathcal{S}$ be the set of all possible object shapes (See Figure 1); we also know the shape of the current falling object, denoted by y, where $y \in \mathcal{S}$.

As soon as the most recent object has been placed, a new component $y \in \mathcal{S}$ is sampled independently of the preceding history of the game from the uniform distribution over $\mathcal{S}$.

Unfortunately, the number of states in the Tetris problem is extremely large. It is roughly equal to $|\mathcal{S}|2^{hw}$, where h and w are the height and width of the grid, respectively. In this project, $|\mathcal{S}| = 7$, $h = 20$, and $w = 10$, for a total of $\sim 10^{61}$ states. This motivates the need to approximate the utility function of the Tetris player with, for example, a good heuristic/evaluation function.

In particular, we can construct a heuristic function that evaluates a Tetris board position based on some characteristic features of the position. Such features are easily recognizable by experienced players, and include the current height of the wall, the presence of "holes" and "glitches" (severe irregularities) in the first few rows, etc. Consequently, the agent's utility function can be approximated by a heuristic function that comprises a linear weighted sum of features[1]. For a given grid state B, we have

$$\widetilde{V}(B) = \omega(0) + \sum_{k=1}^{n} \omega(k)\phi_k(B) \tag{1}$$

where n is the number of features, $\omega = (\omega(0), \omega(1), \ldots, \omega(n))$ is the weight vector, and $\phi_k(B), k = 1, \ldots, n$, are the feature functions for board position B. For example, the feature functions can be defined to be

1. $\phi_1(B), \ldots, \phi_{10}(B)$ correspond to the 10 column heights of the wall;
2. $\phi_{11}(B), \ldots, \phi_{19}(B)$ are the absolute differences between adjacent column heights;
3. $\phi_{20}(B)$ is the maximum column height; and
4. $\phi_{21}(B)$ is the number of holes in the wall, that is, the number of empty positions of the wall that have at least one full position above them.

Note that you do not have to adhere to the feature functions that are specified above; you are welcome to specify your own. Even though you may be tempted to handcode your Tetris agent by simply specifying the values of the weight vector ω, we require you to learn ω using data (e.g., see [1]).

Given a board position B and shape of current falling object $y \in \mathcal{S}$, the agent's strategy μ for playing the Tetris game is defined as follows:

$$\mu(B, y) \triangleq \text{argmax}_a \; r(B, y, a) + \widetilde{V}(Brd(B, y, a))$$

Here, $r(B, y, a)$ is the reward function: the number of rows removed when player action a. The function $Brd(B, y, a)$ outputs the board state after applying action a, given that the input shape was y and the previous board status was B. The function $\widetilde{V}$ is the utility function defined in (1).

3 Grading Criteria

Note that you can propose your own agent's strategy; you do not necessarily have to implement the utility-based agent specified in Section 2.

All members of the team will be given the same grade/mark[2]. The grading criteria are:

[1]It is a common approach to specify a heuristic function using a linear weighted sum of features. For example, see sections 3.6.4 and 5.4.1 of AIMA 3rd edition.

[2]Free riders are given zero.

15%	**Project report**: This 4-page report should document your agent's strategy, experimental results demonstrating its performance, and observations and analysis to discuss why your strategy performs well (or doesn't). State clearly and explicitly the *novel* and *significant* contributions and achievements (if any) of your team that, in your opinion, will distinguish your agent's strategy and project work from that of the other teams and the existing literature. In particular, thanks to the recent hype on big data, you need to demonstrate that your agent's *learning* method can scale up to big data; that is, it has to be capable of learning a competent/proficient Tetris agent using millions, billions, or even trillions of Tetris objects in a sufficiently short time. Describe, in the report, your proposed novel learning method that is *EXPLICITLY*[3] designed and implemented to handle big data and empirically demonstrate through experiments that it can indeed scale up to big data (e.g., by reporting the speedup[4]) while preserving the agent's good learning performance. One possibility is to consider parallel/distributed learning on multiple cores/machines[5]. We will place particular emphasis on the quality of your writing: make sure that you write in clear, correct, academic English. I recommend that you use LaTeX, rather than MS Word to typeset your work; in any case *the submitted file must be in PDF format*.
4%	**Wow! factor**: when reading your report, I should be impressed by at least one part of the implementation; for example, an interesting approach to learning from data, an unorthodox choice of features, a novel agent design paradigm. This should be highlighted and demonstrated in the report.
3%	**Performance against the agents designed by other project teams**: We will test your agent against our favorite sequence of object shapes, which is of course not revealed to you! Be ready to demo your program when called upon by the TAs. You will not receive any points in this category if your code cannot be compiled on our machine.
2%	**Project program**: A template written in java is provided. For the java code, you should only be modifying the "PlayerSkeleton.java" file. The rest of the template should NOT be modified to suit your needs; such requests will NOT be entertained. You should provide detailed comments in your code to facilitate our understanding. Zip your completed "PlayerSkeleton.java" code with the project report for submission. You will not receive any points in this category if your code cannot be compiled on our machine.
1%	**Following instructions**: The instructions are provided on page 1.

[3]Note that some simple learning methods may already be *implicitly* capable of handling big data. How have you *explicitly* modified it in a novel way to speed it up or cater to many times more data given the same amount of time for learning?

[4]Speedup is the incurred time of a sequential/centralized algorithm divided by that of its parallel/distributed counterpart.

[5]NUS offers several excellent computing clusters; you may also use the NSCC. However, we do not offer any dedicated computing resources for this task.

References

[1] D. P. Bertsekas and S. Ioffe. Temporal differences-based policy iteration and applications in neuro-dynamic programming. Technical Report LIDS-P-2349, MIT, 1997.