Theory of Computer Games 2018 - Project 3

In the series of projects, you are required to develop AI programs that play *Threes!*, the origin of other 2048-like games.

Overview: **Solve the 2x3 *Threes!***

1. Modify the board size to 2x3.
2. Implement **expectimax search algorithm** with transposition table.
3. Calculate the value of the entire game tree.

Specification:

1. The rules of 2x3 Threes! for Project 3:
   a. The board size is **only 2x3**. (changed)
   b. The initial states **only contain 1 tile**. (changed)
   c. Tiles (including the initial tile) should be generated from the bag. The bag size is 3, no bonus tile.
   d. Tiles should be generated **at any empty cell on the opposite side** of last sliding direction.
   e. The board value is the sum of $3^{\log_2\left(\frac{v}{3}\right)+1}$ of all v-tiles with $v \geq 3$.
   f. For more details, please refer to [1] and [2].

0	1	2
3	4	5

1-d array form

2. The position of cells in a 2x3 game board are defined in 1-d array form.
3. The game tree of 2x3 Threes! In Project 3 is defined as
   a. The root node (layer 0) is an empty state.
   b. Each edge represents a legal action played by either environment or player.
   c. **The n[th] layer contains states that n actions have been applied**.
      i. Nodes in layer 1, 3, 5, 7, … are before-states (a.k.a. max node).
         Nodes in layer 0, 2, 4, 6, … are after-states (a.k.a. expected node).
      ii. See the Methodology for more details
4. The program should use **standard input** and **standard output**.
   a. Input: `s t0 t1 t2 t3 t4 t5 +h`
      i. Input has several lines and ends with EOF. Each line contains a test case.
      ii. `s` is the state type, **will be either character `a` (after-state) or `b` (before-state)**.
      iii. `t0 ~ t5` represent the **tiles of the state** (in 1-d array form, **0 ≤ t_i ≤ 6144**).
      iv. `h` will be **the hint of next generated tile** (character `1`, `2`, or `3`).
   b. Output: `input = value`
      i. Output a single line for each test case.
      ii. `input` should be the input line of this test case. (`s t0 t1 t2 t3 t4 t5 +h`)
      iii. If `input` is a valid state, `value` should contain three values: **`min avg max`**. Where `min/avg/max` is the expected/min/max value of the given board when oracle play; otherwise `value` should be `-1`.
      Check the Methodology for the definition and description of these values.

iv. If there are more than one valid output, just output one.

5. Transposition table is required.
   a. The solver should be able to calculate the game tree within 10 minutes, and answer at least 1 test case per millisecond. See the scoring criteria for details.
   b. Your program **should not use any pre-calculated external database**.
6. Implementation details:
   a. Your program should be able to compile and run under the workstation of NCTU CS.
      i. Write a makefile (or CMake) for the project.
      ii. C++ is highly recommended for TCG.
      You may choose other programming language to implement your project.
   b. Your implementation needs to follow the standard output format.
   (see above description)
   c. The representation error of floating point should be less than 0.001.

Methodology:

1. Expectimax approach: **expand the tree** from the root node and **store the result**.
   a. At before-states, the player always applies the optimal action based on the average value.
      i. Therefore, the min/avg/max for a before-state are its best successor's value.
      ii. For a terminal state, you can directly calculate its value.
      iii. Terminal states are always before-states.
   b. At after-states, the environment generates the new tile with probabilities.
      i. Note that the environment needs to follow the hint tile and generate the new tile at the opposite side of last action.
      ii. However, you may have multiple legal positions to generate the new tile.
      iii. The min/max value for an after-state will be its successors' min/max value.
      The avg value for an after-state is the average of successors' values.
   c. Terms min/avg/max are defined as:
      i. Avg: the expected value of target node.
      i. Min: the min value of target node **when you are catastrophically out of luck**.
      i.e. the environment always generates the worst tile at the worst position, "worst" in terms of corresponding value. Note that the player always tries to maximize its value at every before-states.
      ii. Max: the max value of target node **when you are extremely lucky**.
      i.e. the environment always generates the best tile at the best position, "best" in terms of corresponding value.
   d. Use a large table to store the information of states.
      iii. Be careful with before-states and after-states. You should store their value separately.
2. Besides the **board (6 tiles)** and the **state type**, you also need to take the **hint tile**, and the **last player's action** into account.

   a. You need to keep track of **state type**, **board**, **hint tile**, and **last action** when expanding the tree, they are necessary for calculating the correct value.
   b. The table for before-states should be $\mathrm{LUT_b}[board][hint]$.
   c. The table for after-states should be $\mathrm{LUT_a}[board][hint][last]$.
      i. Note that **test cases do not contain the last player's action**.
      Therefore, a test case of after-state may have more than one valid answer (e.g. "`a 0 0 0 1 3 0 +2`" has two values for different last action: down or left).
      Please simply output any one of the valid answers.
   d. Bag status (the remaining tiles in the bag, there are 7 different bag status: {1,2,3}, {1,2}, {1,3}, {2,3}, {1}, {2}, {3}) does not need to be included in the state since each valid state is exactly mapping to one bag status in the modified 2x3 Threes!
3. There are two different value output schemes, **both of them are acceptable**:
   a. Output the **expected board value of terminal state**: $\mathbb{E}[\mathrm{V}(s_T)]$.
      i. Board value is the sum of $3^{\log_2\left(\frac{v}{3}\right)+1}$ of all v-tiles with $v \geq 3$.
      ii. For terminal states, output its **board value (directly calculated by tiles)**.
   b. Output the **expected remaining reward**: $\mathbb{E}[r_t + r_{t+1} + \cdots + r_T]$.
      i. For initial states, the remaining reward is the same as the expected board value.
      ii. For terminal states, output 0 (since there is no remaining reward).
      iii. You should use **the difference of board values** as the reward.
4. Never recalculate the value of the same state.
   a. Note that **state = board + type + hint + last**.
5. **Isomorphic states** always have the same value. Do not recalculate them.
   a. "`b 1 0 0 0 0 0 +2`" and "`b 0 0 0 0 0 1 +2`" have the same value.
   b. In practice, you can also update the isomorphic states after calculating a state. You need to **modify the board and the last action** when updating isomorphic states.
   c. In fact, recalculate these isomorphic states won't take you too much time in this project. So, you can simply let your solver recalculate them.
6. Note that the max tile of 2x3 Threes! is obviously less than 6144-tile (14-index).
7. **Remember to exclude illegal states**. Your program should output "`= -1`" for them.
8. Sample program is provided, which is a non-implement AI that plays *2048*. You are allowed to modify everything (remember to follow the specification).

Submission:

1. Your solution **should be archived in zip/rar/7z file**, and **named as `XXXXXXX.zip`**, where XXXXXXX is the student ID (e.g. `0356168.zip`).
   a. Pack your **source files**, **makefiles**, and other relative files in the archive.
   b. Do **NOT** upload the statistic output or the network weights.
   c. Provide the version control repository of your project (URL), while do **NOT** upload the hidden folder (e.g. **.git** folder).
2. Your project should be able to run under the workstations of NCTU CS (Linux).
   a. **Test your project on workstations**.

b. Only run your project on workstations reserved for TCG. Do not occupied the normal workstations, otherwise you will get banned.

Scoring Criteria:

1. Demo: **TBD**.
2. Test cases (100 points): Pass all the test cases.
   a. 100 test cases, 1 point for each correct answer.
   b. See the attachment for sample input and output. (TBA)
   c. The **online judge** will be released later. You can test the solver before project due.
3. Penalty:
   a. Time limit exceeded (–30%): Slower than 10 minutes.
   b. Late work (–30%): Late work including but not limited to **uncompilable sources** or **any modification** after due.
   c. No version control (–30%).

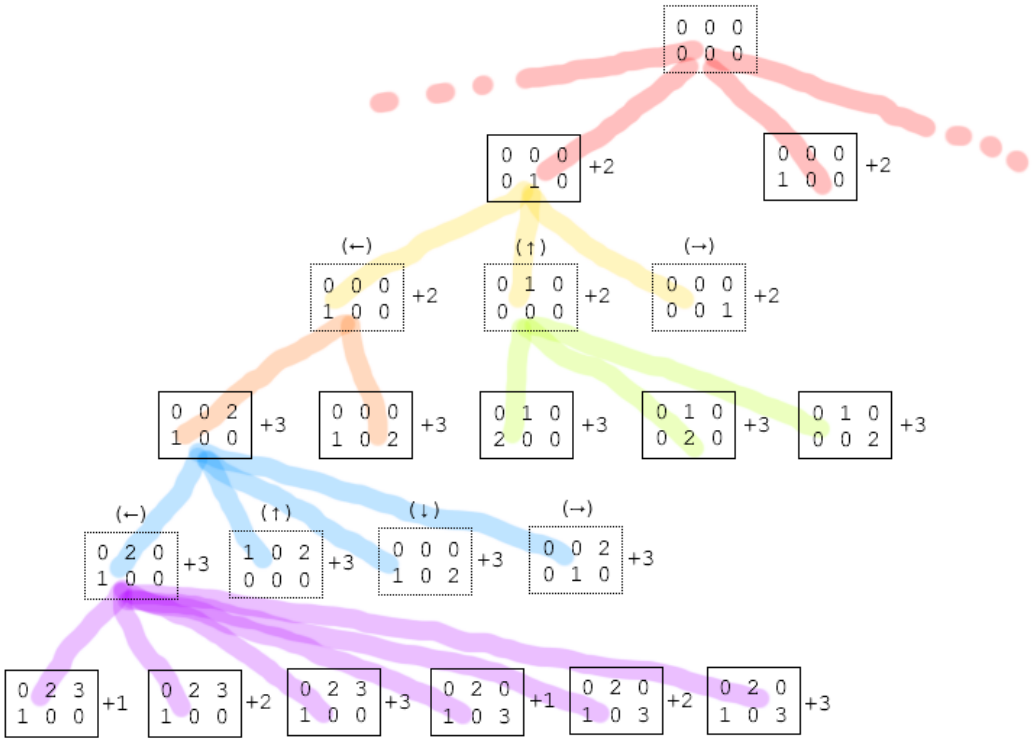
References:

[1] Multi-Stage Temporal Difference Learning for 2048-like Games. https://arxiv.org/ftp/arxiv/papers/1606/1606.07374.pdf

[2] Threes JS. http://threesjs.com/

Appendix:

1. Root of initial state "`b 0 0 0 0 1 0 +2`":



Hints:

You are NOT required to use the framework of project 3 since the **it only provides the I/O implementation of sample I/O format** (you can keep working on project 1's framework, however, remember to create some branches).

Version control (GitHub, Bitbucket, Git, SVN, etc.) is recommended but not required.

Having some problems? Feel free to ask on the Discussion of e3 platform.

You may use Github Student Developer Pack or Bitbucket for the version control.

Remember to share the sources on sharing platform, for example, GitHub Gist.