Theory of Computer Games 2017 - Project 3

In the series of projects, you are required to develop AI programs that play *2584 Fibonacci*, a 2048-like game, which is similar to the one at here.

Overview: Solve $2 \times 3$ *2584 Fibonacci*.

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

Specification:

1. The rules follow the original rules, except for:
   a. Environment should drop **1-tiles** or **2-tiles** with probabilities of **0.9** and **0.1**, respectively.
   b. The distribution of initial state (with two tiles) is equivalent to dropping two tiles (with probabilities mentioned above) on an empty board.
2. The position of grids in a 2x3 game board are defined as

| 0 | 1 | 2 |
|---|---|---|
| 3 | 4 | 5 |

1-d array form

| 0,0 | 0,1 | 0,2 |
|---|---|---|
| 1,0 | 1,1 | 1,2 |

2-d array form

3. The game tree of *2584 Fibonacci* is defined as
   a. The root node 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 2, 4, 6, 8, … are before-states (a.k.a. max node).
      Their expected values are the maximum of their successors' reward and value, or 0 if no successors.
      ii. Nodes in layer 3, 5, 7, 9, … are after-states (a.k.a. expected node).
      Their expected value is the weighted mean of their successors' value.
      iii. Note that **nodes in layer 0 and 1** are neither before-states nor after-states.
4. The program should use **standard input** and **standard output**.
   a. Input: `x t0 t1 t2 t3 t4 t5`
      i. Input has several lines and ends with EOF. Each line contains a test case.
      ii. Character `x` is the type of state, **will be either `a` (after-state) or `b` (before-state)**.
      iii. Integers $t_0 \sim t_5$ represent the 2x3 board (in 1-d array form, $0 \le t_i \le 33$).
   b. Output: `= v`
      i. Output a single line for each test case.
      ii. If input test case is a valid state, `v` should be the expected value of the given board when oracle play; otherwise `v` should be `-1`.
5. Transposition table is required.

   a. The solver should be able to calculate the game tree within 1 minute, 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. You program should be able to compile 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, however, the scoring criteria (time limit) will keep unchanged.
   b. The **representation error of floating point** should be less than 0.001.

Methodology:

1. Once the solver starts, **expand the game tree** from the root node and **save the result**.
   a. Use a large table to store the information of states.
2. The time limit for answering 1000 test cases would be 1 minute and 1 second.
   a. Expand the game tree and save the result: ≤ 1 minute.
   b. Answer the questions: ≤ 1000 milliseconds (for 1000 test cases).
3. **Isomorphic states** in the same situation always have the same value.
   a. The answers to "`b 2 2 0 0 0 0`", "`b 0 2 2 0 0 0`", "`b 0 0 0 2 2 0`" and "`b 0 0 0 0 2 2`" are all the same.
4. **Be careful with before-states and after-states**. Take "`2 0 2 0 0 0`" and "`0 0 0 0 4 2`" as examples.
   a. The answers to "`b 2 0 2 0 0 0`" and "`a 2 0 2 0 0 0`" are the same.
   b. The answers to "`b 0 0 0 0 4 2`" and "`a 0 0 0 0 4 2`" are **not** the same.
5. **Remember to exclude illegal states**. Note that all the following test cases are illegal states, and your program should output "`= -1`".
   a. Root node is neither before-state nor after-state: "`a 0 0 0 0 0 0`".
   b. Nodes in layer 1st are neither before-state nor after-state: "`a 0 0 1 0 0 0`".
   c. Unreachable nodes: "`b 1 1 1 1 1 1`", "`a 24 4 18 3 0 0`".
6. **Sample program is provided**, which is a non-implemented solver that plays 2x3 *2048*. You are allowed to modify everything (remember to follow the specification).

Submission:

1. Your solution **should be archived in zip file**, and **named as `ID_vX.zip`**, where X is the version number (e.g. `0356168_v1.zip`, `0356168_v2.zip`).
   a. Upload your **source files**, **makefiles**, and other relative files.
   b. (Optional) Provide the Git repository of your project.
2. Your project should be able to run under the workstations of NCTU CS (Arch Linux).
   a. **Test your project on workstations**. Use the NCTU CSCC account to login:
      i. tcglinux1.cs.nctu.edu.tw
      ii. tcglinux2.cs.nctu.edu.tw

iii. tcglinux3.cs.nctu.edu.tw
iv. tcglinux4.cs.nctu.edu.tw

b. Only run your project on workstations reserved for TCG (tcglinux). Do not occupied the normal workstations (linux1 ~ linux6), otherwise you will get banned.

Scoring Criteria:

1. Demo: **You need to demo your program in person**.
   a. The date and location will be announced later.
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.
   c. A **judge program** will be released later, you can test the solver before project due.
3. Penalty:
   a. Time limit exceeded (–30%): Slower than 1 minute 100 milliseconds.
   b. Late work (–30%): Late work including but not limited to **uncompilable sources** or **any modification** after due.

Hints:

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