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](https://www.crazygames.com/game/2584-fibonacci).

Overview: Solve *2584 Fibonacci*.

1. Modify the board size to .
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:
   1. Environment should drop **1-tiles** or **2-tiles** with probabilities of **0.9** and **0.1**, respectively.
   2. 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

1. The game tree of *2584 Fibonacci* is defined as
   1. The root node is an empty state.
   2. Each edge represents a legal action played by either environment or player.
   3. **The nth layer contains states that n actions have been applied**.
      1. 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.
      2. 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.
      3. Note that **nodes in layer 0 and 1** are neither before-states nor after-states.
2. The program should use **standard input** and **standard output**.
   1. Input: x t0 t1 t2 t3 t4 t5
      1. Input has several lines and ends with EOF. Each line contains a test case.
      2. Character x is the type of state, **will be either a (after-state) or b (before-state)**.
      3. Integers t0 ~ t5 represent the 2x3 board (in 1-d array form, 0 ≤ ti ≤ 33).
   2. Output: = v
      1. Output a single line for each test case.
      2. 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.
3. Transposition table is required.
   1. 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.
   2. Your program **should not use any pre-calculated external database**.
4. Implementation details:
   1. You program should be able to compile under the workstation of NCTU CS.
      1. Write a makefile (or CMake) for the project.
      2. 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.
   2. 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**.
   1. 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.
   1. Expand the game tree and save the result: ≤ 1 minute.
   2. Answer the questions: ≤ 1000 milliseconds (for 1000 test cases).
3. **Isomorphic states** in the same situation always have the same value.
   1. 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.
   1. The answers to “b 2 0 2 0 0 0” and “a 2 0 2 0 0 0” are the same.
   2. 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”.
   1. Root node is neither before-state nor after-state: “a 0 0 0 0 0 0”.
   2. Nodes in layer 1st are neither before-state nor after-state: “a 0 0 1 0 0 0”.
   3. 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).
2. Upload your **source files**, **makefiles**, and other relative files.
3. (Optional) Provide the Git repository of your project.
4. Your project should be able to run under the workstations of NCTU CS (Arch Linux).
   1. **Test your project on workstations**. Use the [NCTU CSCC account](https://www.cs.nctu.edu.tw/cchonor/faqs/2/%E5%B8%B3%E8%99%9F) to login:
5. tcglinux1.cs.nctu.edu.tw
6. tcglinux2.cs.nctu.edu.tw
7. tcglinux3.cs.nctu.edu.tw
8. tcglinux4.cs.nctu.edu.tw
   1. 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**.
   1. The date and location will be announced later.
2. Test cases (100 points): Pass all the test cases.
   1. 100 test cases, 1 point for each correct answer.
   2. See the attachment for sample input and output.
   3. A **judge program** will be released later, you can test the solver before project due.
3. Penalty:
   1. Time limit exceeded (–30%): Slower than 1 minute 100 milliseconds.
   2. 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.