Theory of Computer Games 2017 - Project 1

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: **Familiarize yourselves with *2584 Fibonacci***.

1. Implement the environment (game rules).
2. Implement the state container (array-based game board).
3. Build an AI based on some simple heuristics.

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 sequence of tiles in *2584 Fibonacci* is defined as
   0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, ... (-tile)
   with index value of
   0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, … (-index).
   Note that 2584-tile and 17-index are the different representations of the same value.
3. The position of grids in a game board are defined as

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

1-d array form

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

2-d array form

4. The implementation of board should contain following operations:
   a. Actions: Slide the board **up**, **right**, **down**, or **left**.
   b. Getter & Setter of grids: Provide read/write access of specific position.
5. The player should select actions based on **some simple heuristics**, where:
   a. Not required to be very strong.
   b. Not required to perform searching.
   c. The speed should be at least **100,000 actions per second** (time limit). (approximate value, see Scoring Criteria for details)
6. Statistic is required, and should include following measures:
   a. Average score.
   b. Maximum score.
   c. Speed (action per second).
   d. Win rate of each tiles.

7. 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. You program should recognize the following arguments:
        i. `--total=TOTAL_GAMES`: Indicate how many games to play.
        ii. `--play=ARGS_OF_PLAYER`: The arguments of player initialization.
        iii. `--evil=ARGS_OF_EVIL`: The arguments of evil (environment) initialization.
        iv. `--save=PATH_TO_SAVE_STAT`: Path to save statistic data.
    c. Statistic data should be saved as binary file, see Methodology for details.
        ```
        STAT = [TOTAL_RECS] [REC_1] ... [REC_R]
        REC = [TOTAL_ACTS] [ACT_1] ... [ACT_A] [TICK] [TOCK]
        ```

Methodology:

1. As a player, your program should calculate all the after-states (at most 4). **Determine the value of available after-states by heuristics**. Finally, select a proper action based on the values.
    a. You can design your heuristics by **the immediately reward**, **the number of empty space**, **the position of largest tiles**, **the monotonic decreasing structures**, etc.
    b. However, be careful to design with the number of children nodes, or something that requires searching.
2. **Sample program is provided**, which is a dummy AI that plays *2048*. You are allowed to modify everything (remember to follow the specification).
    a. 2048-game is treat as two-player game in the sample program.
        i. The evil (a.k.a. the environment) puts new tiles.
        ii. The player slides the board and merge the tiles.
    b. The process of 2048-game is designed as:
        i. A game begins with an empty board, the evil puts two tiles first.
        ii. Then, the player and the evil take turns to take action.
        iii. If the player is unable to find any action, the game terminated.
3. Statistic data should be saved as binary file, the format is:
    ```
    STAT = [TOTAL_RECS] [REC_1] ... [REC_R]
    REC = [TOTAL_ACTS] [ACT_1] ... [ACT_A] [TICK] [TOCK]
    ```
    a. `STAT`: The whole statistic file.
    b. `TOTAL_RECS`: Number of records, an 8-byte integer in little-endian, follow by `TOTAL_RECS` records (`REC`).
    c. `REC`: A record of games.
    d. `TOTAL_ACTS`: Number of actions, an 8-byte integer in little-endian, follow by `TOTAL_ACTS` actions (`ACT`).

Note that the environment and the player take turns in a record:
(initial) drop > drop > slide > drop > slide > … > drop > slide > drop (terminal)

e. `ACT`: An action, a 2-byte integer in little-endian, save in opcode:
   i. Slide: up, right, down, left with code `0, 1, 2, 3` respectively.
   ii. Drop new tile: should be `((k << 4) | (p))` for placing k-index at position p (1-d form).
f. `TICK`: The start time (milliseconds since epoch) of this record, an 8-byte integer in little-endian.
g. `TOCK`: The end time (milliseconds since epoch) of this record, an 8-byte integer in little-endian.

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. Do **not** upload the statistic output generated by programs.
   c. (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. Framework (85 points): Pass the statistic file test.
   a. A **sample test program** will be released later, you can test the statistic file by yourself before project due.
3. Average score (15 points): Calculated by $\min(\lceil \mathrm{Avg} \div 1000 \rceil, 15)$.
   a. Avg is the average score calculated in 1000 games (10 attempts).
4. (Bonus) Maximum tile: Calculated by $max(\mathrm{k} - 13, 0)$.
   a. k-index is the max tile calculated in 1000 games (10 attempts).
5. Penalty:
   a. Time limit exceeded (–30%): 100,000 is an approximate speed, your program should run faster than the **sample test program** mentioned above.
   b. Late work (–30%): Note that 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.

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