Theory of Computer Games 2021 – Project 1

Overview: **Familiarize yourselves with *2584 Fibonacci***, a 2048-like game.

1. Implement the framework of the game.
2. Implement the environment, i.e., the rules.
3. Develop a simple player based on some simple heuristics.

Specification:

1. 2584 Fibonacci is similar to 2048, and its basic rules are described as follows:
   a. The tiles are labeled by numbers in the Fibonacci sequence: **0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, ..., 3524578** (-tile), corresponding to index 0, 1, 2, 3, 4, 5, …., 32. Note that 0-tile indicates an empty cell, 3524578-tile is the maximum tile of the 4×4 board, and there is only one 1-tile in the definitions.
   b. Two **adjacent numbers in the Fibonacci sequence can be merged** to the next number, e.g., 1-tile + 1-tile = 2-tile; 1-tile + 2-tile = 3-tile; 13-tile + 21-tile = 34-tile. Note that tiles with the same value are not mergeable except for two 1-tiles.
   c. The environment drops **1-tiles** or **2-tiles** with probabilities of **0.9** and **0.1**, respectively.
   d. The distribution of the initial state (with two tiles) is equivalent to dropping two tiles (with probabilities mentioned above) on an empty board.
2. The positions of cells 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

3. The implementation of the board should contain the following operations:
   a. Actions: Slide the board **up**, **down**, **left**, or **right**.
   b. Getter & Setter of cells: Provide read/write access to a specific position.
4. 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 program speed should be at least **100000 actions per second**. (an approximate value, see Scoring Criteria for details)
5. The statistic is required, and should include the following measures:
   a. Average score.
   b. Maximum score.
   c. Speed (action per second).
   d. Win rate of each tile.

6. Other implementation requirements:
    a. The program should be able to execute in the Linux environment.
        i. C/C++ is highly recommended for TCG projects since the methods involved are sensitive to CPU speed. If you need to use other programming languages (e.g., Python) for the projects, contact TAs for more details.
        ii. The makefile (or CMake) for the program should be provided.
    b. The program should recognize the following arguments:
        i. `--total=TOTAL_GAMES`: The total number of episodes to be played.
        ii. `--play=ARGS_OF_PLAYER`: The arguments to initialize the player.
        iii. `--evil=ARGS_OF_EVIL`: The arguments to initialize the environment.
        iv. `--save=PATH_TO_SAVE_STAT`: Path to save the recorded episodes.
    c. The recorded episode should be able to be serialized as text as described below:
    `PLAYER:ENVIRONMENT@TICK|ACTIONS|WINNER@TOCK`
        i. `PLAYER`: The name of the player.
        ii. `ENVIRONMENT`: The name of the environment.
        iii. `WINNER`: The name of the winner.
        iv. `ACTIONS`: All actions in this episode.
        The environment and the player take turns in the `ACTIONS`:
        (initial) `PLACE` > `PLACE` > `SLIDE` > `PLACE` > … > `SLIDE` > `PLACE` *(terminal)*
        Each `PLACE` action and `SLIDE` action are represented in two characters. For `PLACE`, the first character presents the location of a new tile, by using hex numbers `0`, `1`, …, `F` corresponding to the 1-d index; the second character presents the tile value, which is either `1` or `2`. For `SLIDE`, the first character is `#`; the second character is one of `U`, `D`, `L`, `R` corresponding to the four directions. If the reward or the time usage (milliseconds) of action are not 0, the value should be also present after the action code by `[REWARD]` or `(TIME)`.
        v. `TICK`: The start time of this episode (milliseconds since the epoch).
        vi. `TOCK`: The end time of this episode (milliseconds since the epoch).

Methodology:

1. **The sample code is provided**, which is a framework for 2048-like games. You are allowed to modify everything under the specifications.
    a. The game of 2048 is treated as a two-player game in this framework.
        i. The evil (i.e., the environment) puts new tiles.
        ii. The player slides the board and merges the tiles.
    b. The process of a game (i.e., an episode) 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 turn to take action.
        iii. If the player is unable to find any action, the game is terminated.
2. Before developing the player, ensure that the environment of 2584 is correct.

3. The player should evaluate all the available afterstates (at most 4). **Determine the value of these afterstates by heuristics**. Finally, select a proper action based on the value.
    a. You can design the heuristics by **the immediate reward**, **the number of spaces**, **the position of the largest tiles**, **the monotonic decreasing structures**, etc.
    b. However, be careful to design with the number of children nodes or something that requires searching. It can easily lead to exceeding the time limit.

Scoring Criteria:

1. **Framework (85 points)**: Pass the judge to get 85 points.
    a. A **sample judge program** is provided, you should check the correctness of your work by yourself before the project is due.
2. **Average score (15 points)**: Calculated by $\min(\lceil \mathrm{Avg} \div 1000 \rceil, 15)$.
    a. Avg is the average score calculated in 1000 games.
3. **Maximum tile (8 points)**: Calculated by $\min(\max(\mathrm{k} - 13, 0), 8)$.
    a. k is the index of the maximum tile calculated in 1000 games. Note that 617-tiles (the 14th tile) or other larger tiles are necessary to get the points.
4. Penalties:
    a. **Time limit exceeded (–30%)**: 100000 is an approximate speed measured on our standard machine. You should check your work with the judge program.
    b. **Late work (–30%)**: Late work refers to any modification after the due date.
    c. **No version control (–30%)**: If it is found that there is no version control during the spot check, points will be deducted.
5. The final grade is the sum of the indicators minus the penalties, and the maximum is 100 points. However, if the program does not pass the judge, the final grade is 0.

Submission:

1. The submitted files **should be archived as a ZIP file** and **named `ID.zip`**, where **`ID`** is your student ID, e.g. `0356168.zip`.
    a. Pack your **source files**, **makefiles**, and other relative files.
    b. Submit the archive through the E3 platform.
    c. Do not upload the version control hidden folder, e.g., the `.git` folder.
2. The program **should be able to run under the provided Linux workstations**.
    a. Available hosts: tcglinux1.cs.nctu.edu.tw, tcglinux2.cs.nctu.edu.tw, …, tcglinux10.cs.nctu.edu.tw (more information will be announced on the E3 platform)
        i. Use the NYCU CSIT account to log in via SSH.
    b. The projects will be graded on the provided workstations.
        i. You may use your machine for development.
        ii. If the program cannot be compiled or executed on our workstations without additional modifications, the project may be regarded as late work.
    c. Do not occupy the workstations. Contact TAs if the workstations are crowded.
3. Version control (e.g., Github or Bitbucket) is required during the development.

Appendix:

1. An example of the statistic output of the sample program:

```
1000   avg = 2003, max = 7522, ops = 398783 (486843|381955)
       21     100%   (0.7%)
       34     99.3%  (1.8%)
       55     97.5%  (11.7%)
       89     85.8%  (19.7%)
       144    66.1%  (30.4%)
       233    35.7%  (27%)
       377    8.7%   (7.9%)
       610    0.8%   (0.8%)
```

   Description of indicators:

   `1000`: the statistic is for the 1st episode to the 1000th episode.

   `avg = 2003`: the average score of these episodes.

   `max = 7522`: the maximum score of these episodes.

   `ops = 398783`: the speed of the program measured from playing these episodes.

   `233 35.7% (27%)`: 35.7% of episodes reached 233-tile, and 27% ended with 233-tile as the maximum.

2. An example of a saved episode (environment's placing; player's sliding):

```
dummy:random@1537878040221|11D1(1)#L(2)61(3)#D[4](3)11(2)#D(3)B1(3)#L[4](3)61#R
[8](2)D1(2)#D[4]A1(3)#D31(3)#L[4]A1#U[8](3)E1(2)#D[4](3)11(2)#R[16](3)71(2)#L21
#L[4]31(2)#R41(3)#R[4]A1(2)#L[4](3)B1(3)#U[8]71(2)#U[4](2)62(3)#L[8](2)91(3)#D[
16](3)11#D[36](2)81(3)#L71#UC1#L[12](2)91(1)#U(2)D1(3)#L[4](2)31(1)#D(2)31#R[4]
01(2)#U[8](3)52(3)#D[8]91(2)#L(2)71(2)#D[4]61(2)#L[4]71(3)#D[4](2)71(2)#R(2)21(
1)#D[8](3)11(2)#L[4]61(2)#L71(3)#L[4]62(1)#U[8](1)F1(2)#D[16](1)11(2)#R[28](2)1
2#R(1)11(2)#L(3)71#U(2)B1(3)#R[4](2)01(3)#R[12](2)C2(3)#U[8]91#D(2)11(3)#R[20](
2)11(3)#L[4](2)F1(3)#U(2)B1#D[4](1)21#U(2)F1(3)#U(3)F1(3)#U[4]F1(2)#U[8](2)B1#U
[4](2)F1(1)#D(1)31#R[12](3)01(2)#D[24](1)11#R(3)02(2)#L(2)31(2)|random@15378780
40433
```

3. The reaching rate of each tile when using the two baseline methods:

Max Tile	Random (AVG = 1970)	1-ply Greedy (AVG = 6197)
8	100.0%	100.0%
13	100.0%	100.0%
21	99.9%	100.0%
34	98.6%	100.0%
55	96.1%	100.0%
89	85.6%	99.7%
144	65.6%	98.1%
233	35.0%	88.9%
377	8.4%	66.5%
610	0.5%	32.1%
987	0.0%	6.9%
1597	0.0%	0.4%