

Theory of Computer Games 2019 - Project 1

Overview: **Familiarize yourselves with *Threes!***

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 [1] [2], except for:
 - a. The **bag size of new tiles is 3**.
 - b. The environment should **generate all tiles (including the initial 9 tiles) from the bag**.
 - c. The new tile should be randomly placed **at any empty cell on the opposite side of the last sliding direction**.
 - d. **No bonus tiles**.
2. The sequence of tiles in *Threes!* is defined as
0, 1, 2, 3, 6, 12, 24, 48, 96, 192, 384, 768, 1536, 3072, 6144 (-tile)
with index value of
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 (-index).
Note that 6144-tile and 14-index are the different representations of the same value.
3. The implementation of board should contain following operations:
 - a. Getter & Setter of tiles: Provide read/write access of tiles.
 - b. Player's actions: Slide the board **up, right, down, or left**.
 - c. Environment's actions: Generate the new tiles with bag rule.
4. Statistic is required, each line represents an episode:
`PLAYER:ENVIRONMENT@TICK|ACTIONS|WINNER@TOCK`
(see Appendix for an example)
 - a. `PLAYER`: The name of the player.
 - b. `ENVIRONMENT`: The name of the environment.
 - c. `WINNER`: The name of the winner.
 - d. `ACTIONS`: All actions in this episode.
Note that 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.
If the reward and the time usage of an action is not 0, the value should be present after the action code with `[REWARD]` or `(TIME)`:
 - e. `TICK`: The start time of this episode.
 - f. `TOCK`: The end time of this episode.
5. The player should select actions based on **some simple heuristics**.
6. Speed requirement for the player program:
 - a. For C/C++ is **200,000** actions per second.
 - b. For Python is **2,000** actions per second.

- c. For any other programming languages, please contact TAs.
- 7. 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 statistic output format.
(see the Methodology for details)
- 8. Version control (Git) is necessary, use [Github Student Developer Pack](#) or [Bitbucket](#) for the version control.

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 spaces, the position of largest tiles, the monotonic decreasing structures**, etc.
 - b. Your program in Project 1 is not required to be very strong.
 - c. However, be careful to design with the number of children nodes, or something that requires searching.
2. **Sample code is provided**, which is a dummy AI that plays 2048. You are allowed to modify everything (however, remember to follow the I/O specifications).
 - a. 2048-game is treated as two-player game in the sample program.
 - i. The environment places new tiles.
 - ii. The player slides the board to merge the tiles.
 - b. The process of 2048-game is designed as:
 - i. A game begins with an empty board, the environment puts two tiles first.
 - ii. Then, the player and the environment take turns to make action.
 - iii. If the player is unable to find any action, the game terminated.

Submission:

1. Your solution **should be archived in zip/rar/7z file**, and **named as XXXXXXXX.zip**, where XXXXXXXX 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 executable, 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 TCG workstations of NCTU CS (Arch Linux).
 - a. **Test your project on the workstations.**
Login info: TBA.
 - b. **Your work will be graded on the workstations.**

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

Scoring Criteria:

1. Framework (85 points): Pass the statistic file test.
 - a. You can test the statistic file by yourself before project due with the **judge program**, check the e3 discussion for details.
2. Average score (10 points): Calculated by $\min\left(\log_2\left(\frac{AVG}{3}\right) + 1, 10\right)$.
 - a. AVG is the average score calculated in 1000 episodes.
3. Maximum tile (5 points): Calculated by $\max(k - 9, 0)$.
 - a. k -index is the max tile calculated in 1000 episodes.
4. Penalty:
 - a. Time limit exceeded (−30%):
 - i. Speed requirement for C/C++ is **200,000** actions per second.
 - ii. Speed requirement for Python is **2,000** actions per second.
 - iii. For other programming languages, please contact TAs.
 - b. Late work (−30% per week): Note that late work including but not limited to **uncompilable sources** or **any modification** after due.
 - c. No version control (−10%).
 - d. Bad submission (−10%).

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. An example of the record of 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)12#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@1537878040433
```

2. The position of cells in a game board are defined in 1-d array form.

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

1-d array form

3. Statistic of a million episodes of Threes!:

Random Play				
tile	score	move	rate	win
3	21	8	0%	100%
6	44	14	1%	100%
12	94	23	14%	99%
24	193	36	45%	85%
48	408	55	35%	40%
96	929	82	5%	5%
192	2404	132	0%	0%

Greedy Play				
tile	score	move	rate	win
6	47	14	0%	100%
12	106	25	5%	100%
24	234	42	25%	95%
48	520	68	41%	70%
96	1173	106	25%	29%
192	2804	166	4%	4%
384	7265	266	0%	0%

Hints:

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.