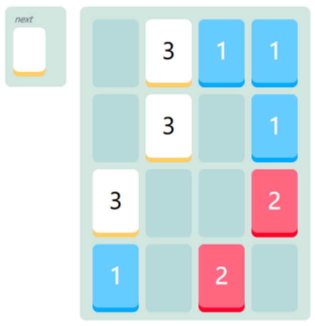
Theory of Computer Games 2022 – Project 1



Overview: **Familiarize yourselves with the simplified *Threes!***
1. Implement the framework of the game.
2. Implement the environment with simplified rules.
3. **Develop a simple player based on simple heuristics**.

Specification:
1. The tiles in *Threes!* are labeled as ***0, 1, 2, 3, 6, 12, 24, 48, 96, 192, 384, 768, 1536, 3072, 6144*** (-tile), corresponding to index number 0, 1, 2, …, 12, 13, 14, respectively.
   Note that 0-tile indicates an empty cell, and 6144-tile is designed as the maximum tile.
2. The rules follow the original [1] [2] with simplified tile generation as follows.
   a. **No bonus tiles**. All tiles (including the initial tiles) are generated **from a bag size 3**.
   b. A non-initial new tile is randomly placed **at an empty cell on the puzzle border of the opposite side of the last sliding direction**.
   For example, the new tile is randomly placed at an empty cell selected from position 12, 13, 14, and 15 if the last sliding direction is up.
3. The board should provide the basic operations.
   a. Actions: Slide the board **up**, **down**, **left**, or **right**.
   b. Getter & Setter of cells: Provide read/write access to a specific position and the hint tile.

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

position index

4. The player should select actions based on **simple heuristics**.
   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 statistics are required and should include measures as follows.
   a. Average score.
   b. Maximum score.
   c. Speed (action per second).
   d. Win rate of each tile.
6. The program should be able to execute in the Linux environment.
   a. C++ is highly recommended for TCG projects since the methods involved are sensitive to CPU speed.
   b. For other programming languages (e.g., Python), contact TAs for more details.
   c. A makefile for the program should be provided.
      i. Provide `make` for compiling the program.
      ii. Provide `make stats` for executing the program, generating statistics of 1000 games and saving the results into a file named `stats.txt`.

7. The program should recognize the arguments as follows.
   a. `--total=NUM_GAMES`: The total number of episodes to be played.
   b. `--slide=SLIDER_ARGS`: The arguments to initialize the player.
   c. `--place=PLACER_ARGS`: The arguments to initialize the environment.
   d. `--save=STATS_PATH`: Path to save the recorded episodes.
8. The recorded episodes should be serialized as text in the format described as follows.
   `SLIDER:PLACER@TICK|ACTIONS|WINNER@TOCK`
   a. `SLIDER`: The name of the player.
   b. `PLACER`: The name of the environment.
   c. `WINNER`: The name of the winner.
   d. `ACTIONS`: All actions in this episode; two types of actions: `PLACE` and `SLIDE`.
      i. A `PLACE` action consists of three characters: the first character represents the location of the new tile; the second character represents the value of the new tile; the third character represents the value of the hint tile.
      ii. A `SLIDE` action consists of two characters: the first character is `#`; the second character represents the sliding direction.
      iii. If the reward or the time usage (milliseconds) of this action is not 0, the value is also recorded after the action code by `[REWARD]` or `(TIME)`.
   e. `TICK`: The start time of this episode (milliseconds since the epoch).
   f. `TOCK`: The end time of this episode (milliseconds since the epoch).

Methodology:

1. **The sample code has implemented the required rules of the simplified Threes!**. However, you are allowed to modify everything under the specifications.
   a. The game is organized as a two-player game in this framework.
      i. The environment (the placer) puts new tiles and provides hints.
      ii. The player (the slider) slides the puzzle and merges the tiles.
   b. The process of a game (an episode) is designed as:
      i. A game begins with an empty board; the environment puts nine tiles first.
      ii. Then, the player and the environment take turns taking action.
      iii. If the player has no legal actions to play, the game is over.
2. The player should **determine the value of the available afterstates by heuristics**. Then select a proper action based on the value.
   a. You can design the heuristics by **the puzzle value**, **the number of spaces**, **the position of the largest tiles**, **the monotonic decreasing structures**, etc.
      i. The official value is the sum of $3^{1+\log_2(v/3)}$ of all v-tiles with $v \geq 3$ on the puzzle. Note that Threes! has no immediate reward.
      ii. You may design a unique calculation method for the heuristic.
   b. However, be careful to design with the number of children nodes or something that requires searching. It can lead to exceeding the time limit.

Scoring Criteria:

1. **Performance (100 points)**: Calculated by $\text{round}\left(\frac{40}{1+\exp(-0.0065\times(\text{AVG}-560))}+60\right)$.
   a. AVG is the average score of 1000 games by using the official calculation method, i.e., the sum of $3^{1+\log_2(v/3)}$ of all v-tiles with $v \geq 3$ on the puzzle when the game ends.
   b. A **judge program** is provided to assess the grade.
      i. First, play 1000 games and write the statistics to stats.txt by your program:
      ```
      $ ./threes --total 1000 --save stats.txt --OTHER_ARGS
      ```
      ii. To judge the statistics, load it by the judge:
      ```
      $ ./threes-judge --load stats.txt
      ```
2. **Report (10 points, optional)**: Graded according to the completeness of the report.
3. Penalties:
   a. **Time limit exceeded (–30%)**: If the program speed does not meet the minimum speed expected by the judge program.
   b. **Late submission (–30%)**: If the project requires any modifications after the deadline.
   c. **No version control (–30%)**: If it is found that there is no version control during the spot check.
4. The final grade is the sum of the indicators minus the penalties.
   a. **The maximum grade is limited to 100 points**.

Submission:

1. The submission **should be archived as a ZIP file** and **named `ID.zip`**, where **`ID`** is your student ID, e.g., `0356168.zip`.
   a. Pack the **source files**, **makefile**, **report**, and other required 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.nycu.edu.tw, …, tcglinux10.cs.nycu.edu.tw
      i. Use the NYCU CSIT account to log in via SSH.
      ii. Place project files in `/tcgdisk/ID`, where ID is your student ID. Note that you need to create the folder first. For example, suppose that your ID is 0356168:
      ```
      $ mkdir /tcgdisk/0356168 && chmod 700 /tcgdisk/0356168
      ```
   b. The projects will be graded on the provided workstations.
      i. You may use your machine for development. The judge program should work on most Linux platforms.
   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.

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/