Theory of Computer Games 2017 - Project 2

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: **Write a player to play *2584 Fibonacci* with high win rates**.

1. Implement the function approximator with n-tuple network.
2. Implement TD(0) after-state backward training method.
3. Build an AI based on n-tuple network and TD learning (no search is required).

Specification:

1. The environment and the rules are the same as Project 1's.
2. Train the agent by temporal difference learning.
3. The **testing speed** should be at least **50,000 actions per second** (time limit). (approximate value, see Scoring Criteria for details)
4. Statistic is required, and the requirement is the same as Project 1's requirement.
   a. Average score.
   b. Maximum score.
   c. Speed (action per second).
   d. Win rate of each tiles.
5. 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
6. Your program should be able to serialize and deserialize the weight tables of N-tuple network.

Methodology:

1. As a player, your program should calculate all the after-states (at most 4). **Determine the value of available after-states by the N-tuple network**. Finally, select a proper action based on the values.
   a. Design and **implement the N-tuple network**.
   b. States and actions should be saved during an episode.
   c. After the episode done, **train the N-tuple network by TD(0) method**.
2. Some useful advices for designing the N-tuple network of *2584 Fibonacci*:

   a. Larger networks need more time to converge.
   b. You may need to **merge** some index in order to avoid the size of network being too large. In *2048*, the size of a 6-tuple is usually designed as 16^6 (64MB). The maximum tile of *2584 Fibonacci* (under our environment) is much larger than 16. Take 24-index as an example, the size would be 24^6 (729MB).
   c. Isomorphic patterns are able to speed up the training process.
   d. The initial value of weight tables should be set as 0.
3. Some useful advices for training:
   a. Do not forget to train **the terminal state**. Its TD target should be 0.
   b. Do not forget to take **the immediately reward** into account.
   c. You should train the network based on "**after-state**", not on "before-state".
   d. The learning rate is based on the number of features. For example, the learning rate could be $0.1 \div 32 = 0.003125$ if you have four N-tuples with 8 isomorphic patterns (32 features).
   e. Reduce the learning rate if the network is converged.
4. **Sample program is provided**, which is a non-implement 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.
   c. In this project, the player needs some arguments to initialize. You could define and use them in `ARGS_OF_PLAYER` as your needed.

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. Do **not** upload the weight tables generated by programs.
   d. (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 occupy the normal workstations (linux1 ~ linux6), otherwise you will get banned.
   c. Respect the rights of others. Do not occupied the resources of workstation.

Scoring Criteria:

1. Demo: **You need to demo your program in person**.
   a. The date and location will be announced later.
2. Framework: Pass the statistic file test.
   a. A **sample test program** which is similar to HW1's will be released later, you can test the statistic file by yourself before project due.
   b. **Your score is not counted if the statistic file cannot pass the judge**.
3. Win rate of 2584-tile (100 points): Calculated by $\lceil \mathrm{WinRate}_{2584} \rceil$.
   a. $\mathrm{WinRate}_{2584}$ is the win rate of 2584-tile calculated in 1000 games (10 attempts).
4. (Bonus) Maximum tile: Calculated by $max(\mathrm{k} - 17, 0)$.
   a. k-index is the max tile calculated in 1000 games (10 attempts).
5. Penalty:
   a. Time limit exceeded (–30%): 50,000 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.
   c. **Updating the weight tables will NOT be considered as late work**. You can keep training the weights if your program uploaded before the deadline is able to read the newer weight tables.

References:

[1] Szubert, Marcin, and Wojciech Jaśkowski. "Temporal difference learning of N-tuple networks for the game 2048." 2014 IEEE Conference on Computational Intelligence and Games. IEEE, 2014.

[2] Kun-Hao Yeh, I-Chen Wu, Chu-Hsuan Hsueh, Chia-Chuan Chang, Chao-Chin Liang, and Han Chiang, Multi-Stage Temporal Difference Learning for 2048-like Games, accepted by IEEE Transactions on Computational Intelligence and AI in Games (SCI), doi: 10.1109/TCIAIG.2016.2593710, 2016.

[3] Oka, Kazuto, and Kiminori Matsuzaki. "Systematic selection of n-tuple networks for 2048." International Conference on Computers and Games. Springer International Publishing, 2016.

Hints:

You are NOT required to use the framework of project 2 since the it only provides the weight table and some hints (you can keep working on project 1's framework).
Version control (GitHub, Bitbucket, Git, SVN, etc.) is recommended but not required.
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.