Theory of Computer Games 2017 - Project 4

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: **Retrain the player under different environment and increase the win rate**.

1. Modify the environment and retrain the network.
2. (Optional) Build an AI based on TD learning and expectimax search.
3. (Optional) Improve the state container, replace the array-board with bitboard.
4. (Optional) Implement TCL or TD(λ) to speed up the training process.

Specification:

1. The rules are similar to the one of Project 2 with the following change:
   a. Environment should drop **1-tiles** or **3-tiles** with probabilities of **0.75** and **0.25**, respectively.
2. Train the agent by temporal difference learning.
3. The **testing speed** should be at least **10,000 actions per second** (time limit). (approximate value, see Scoring Criteria for details)
4. Statistic is required, and the requirement is the same as Project 2's requirement.
   a. Average score.
   b. Maximum score.
   c. Speed (action per second).
   d. Win rate of each tiles.
5. Implementation details (the same as Project 2's spec):
   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. Some useful advices for retraining your program under a harder environment:
   a. Use **larger n-tuple networks** to improve the final performance.
   b. Use **expectimax search** to improve the testing.
   c. Use **bitboard** to improve the runtime performance.

   d. Use **Temporal Coherence Learning** (TCL) to speedup the training.
   e. Use **TD(λ)** to speedup the training.
2. For those who want to use a larger n-tuple network [2] [3]:
   a. Larger networks need more time to converge.
   b. You may need to merge some index to avoid the size of network being too large.
   c. Isomorphic patterns are able to speed up the training process.
3. For those who want to use expectimax search:
   a. **Expectimax search is costly**.
   b. You may search deeper if you have time, while 3-layer search is usually enough.
   c. Note that your previous implementation in Project 2 is defined as 1-layer search.
   d. You are unable to stop the search at a layer of before-states.
4. For those who want to use bitboard [5]:
   a. There are some implementations of 2048's bitboard on GitHub.
   b. Note that **64-bit (4-bit each tile) is too small** for 2584 states.
5. For those who want to use TCL [4]:
   a. Note that TCL will **triple the memory usage** during the training.
   b. Your network may converge very fast and unable to escape the local optima.
6. For those who want to use TD(λ) [2] or even TC(λ) [4]:
   a. There are many variants of TD(λ), be careful with the update rules.
   b. λ = 0.5 in most cases.

Submission:

1. **Archive your solution in a zip file**, and **name it 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. However, do **not** upload the ".git" folder.
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**.
   a. The details will be announced later.
2. Framework: Pass the statistic file test.

   a. A **judge program** will be released. You can judge the statistic file 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 **100 games** (10 attempts).
4. Maximum tile (Bonus): Calculated by $max(\mathrm{k} - 17, 0)$.
   a. k-index is the max tile calculated in 100 games (10 attempts).
5. Penalty:
   a. Time limit exceeded (–30%): 10,000 is an approximate speed, your program should run faster than the **sample program**.
   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.

[4] Jaskowski, Wojciech. "Mastering 2048 with Delayed Temporal Coherence Learning, Multi-Stage Weight Promotion, Redundant Encoding and Carousel Shaping." IEEE Transactions on Computational Intelligence and AI in Games (2017).

[5] https://github.com/moporgic/TDL2048-Demo

Hints:

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.