Theory of Computer Games 2018 - Project 4

In the series of projects, you are required to develop AI programs that play *Threes!*, the origin of other 2048-like games.

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

1. Retrain the player under a different environment.
2. Write a player with TD learning and expectimax search.
3. Try larger n-tuple networks to improve the final performance.
4. Try advanced TD methods, such as TC or TD(λ) to speed up the training process.

Specification:

1. The rules follow the original rules **[1]** [2], with some small modifications:
   a. **The bag size of new tiles is 12**.
      i. The initial 9 tiles should also be generated from the bag.
   b. **The bonus random tile** should be generated with the **probability of 1/21** when the largest tile is at least a 48-tile.
      i. The bonus tile ranges from a 6-tile to a $\frac{v_{\max}}{8}$-tile with equal probability, where $v_{\max}$ is the largest tile on the current board.
      ii. While other tiles from the bag are generated with 20/21.
      iii. **At any timestamp of an episode**, the ratio of the number of bonus tile and the total generated tiles **should not more than 1/21**, i.e. $\frac{\#\text{ of bonus tiles}}{\#\text{ of total generated tiles}} \leq \frac{1}{21}$.
2. Train the agent by temporal difference learning.
3. 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.
4. Speed requirement:
   a. For C/C++ is **10,000** actions per second.
   b. For Python is **100** actions per second.
   c. For any other programming languages, please contact TAs.
5. Implementation details (the same as Project 2's spec):
   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 Project 2's spec for details)
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.
      i. Try some complex configurations, e.g. use 6-tuple instead of 4-tuple.
      ii. Add hint tile and last action into the network design.
      iii. Try multi-stage temporal difference learning (MS-TD) if one network is not enough [4].
   b. Use **expectimax search** to improve the decision of player.
   c. Decrease the learning rate manually.
      i. Use **Temporal Coherence Learning** (TC) for automatic learning rate decay.
   d. Use **bitboard** or pre-calculated move table to improve the runtime performance.
   e. Use **TD(λ)** to speedup the training.
2. For those who want to use a larger n-tuple network [4] [5]:
   a. Larger networks need more time to converge.
   b. Isomorphic patterns are able to speed up the training process.
   c. If you have not added hint tile or last action into the network, add them first.
   d. Parallel training may be necessary when the network is really large.
   e. MS-TD improves the performance at the cost of memory and training time [4]; however, the implementation and training procedure are much more complicated than ordinary TD. Be careful if you really want to implement it.
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 TCL [6]:
   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.
5. For those who want to use bitboard [7]:
   a. There are some implementations of 2048's bitboard on GitHub.
   b. Depending on your implementation, bitboard **may not** greatly improve the speed.
6. For those who want to use TD(λ) [4] or even TC(λ) [6]:
   a. There are many variants of TD(λ), be careful with the update rules.
   b. λ = 0.5 in most cases.

Submission:

1. Your solution **should be archived in zip/rar/7z file**, and **named as `XXXXXXX.zip`**, where XXXXXXX 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 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 workstations of NCTU CS.
   a. **Test your project on workstations**.
   b. Only run your project on workstations reserved for TCG. Do not occupied the normal workstations, otherwise you will get banned.
   c. Respect the rights of others. Do not occupied the resources of workstation.

Scoring Criteria:

1. Demo: **TBD**.
2. Framework: Pass the statistic file test.
   a. A **judger** which is similar to HW2'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 384-tile (100 points): Calculated by $\lceil \mathrm{WinRate}_{384} \rceil$.
   a. $\mathrm{WinRate}_{384}$ is the win rate of 384-tile calculated in 1000 games.
4. Maximum tile (5 points): Calculated by $\max(k-9,\ 0)$.
   a. k-index is the max tile calculated in 1000 episodes.
5. Penalty:
   a. Time limit exceeded (–30%):
      i. For C/C++ is **10,000** actions per second.
      ii. For Python is **100** actions per second.
      iii. For any other programming languages, please contact TAs.
   b. Late work (–30%): Note that late work including but not limited to **uncompilable sources** or **any modification** after due.
   c. No version control (–30%).
   d. **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] 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/

[3] 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.

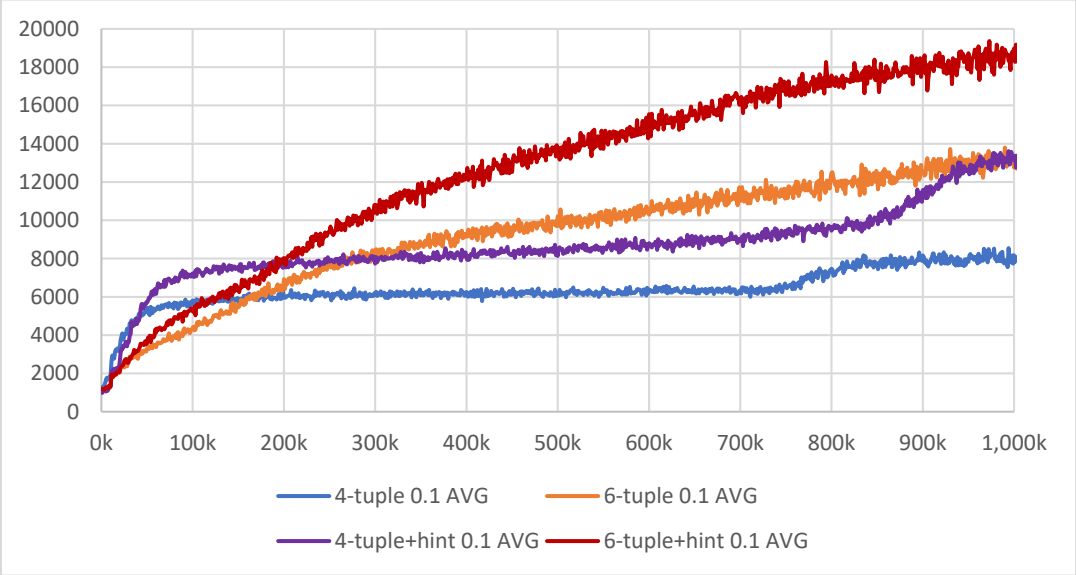
[4] Kun-Hao Yeh, I-Chen Wu, Chu-Hsuan Hsueh, Chia-Chuan Chang, Chao-Chin Liang, and Han Chiang, Multi-Stage Temporal Difference Learning for 2048-like Games, accepted by IEEE Transactions on Computational Intelligence and AI in Games (SCI), doi: 10.1109/TCIAIG.2016.2593710, 2016.

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

[6] 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).

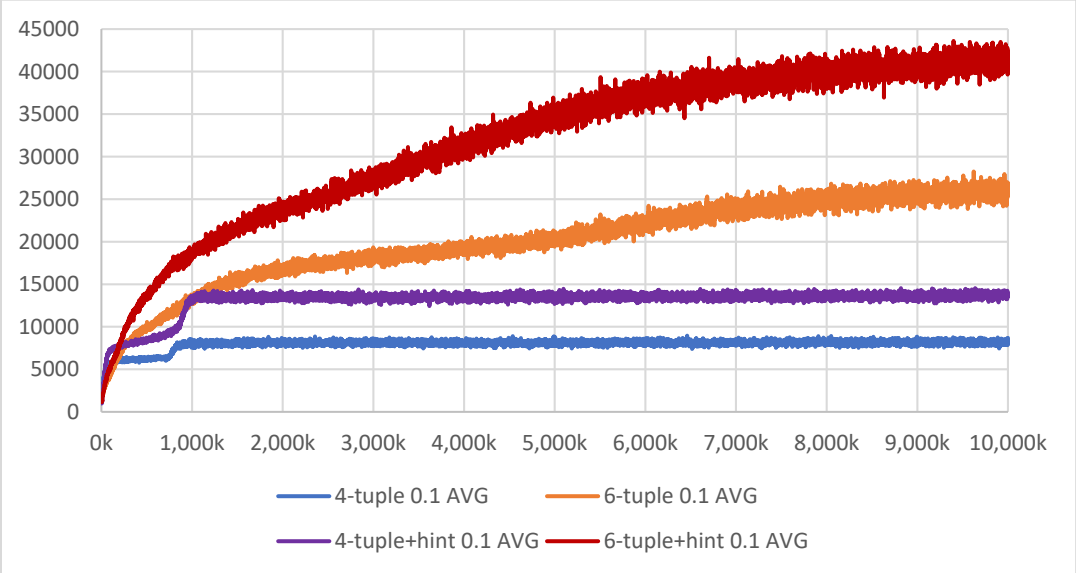
Appendix:

1. Average score during the training of a million episodes:



Final $\mathrm{WinRate}_{384}$ after training: **60.3% 81.9% 80.9% 88.3%**

2. Average score during the training of 10 million episodes:



Final $\mathrm{WinRate}_{384}$ after training: **60.7% 80.8% 85.8% 94.9%**

Hints:

You are NOT required to use the framework of project 4 since it is the same as project 2's (you can keep working on project 2's framework, however, remember to create some branches).

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.

Remember to share the sources on sharing platform, for example, GitHub Gist.