Theory of Computer Games 2017 - Project 5

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 an evil program**.

1. Design an evil (environment) agent for selecting the position of new tile.
2. Implement the arena protocol and connect the agents to the *2584 Fibonacci* arena.
3. (Optional) Implement alpha-beta search or Monte-Carlo tree search.

Specification:

1. The rules are similar to the one of Project 4 with the following change:
   a. The environment should decide the **position** of new tile.
   b. The 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. Your 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. Your 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. Implementation details (**the arena protocol**):
   a. Your program should recognize the following input arena messages:
      i. `match ID open AGENT:AGENT`
      ii. `match ID [play/evil] take turn`
      iii. `match ID [play/evil] move ACTION_CODE`
      iv. `match ID close AGENT`
   b. Your program should use the following output arena messages:
      i. `match ID [play/evil] ready`

ii. `match ID [play/evil] move ACTION_CODE`

c. Please refer to the sample code for details.

7. Your program should be able to serialize and deserialize the weight tables of N-tuple network.

Methodology:

1. Start a 2-layer **minimax search at the after-state**, and select the action (the position of new tile) which has the minimum value.
   a. Try the naïve minimax search first.
   b. Try alpha-beta search if everything goes well.
   c. Reuse the weights trained for normal player in Project 4.
2. The arena protocol has already been implemented in the sample code.
   a. While you may need to modify some lines.

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 $\lceil 100 - \mathrm{WinRate}_{2584} \rceil$.
   a. The environment agent needs to play 100 games with **a collection of players**.
   b. $\mathrm{WinRate}_{2584}$ is the win rate of 2584-tile calculated by the players in 100 games.
4. Maximum tile (Bonus): Calculated by $max(17 - \mathrm{k}, 0)$.
   a. k-index is the max tile of players calculated in 100 games mentioned above.

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.

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.