Theory of Computer Games 2021 – Project 4

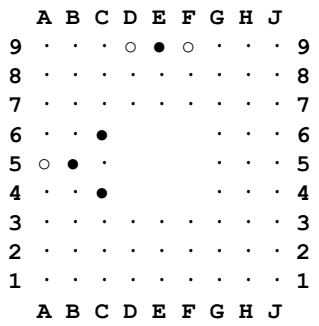
Overview: **Write an MCTS program to play *Hollow NoGo*, a variant of NoGo.**

1. Implement a Hollow NoGo framework on the 9x9 board.
2. Design a Hollow NoGo player with Monte Carlo tree search (MCTS).
3. Use Go Text Protocol (GTP) to communicate with other programs.

Specification:

1. Generally speaking, 9x9 Hollow NoGo has the same rules as 9x9 NoGo. The only difference is that the center 3x3 board is cut out, which can not be counted as liberty.
2. Two players (black and white) alternatively take turns to play. The same as in NoGo, **capturing, suicide, and pass are forbidden**. The player who has no action to play loses the game.
   a. Capturing: take the last liberty of an opponent block.
   b. Suicide: create a block that has no liberty.
   c. Pass: skip this turn without placing a stone.

   For example, in the right figure, the white player cannot play at E8 (capturing), C5 (suicide), E5 (illegal position).



3. The player should take actions based on MCTS and should be able to play as both sides.
   a. **The thinking time of the player is 36 seconds for a game**, i.e., for the black player, the total time spent on black stones in a game should not exceed 36 seconds.
4. The Go Text Protocol (GTP) should be implemented with standard input/output.
5. Other implementation requirements:
   a. The program should be able to execute in the Linux environment.
      i. C/C++ is highly recommended for TCG projects since the methods involved are sensitive to CPU speed. If you need to use other programming languages (e.g., Python) for the projects, contact TAs for more details.
      ii. The makefile (or CMake) for the program should be provided.

Methodology:

1. **The framework of Hollow NoGo with GTP is provided**.
   a. It is similar to the 2048 framework, you should be able to get familiar with it quickly.
   b. It is recommended to follow the instructions below step by step.
2. **Make sure that you are familiar with the MCTS**. Think carefully about every implementation detail. You should implement the simplest version first.
   a. Use fixed simulation count, e.g., a total number of 100 MCTS cycles.
   b. At the selection phase, just use a simple UCB formula. Keep selecting child nodes until a leaf node, i.e., fully expanded node, is reached.
   c. At the expansion phase, expand the child node in random order. You may expand all the child nodes together, but only one child node is visited and updated, the other

child nodes are still unvisited (with visit count = 0). When the next time this node is selected, an unvisited child will be selected due to its highest UCB value.

   d. At the simulation phase, rollout by the random policy. Just place stones randomly until the end of the game and obtain the result. If the node is already a terminal state, just return its result.
   e. After the search, select the best action based on the visit count of child nodes.
3. With a correct MCTS implementation, the player with a few simulation counts (e.g., 100) should be able to outperform the random player, with a win rate of more than 90%.
4. You may change the fixed simulation count to timestamp measurement.
   a. Try 1 move per second first. It is generally safe since it is almost impossible for a program to play a total of 36 moves in a game.
5. Finally, try more improvements, e.g., more UCB variants, RAVE, heuristic rollout, move ordering, time management, parallel MCTS, etc.

Scoring Criteria:

1. **Win rate against a weak player (70 points)**: Calculated by $\lceil \mathrm{WinRate}_{\mathrm{weak}} \times 70\% \rceil$.
   a. $\mathrm{WinRate}_{\mathrm{weak}}$ is the win rate against a weak sample player in 70 games.
   Your program will be black in 35 games, and be white in another 35 games.
   b. **The weak sample player is provided**, you should check the correctness of your work by the sample program and the provided scripts before the deadline.
2. **Win rate against stronger players (30 points)**: Calculated by $\lceil \mathrm{WinRate}_{\mathrm{strong}} \times 30\% \rceil$.
   a. $\mathrm{WinRate}_{\mathrm{strong}}$ is the win rate against stronger sample players in 30 games.
   Your program will be black in 15 games, and be white in another 15 games.
   b. **The stronger players are not provided**.
3. **Report (10 points, optional)**: Graded according to the completeness of the report.
   a. Summarize the used methods, improvements, and so on.
4. Penalties:
   a. **Time limit exceeded (lose that game)**.
   b. **Illegal action (lose that game)**.
   c. **Late work (–30%)**: Late work refers to any modification after the due date.
   d. **No version control (–30%)**: If it is found that there is no version control during the spot check, points will be deducted.
5. The final grade is the sum of the indicators minus the penalties, up to a maximum of 100 points.
   a. Note that the report is optional. **You can choose NOT to submit a report**.

Submission:

1. The submitted files **should be archived as a ZIP file** and **named `ID.zip`**, where **`ID`** is your student ID, e.g. `0356168.zip`.
   a. Pack your **report**, **source files**, **makefiles**, and other relative 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.nctu.edu.tw, tcglinux2.cs.nctu.edu.tw, …, tcglinux10.cs.nctu.edu.tw (more information will be announced on the E3 platform)
      i. Use the NYCU CSIT account to log in via SSH.
   b. The projects will be graded on the provided workstations.
      i. You may use your machine for development.
      ii. There is no GPU on these machines. **If you implement advanced methods such as AlphaZero that require GPU resources, contact TAs**.
      iii. If the program cannot be compiled or executed on our workstations without additional modifications, the project may be regarded as late work.
   c. Do not occupy the workstations. Contact TAs if the workstations are crowded.
      i. Each workstation has 4 threads and 7.8GB of memory, which can be shared by 3–4 people. **Pay attention to the resources used by your program**.
3. Version control (e.g., Github or Bitbucket) is required during the development.