CONTENTS


- Assignment Overview
  - **New Feb 17:** Assignment 2 Update
- Setup
- Goals of the Assignment
  - timelimit
  - solve
  - Modify genmove to use solver
- Recommended: Test Matches
- Public Test Cases
- Other GTP Commands
- Submission Checklist
- Do Your Own Pre-submission Test
- Marking
- Notes


# Cmput 496 Assignment 2

Due **Feb 26, 11:55pm**. Submit via eClass submission link on the main eClass course page.
Late submission (with 20% deduction) Feb 28, 11:55pm. Submit via separate eClass link for late submissions.

**New Feb 17:** Assignment 2 Update

In this assignment, you develop a perfect endgame solver for the game of Go, based on our starter program `Go2`. You also integrate your solver into the `Go2` player. This player will play randomly at first, but will then play a perfect endgame as soon as it can solve the game. The main steps are:

1. Implement a Go endgame solver based on boolean negamax search, following the Tic-Tac-Toe example from class.
2. Implement a new GTP command `timelimit`, which sets the maximum time that your search can use.
3. Implement a new GTP command `solve`, which calls your Go endgame solver.
4. Integrate your solver into the `Go2` player so that it is used by the `genmove` command as well.
5. Use the public test cases to help assure correctness of your solver, and optimize your solver to crack harder test cases.

## Setup

1. Before assignment 1, you should have completed your activities 1b, 1c, 1d and 1e to set up Python 3 and NumPy, GoGui, and the initial cmput_496 code.
2. Now download assignment2.tgz. **New version Feb 17**. Unpack it to a directory `assignment2` which contains:
   - Directories `util` and `Go2` which together implement the `Go2` program.
   - Note that the `util` directory is different from the one originally released in `cmput_496`. `util` contains a new, faster version of the simple Go board, and some new useful functions such as `score` and `undo_move`.
   - Modify/add code **in subdirectory Go2 only** to implement your solution.
   - Several files with public test cases for your solver and player.
   - For reference, download the Tic Tac Toe solver from class.

# Goals of the Assignment

Implement the following functionality by adding or modifying existing GTP commands:

## TIMELIMIT

1. Implement a GTP command

   `timelimit seconds`

   The argument `seconds` is an integer in the range `1 <= seconds <= 100`. This command sets the maximum time to use for all following `genmove` or `solve` commands, until it is changed by another `timelimit` command.

   Before the first `timelimit` command is given, the default should be set to 1 second. See the public test cases.

## SOLVE

2. Implement a new GTP command `solve` which attempts to compute the winner of the current position, assuming perfect play by both, within the current time limit. Format:

   `solve`

   Your GTP response should be in the format:

   `= winner [move]`

   In the response, `winner` is either `b`, `w`, or `unknown`. Write `unknown` if your solver cannot solve the game within the current time limit. Solving always starts with the current player (toPlay) going first. If the winner is toPlay, then also write a winning move that you found. If there are several winning moves, then write **any one** of them. If the winner is the opponent or unknown, then do not write any move in your GTP response. See the public test cases.

   Your solve command must stay within the time limit. For some ideas of how to do that in Python 3, see Measuring Time. Whatever you do, make sure it works on the undergrad machines, since that is where we test your code.

   We will use non-integer komi in our test cases, so you do not need to handle the possibility of a draw. Also note that you only need to find the winner, not the best possible score, which is a more computationally demanding problem.

## MODIFY GENMOVE TO USE SOLVER

3. `genmove color`

   Here, the argument `color` is either 'b' or 'w'. See the public test cases. The `genmove` command is already implemented in the sample code. However, it does not call your solver yet. Change its behavior as follows:
   1. The program should use the solver to try to play perfectly. The solver must respect the current time limit as described under `timelimit` above. Note that the time limit is per move, not for the whole game.
   2. If the game is not solved within the timelimit, or if toPlay is losing, then the `genmove` command should fall back and play the same move as the original `Go2`: a random non-eye-filling move, or a pass.

# Recommended: Test Matches

**Note: please be considerate of shared resources! Run these tests on your own computer as far as possible, or on**

**very small boards if using a lab machine. Monitor the load of the shared machines e.g. with `top` and do not overload them. Last year, I got some unhappy emails from IST...**

We recommend that to ensure that your solver is working and keeping the time limits, you test your player by playing small-scale test matches between the original `Go2` player, and your player which includes your solver. For example, you could try games on 4x4 or 5x5 and use short timelimit for `genmove`. Use the gogui-twogtp tool to run the matches. See the gogui-twogtp notes.

Note that while we strongly recommend that you do these tests, since we think it will help you to develop the program, you do not submit any test results with the assignment.

## Public Test Cases

These test cases are in your `assignment2` directory.

- assignment2-test1.gtp
  - A few first test cases.
- **New Feb 17:** Public test cases: see the Assignment 2 Update

With an unmodified `Go2` program, all the tests which use the `solve` command currently fail, since it is not implemented. Tests with the genmove command will randomly succeed or fail, depending on whether `Go2` picks a winning move or not. As always, our public test cases only cover a small subset of possibilities. For evaluation, we will use a more comprehensive set of tests.

Note that our public tests include **multiple correct answers** in case a test has multiple possible solutions, while your program should always output **a single winning move**, if the current player can win. Also see the discussions of simple regular expression syntax used in gogui-regress on the gogui-regress man page and in the Gnu Go documentation.

## Other GTP Commands

Many other GTP commands are already implemented in the Go2 sample code. Do not break their functionality, since we will test your program using those commands, such as:

```
boardsize, clear_board, name, version, play,...
```

You are free to add to the implementations of these commands if you need it, e.g. depending on your approach, you may need to do something extra in the play command. But do not break the existing functionality.

These commands are used by the test script. In this assignment, for convenience we will use multiple calls to `boardsize` and `clear_board` within the same gtp file, so make sure this use case still works in your solution.

## Submission Checklist

### Submit a single tgz file called `assignment2.tgz` which contains exactly the following (and nothing else):

- A single directory `assignment2` which contains all the files in your solution, namely:
  1. All the files from the assignment2 directory **as updated on Feb 17**, but with the Python code in subdirectory Go2 modified to solve the assignment.
  2. The file `presubmission.log` from your presubmission testing (see below).

3. A file `readme.txt` which lists the names and student IDs of your team.

## Do Your Own Pre-submission Test

Follow the steps below on a standard undergraduate machine, and submit a text file `presubmission.log` that shows a copy of your command line presubmission testing.

1. Use the Linux script command to log your testing session to a file `presubmission.log`.
2. Copy your assignment2.tgz into a new directory.
3. Unpack it.
4. run gogui-regress with your program, using `assignment2-public-easy.gtp` as input.
5. Stop `script` logging here with the command `exit`.
6. Add the file `presubmission.log` to your submission, and compress it again.
7. Use `tar -tf` for a final check of your `assignment2.tgz` contents. Use the checklist above.
8. Fix any problems, then submit on eClass when you're ready.

## Marking

There will be a total of 5 marks for this assignment.

- 2 marks for passing our public test cases
- 1 mark for doing your own presubmission check and supplying the evidence in `presubmission.log`.
- 2 marks for passing our private test cases
- Bonus: up to 2 marks if your solver is significantly faster (1 mark), or very significantly faster (2 marks), than our (straightforward) sample solution.

Code that does not run will receive a mark of 0.

## Notes

- You can review the rules of Go in the Lecture 1 slides.
- We recommend that you do your search on a copy of the player's board, not on the original board. This is easier to clean up in case the search gets interrupted at the time limit. Otherwise it is easy to end up with the board in a modified state.
- Like Go1 and Go2 players, never generate eye filling moves in your search. In theory, this may make your search miss a good eye-filling move, but in practice such moves are extremely rare, and will not occur in our test cases.
- Determining the winner at the end of a game involves counting the score, as in Assignment 1. A valid score-counting function is included in the `Go2` program.
- In your search, use the `SimpleGoBoard.end_of_game()` function (new in `Go2`) to determine if a game has ended.
- If the game is over, use the `SimpleGoBoard.score()` function (new in `Go2`) to determine the winner. This function returns a pair `(winner, score)`. See the code for details.
- No special error handling code is required in this assignment. We will only test with valid moves and valid GTP commands which have valid arguments.
- We will only test with alternating play, so any `play` commands will be for the color toPlay. Clearing the board resets toPlay to Black.
- By default, we assume that teams will stay the same as in assignment 1. If you change your team, email the designated TA (Chenjun) before the assignment 2 deadline.
- We allow a 2 day late submission with a 20% deduction. See above.

- Note that being honest and returning "unknown" when your solver does not solve a position will lead to higher marks than random guessing the outcome.
- If you want to improve the efficiency of your solver for the bonus marks, there is a large number of different approaches you can take. We will give many different options in class, but we will leave it up to you what kinds of improvements you try. In any case, make sure to retain correctness of your solver. There will be no bonus marks awarded, and you will lose regular marks as well, if your program ``guesses'' the result, or does not compute the boolean minimax result correctly because of bugs. Extensive testing is the key.

---

Created: Jan 29, 2018 Last modified: Feb 17, 2018

*Martin Müller*

---