

# TCG HW2 Description

November 19, 2015

# HW2 Description

- Implement the  $9 \times 9$  GO.
- Require: *UCB*, *UCT* and *progressive pruning*.
- Bonus: Other techniques.
- Grading policy:
  - Basic grading policy:
    - Implement the basic requirement.
    - Defeat the random version program.
    - Your report.
  - Advance grading policy:
    - Other enhancement
    - Program ranking in the whole class
- Due Day: December 24, 2015, 14:20

- **Go Text Protocol (GTP):** often run with graphic user interface
  - run with option `-display` to auto display board
  - run with option `-nodisplay` or no option to disable auto display board.
- **Note:**
  - Your code will be tested with GTP version
  - All debug message should only output to **file** or **standard error**
  - Do not change the output format of GTP function
  - The time limit is **10 second** per move.

# Basic Commands for GTP version

- These functions is based on the Go Text Protocol
  - Reference: <http://www.lysator.liu.se/~gunnar/gtp/>
- Implemented command
  - `protocol_version` // Display the version of current protocol
  - `name` // Show the program name
  - `version` // Show the version of program
  - `known_command` // Ask program knows command or not.
  - `list_commands` // Show the list of all known commands
  - `boardsize` // Set the board size, currently only 9 is legal.
  - `clear_board` // Reset the board state.
  - `komi` // Set the number of komi (e.g. 6.5, 7, 7.5)
  - `play` // Play White/Black stone on game board
  - `genmove` // Call the engine to generate next move.
  - `undo` // Back to previous move
  - `showboard` // Display the current game board
  - `quit` // End the program

# Description of GTP Command

- boardsize *size*
  - Set the boardsize as *size*
  - The template code only support *size* = 9
- clear\_board
  - clear the gameboard
- kmoi *num*
  - set the komi as *num*, default is 7.
- play *b/w* [ABCDEFGHJ][1-9]
  - like put, put *b/w*'s stone at column [A-J], row [1-9]
  - row id is down to top.
  - column id is left to right.
- genmove *b/w*
  - generate *b/w*'s move
- undo
  - undo one move
- showbaord
  - show current gamebaord

# About the Template Code

- The variable in the template code is naming as follows:
  - Define constant: all upper letters.
    - BOARDSize, BOUNDARYSize.
  - Array: Initial character is upper letter.
    - Board, MoveList
  - Non-array variable: all letter is lower case
    - There are two exceptions, X and Y.
    - game\_length, num\_legal\_move

# Board Structure:

Board[BOUNDARYSIZE][BOUNDARYSIZE]

	0	1	2	3	4	5	6	7	8	9	10	
0	*	*	*	*	*	*	*	*	*	*	*	
1	*	.	.	.	.	.	.	.	.	.	*	9
2	*	.	.	.	.	.	.	.	.	.	*	8
3	*	.	.	+	.	.	.	+	.	.	*	7
4	*	.	.	.	.	.	.	.	.	.	*	6
5	*	.	.	.	.	+	.	.	.	.	*	5
6	*	.	.	.	.	.	.	.	.	.	*	4
7	*	.	.	+	.	.	.	+	.	.	*	3
8	*	.	.	.	.	.	.	.	.	.	*	2
9	*	.	.	.	.	.	.	.	.	.	*	1
10	*	*	*	*	*	*	*	*	*	*	*	
		A	B	C	D	E	F	G	H	J		

- BOUNDARYSIZE: 11
- BOARDSize: 9
- Board[i][j] is (x,y) = (j, 10-i) in the game board

# Genmove Function

- *gen\_legal\_move(Board, turn, game\_length, GameRecord, MoveList)*
  - generate all the legal move
  - return the number of legal moves.
- *rand\_pick\_move(num\_legal\_moves, MoveList)*
  - randomly pick one legal move
  - return the selected move.
  - **You should replace this function.**
- *do\_move(Board, turn, move)*
  - update the current board with “move”



# *gen\_legal\_move* Function

- For each empty intersection
  - Check if the empty intersection is a legal move
  - Check if the legal move will result in a repeat board
  - Add the move to move list.
    - each move is a 3 digit integers  $ejj$
    - $e$  denote this is a capture move (1) or not (0).
    - $ij$  denote the location of  $\text{Board}[i][j]$
    - e.g. 123: put stone in  $\text{Board}[2][3]$  is a capture move.
    - e.g. 056: put stone in  $\text{Board}[5][6]$  is not a capture move.

# Function for Checking Legal Move

- *count\_neighborhood\_state(Board, X, Y, turn, \*empt, \*self, \*oppo, \*boun, NeighborhoodState)*
  - return the number of
    - Empty intersection
    - Self intersection
    - Opponent intersection
    - Boundary intersection
  - Record the state of each neighborhood in NeighborhoodState.
- *count\_liberty(X, Y, Board, Liberties)*
  - count the number of liberties in each direction's string.
  - The result is saved in Liberties.
  - Using DFS method.

# Legal Move

- A move is legal if
  - At least one neighborhood intersection is empty.
  - One of the self string has more than one liberty.
    - And it's not a self-eye.
  - One of the opponent string has only one liberty.

# Do the move

- Update the Board with
  - play Black/White [ABCDEFGHJ][123456789]
- `update_board(Board, X, Y, turn)`
  - put turn's piece in (X, Y)
  - will not check if (X, Y) is a legal move.
- `update_board_check(Board, X, Y, turn)`
  - put turn's piece in (X, Y)
  - will check if (X, Y) is a legal move.
  - return 1 if (X, Y) is a legal move
  - return 0 if (X, Y) is a illegal move

# Avoid the repeat board

- GameRecord[MAXGAMELENGTH][BOUNDARYSIZE][BOUNDARYSIZE]
- game\_length
- Check the all the board in the GameRecord.

# Result Counting

- $\text{final\_score}(\text{Board})$ 
  - black area: black stones + black eyes
  - white area: white stones + black eyes
  - result: black area - white area
- $\text{final\_result} = \text{final\_score} - \text{komi}$ 
  - $> 0$ : B+[result]
  - $= 0$ : 0 (draw)
  - $< 0$ : W+[-result]

# Introduction of GoGui

- Homepage: <http://gogui.sourceforge.net/>
  - You can find the download link here.
- Run a computer selfplay
  - 1 Game  $\Rightarrow$  game size  $\Rightarrow$  9
  - 2 Game  $\Rightarrow$  Game info  $\Rightarrow$  Komi 7
  - 3 Program  $\Rightarrow$  New Program
    - Command: the path to your execution file.
  - 4 Program  $\Rightarrow$  Attached  $\Rightarrow$  Your program
  - 5 Game  $\Rightarrow$  Computer Color  $\Rightarrow$  Both

# Selfplay Via GoGui

- `gogui-twogtp`
  - `-white` [white program name]
  - `-black` [black program name]
  - `-games` [number of games]
  - `-alternate`
  - `-size` 9
  - `-komi` 7
  - `-verbose`
  - `-sgffile` [filename]
    - `filename.dat`: statistic result
    - `filename-0.sgf` - `filename-[N-1].sgf`
  - `-auto`
- Example: `gogui-twogtp -white white.exe -black black.exe -games 10 -alternate -size 9 -komi 7 -verbose -sgffile record_name -auto`
- Using Gogui to display:
  - `gogui -program "gogui-twogtp ..." -size 9 -computer-both -auto`



# About -games and -sgffile in gogui-twogtp

- -games N means gogui-twogtp will play N games.
- -sgffile [filename] means the result will be saved with prefix "filename"
- If filename.data exists and contains k games.
  - If  $N \leq k$ , then gogui-twogtp will do nothing.
  - If  $N > k$ , then gogui-twogtp will play exact  $N-k$  games.
  - If you want to play exact N games:
    - remove filename.data
    - or add option -force to overwrite filename.dat
- Files with extension sgf are the game record of each game.
  - Index from 0 to N-1
  - Can be opened by gogui
    - File  $\Rightarrow$  Open.

# Other notification

- When each game start, the protocol will call function *reset(Board)*.
  - Beware to initial all your self data strcture here.
- Provide your Makefile or specfied how to compile your codes in the report.
- Gogui can show the graphic user interface via Xming and pitty.
  - Start Xming
  - pitty => putty mode
  - session: host name or ip
  - Connection => SSH => X11:
    - Select "Enable X11 forwarding"
    - X display location: 127.0.0.1:0
  - Open