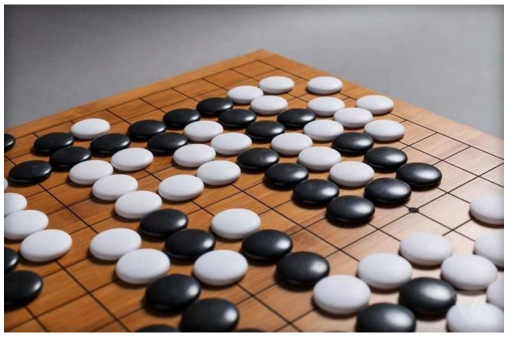
Competition Project: Go

Due Time: December 9, 2019, 3:00PM

Content

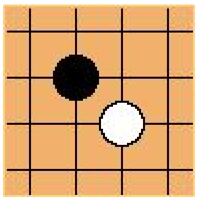
In this competition project, we will have a lot of fun playing with a sophisticated board game: Go. Your task is to train a Go agent using reinforcement learning to beat other players.



1. Go Introduction

Go is an abstract strategy board game for two players, in which the aim is to surround more territory than the opponent.

1.1 Basic Concepts



- Players: Go is played by two players, called Black and White.
- Board: The Go board is a grid of horizontal and vertical lines. The standard size of the board is 19x19.
- Point: The lines of the board have intersections wherever they cross or touch each other. Each intersection is called a **point**. Intersections at the four corners and the edges of the board are also called a **point**. *Go is played on the points of the board, not on the squares*.
- Stones: Black uses black stones. White uses white stones.

1.2 Game Process

- Starts with an empty board (usual size is 19*19).
- The two players take turns placing stone on the board, one stone at a time.
- The players may choose any unoccupied intersection to play on (except for those forbidden by the ko and suicide rules).
- Once played, a stone can never be moved and can be taken off the board only if it is captured.

1.3 Basic Rules

Rule1: The rule of liberty

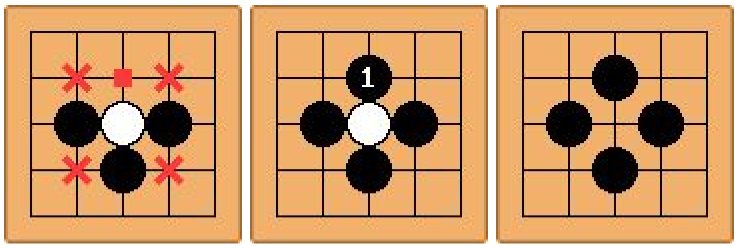
Every stone remaining on the board must have at least one open point, called a *liberty*, directly orthogonally adjacent (up, down, left, or right), or must be part of a connected group that has at least one such open point (liberty) next to it. Stones or groups of stones which lose their last liberty are removed from the board (called *captured*).

Subrule: No Suicide

A player may not place a stone such that it or its group immediately has no liberties, unless doing so immediately deprives an enemy group of its final liberty. In the latter case, the enemy group is captured, leaving the new stone with at least one liberty.

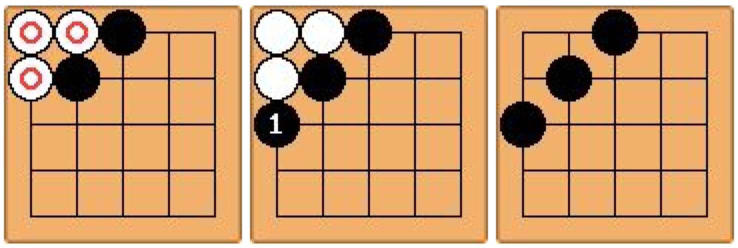
Examples of capturing:

- Example 1



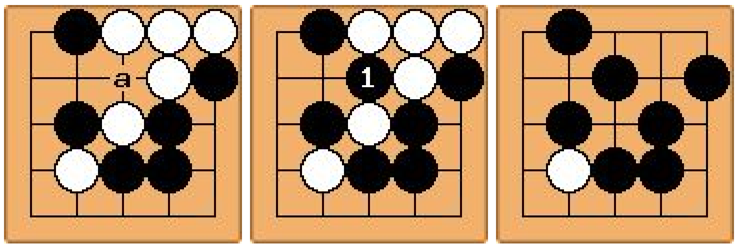
The white stone is captured.

- Example 2



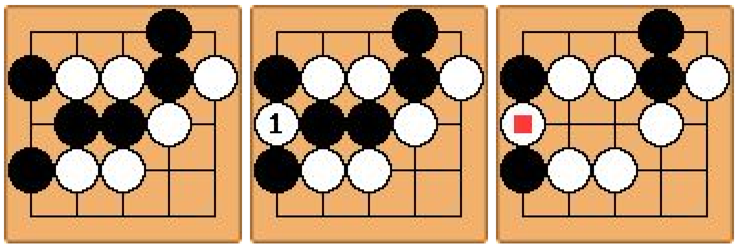
The 3 white stones are captured as a connected group.

- Example 3



The two groups of white stones are captured.

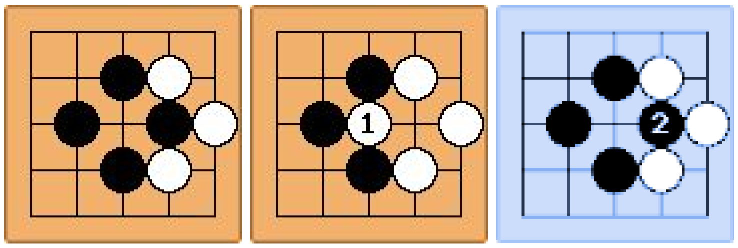
- Example 4 (Special example)



This example illustrates the rule that a capturing stone
need not have a liberty until the captured stones are removed.

Rule 2: "KO rule"
The stones on the board must never repeat a previous position of stones. Moves which would do so are forbidden, and thus only moves elsewhere on the board are permitted that turn.

Example move forbidden by the KO rule:



Black **cannot** play at position 2 in the example.
Black must play on a different point **in this turn**.

1.5 Winner Judging

There are various scoring rules and winning criteria for Go. But we will adopt the following rules for the scope of this project.

“Partial” Area Scoring
A player's score is the number of stones that the player has on the board.

Winning Criteria
The winner is the player that has the higher score at the end of the game.

End of Game
A game ends when it reaches one of the two conditions:

- When one player has no valid moves to make.
- When the game has reached the maximum steps.

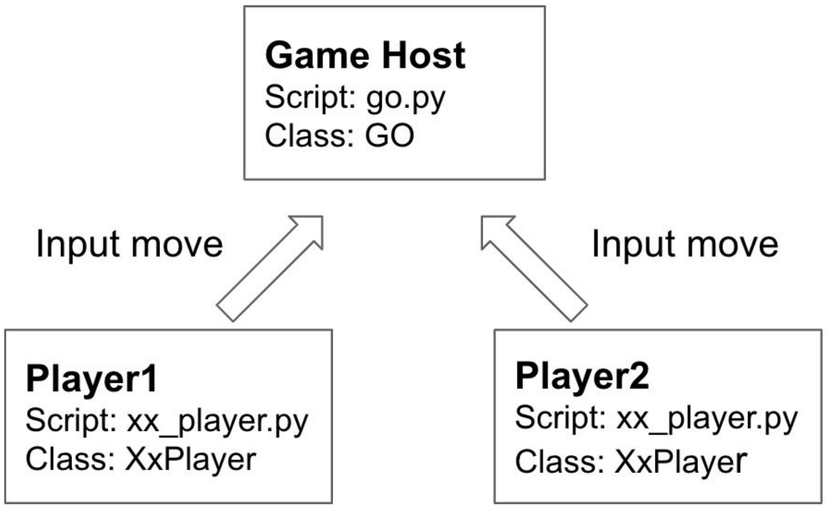
Komi
Because Black has the advantage of playing the first move, awarding White some compensation is called **Komi**, which gives White a compensation of score at the end of the game.

This particular set of rules references several popular rules sets around the world, but some changes are made in order to best adapt to this project. For example, the “Full” Area Scoring counts the number of stones that the player has on the board, **plus the number of empty intersections surrounded by that player's stones**. Also, a player may waive his/her right to make a move, called **passing**, when determining that the game offers no further opportunities for profitable play. **The game ends when both players pass, and is then scored.** Go is a very complicated game, do some more research if you are interested.

2. Project Startcode

In this project, you will be provided some start code to begin with. You are strongly recommended to read and understand the start code before starting your own code.

2.1 Program Structure



The above picture shows the program structure. There are two players and a game host in each game. The **Game Host** keeps track of the game process, gets input from the players, judges if the input is valid, wipes out the dead stones and finally judges the winner. The **Player**s take turns to input the move (row, column) coordinator to the Game Host.

2.2 Rule Parameters

The following parameters are adopted in the scope of this competition project.

- In a board, 0 stands for empty point, 1 stands for Black stone, 2 stands for White stone.
- For visualization, X for Black stone and O for White stone.
- Black always plays first.
- The default board size is 5*5. But we might also play with size 7*7 in the second stage if possible.
- The maximum movement is $n * n - 1$. For example, max movement of a board size 5*5 is 24.
- Komi for White player is $n/2$. For example, Komi of a board size 5*5 is 2.5. If White scores 10 and Black scores 12 at the end of the game, then White is the winner ($10 + 2.5 = 12.5 > 12$).

2.3 go.py

Some attributes and methods that need additional specifications in `go.py` are as below. For more details please read the script carefully.

Attribute: board
There is a board attribute in every GO class. All players make moves on the same board by inputting the (i, j) coordinate. For manual players, we visualize the board using X and O.

Coordinate system

	j=0	j=1	j=2	j=3	j=4
i=0	0	0	1	1	0
i=i	0	0	2	1	0
i=2	0	0	2	0	0
i=3	0	2	0	0	0
i=4	0	0	0	0	0

Board visualization

		X	X	
		O	X	
		O		
	O			

Attribute: died_pieces
This attribute is for keeping track of the dead stones of last round. This is helpful when checking if the stone placement is valid under the KO rule.

Function: copy_board(self)
This function copies the GO instance for potential testing. It is useful when you want to do some tests on the board but don't want to modify the original board.

Function: valid_place_check(self, i, j, piece_type, test_check=False)
This function examines whether a stone placement (i, j) is valid or not for a given piece_type(player). This could be useful when you search the valid placements on the board. The argument test_check is to control the verbosity when there is a manual player so you can basically ignore it when developing your code.

Function: place_chess(self, i, j, piece_type)
This function first examines whether the placement is valid or not by calling **valid_place_check**, then modifies the board in place if the placement is valid. **DO NOT** use this function to the original GO instance in your player class. You might need to copy the GO instance first and then modifies the copied GO instance. You can refer to the code in the greedy player for copying and testing.

Function: remove_died_pieces(self, piece_type)
This function removes all the dead stones in the board of a given piece_type(player) in place. After placing a stone you have to call this function to remove the dead stones for the placement. Again you might need to do this on the copied test board.

Function: play(self, player1, player2, verbose=False)
The most crucial method in GO class is play(player1, player2). This method takes two Player instances as arguments and starts the game. It keeps track of the game process, gets input from both players, update the board and wipes out dead stones, and finally judges the winner when the game ends.

2.4 players

All the players is a XxPlayer class. It should at least have a get_input() method so that it inputs its move when the host calls this function.

Class: ManualPlayer
ManualPlayer class refers to a human player. You can try to play the game against another human player, or against an AI player (random, greedy or your own agent). When there is at least one manual player, the game would become verbose and there would be board visualization after each move.

Class: AI Players
There are some players given to you for testing and grading. The .py files of RandomPlayer and GreedyPlayer are given to you. While only the compiled .pyc files of the AggressivePlayer and SmartPlayer are given to you.

There are some features of those AI players:

- RandomPlayer: moves randomly.
- GreedyPlayer: places the stone that can kill the most stones from the opponent.
- AggressivePlayer: looks on the next two possible moves and tries to kill the most stones from the opponent.
- SmartPlayer: also defends itself when necessary while being aggressive.

Class: MyPlayer
You will need to write this MyPlayer class as your own agent in this project. Write the code in `my_player.py.`

2.5 go_play.py

`go_play.py` integrates go game with players and is the main function we will use.
Usage: `go_play.py [-h] [--size SIZE] [--player1 PLAYER1] [--player2 PLAYER2] [--times TIMES]`

Arguments:

```
  -h, --help            show this help message and exit
  --size SIZE, -n SIZE
                        size of board n*n Default: 5.
  --player1 PLAYER1, -p1 PLAYER1
                        Player class. Options: manual, random, greedy, aggressive, smart, my
                        Default: random.
  --player2 PLAYER2, -p2 PLAYER2
                        Player class. Options: manual, random, greedy, aggressive, smart, my
                        Default: random.
  --times TIMES, -t TIMES
                        playing times. Default: 1.
```

Example execution:

```
$ python3 go_play.py -n=5 -p1=manual -p2=random -t=1
```

- This starts one game between a manual player(Black) and a random player(White) in the board of size 5*5.

```
$ python3 go_play.py -p1=greedy -t=100
```

- This starts 100 games between a greedy player(Black) and a random player(White) in the board of size 5*5.

Example stdout:

```
Black player (X) | Wins:83.0% Loses:17.0%
White player (O) | Wins:17.0% Loses:83.0%
```

3. Project Instructions

3.1 Task Description

Your task is to implement **MyPlayer** in `my_player.py`. You are **NOT** eligible to rewrite `go.py` or go_play.py or any other player scripts. In the later stages of competition we will use your agent to play against the agents from other students, so it is important that we share the same rules. Other helper files or scripts are acceptable, such as files to store Q-values table or helper functions.

Try first to fully understand the Go rules and the start code before developing your own code. You can play the game manually against an AI player, or even play with your friend.

You are **required but not limited to** use reinforcement learning in this project. You are encouraged to research on other possible methods or Go tactics that may help you defeat your opponent.

In your implementation, ***please do not use any existing machine learning library call***. You must implement the algorithm yourself. Please develop your code yourself and do not copy from other students or from the Internet.

3.2 Grading Format

There will be 2 stages in this competition.

First Stage
In the first stage, you will be graded by playing against the AI players for 100 times. 50 times as Black and 50 times as White.

Execution format:
```
$ python3 go_play.py -n=5 -p1=my -p2=random -t=50
$ python3 go_play.py -n=5 -p1=random -p2=my -t=50
$ python3 go_play.py -n=5 -p1=my -p2=greedy -t=50
$ python3 go_play.py -n=5 -p1=greedy -p2=my -t=50
$ python3 go_play.py -n=5 -p1=my -p2=aggressive -t=50
$ python3 go_play.py -n=5 -p1=aggressive -p2=my -t=50
$ python3 go_play.py -n=5 -p1=my -p2=smart -t=50
$ python3 go_play.py -n=5 -p1=smart -p2=my -t=50
```

Second Stage
In the second stage, you will be graded by playing against the agents from other students. We'll first modify the `go_play.py`:

> from student1_player import MyPlayer as s1
> from student2_player import MyPlayer as s2

And then:
```
$ python3 go_play.py -n=5 -p1=s1 -p2=s2 -t=6
$ python3 go_play.py -n=5 -p1=s2 -p2=s1 -t=5
```

3.3 Rubric

- First stage(100 pts)

Win rate	**Opponent**			
	RandomPlayer	**GreedyPlayer**	**AggressivePlayer**	**SmartPlayer**
>= 90%	30pts	30pts	20pts	20pts
>= 70%	20pts * win rate	20pts * win rate	15pts * win rate	15pts * win rate
< 70%	0pts	0pts	10pts * win rate	10pts * win rate

Example:
- VS random player: win 95 games out of 100 games → 30 pts.
- VS greedy player: win 85 games out of 100 games → $20 * 0.85 = 17$ pts
- VS aggressive player: win 80 games out of 100 games → $15 * 0.8 = 12$ pts
- VS smart player: win 65 games out of 100 games → $10 * 0.65 = 6.5$ pts

So total points: $30 + 17 + 12 + 6.5 = 65.5$ pts.

- Second stage(extra 20 pts)
  - Students that get **80 points** or more will get to the second stage.
  - Students will play against each other in this stage.
  - For student A and student B, they will play 11 games. Whoever wins more in the 11 games is the final winner of A and B, counted as one **WIN** for the player.
  - The student that has higher score in the first stage would have one more chance of first move(Black). If two students have the same score, the extra chance to move first is decided randomly.
  - Each student will play with **all the other students** entered this stage.
  - The student will be ranked according to the WIN counts.

Rank	Points	Additional prize
1-2	20 pts	Recommendation from Wei-min Shen
3-5	15 pts	
6-10	10 pts	
After 10	5 pts	

Tentative: a rank leaderboard will be posted on Piazza weekly on Monday.
Note that TAs may also participate in this competition.

3.4 Submission

Submit your code to Vocareum

- Submit `my_player.py` to ***Vocareum*** with other helper documents you may need. Do **NOT** submit `go.py`, `go_play.py` or other AI player programs.
- Your program is suggested to finish within ***20 minutes***(to complete the 400 games against AI players) due to the instability of Vocareum time limit.
- After submission you can view your submission report to see if your code works, while the grading report will be released after the deadline of the project.
- You don't have to keep the page open while the scripts are running.

This time you do NOT need to submit your code or report to Blackboard. Your score on Vocareum will be your final score.

4. Discussion and Feedback

The scope of this competition is quite large. The teaching staff team cannot make sure that our implementation is 100% correct. If you have any questions or suggestions while completing this project, for example, if you feel that some functions are necessary in the GO class, or find a bug in the program, feel free to post and discuss on Piazza with other students and teaching staff team. Also, check Piazza frequently for discussion and possible new announcements. The teaching staff team might adjust the second stage of the competition according to the actual situation in the battlefield.

References

1. https://en.wikipedia.org/wiki/Go_(game)
2. https://senseis.xmp.net/?BasicRulesOfGo
3. https://senseis.xmp.net/?Ko