CSCI-561 – Fall 2023 - Foundations of Artificial Intelligence
Homework 2

Due October 25, 2023, 23:59:59 PST

1. Overview

In this programming assignment, you will develop your own AI agents based on some of the AI techniques for Search, Game Playing, and Reinforcement Learning that you have learned in class to play a small version of the Go game, called Go-5x5 or Little-Go, that has a reduced board size of 5x5. Your agent will play this Little-Go game against some basic as well as more advanced AI agents. Your agents will be graded based on their performance in these online game "tournaments" on Vocareum.com. Your objective is to develop and train your AI agents to play this Little-Go game as best as possible.

2. Game Description

Go is an abstract strategy board game for two players, in which the aim is to surround more territory than the opponent. The basic concepts of Go (Little-Go) are very simple:

- 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, but **in this homework, the board size will be 5x5.**
- 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 **points**. *Go is played on the points of the board, not on the squares*.
- Stones: Black uses black stones. White uses white stones.

The basic process of playing the Go (Little-Go) game is also very simple:

- It starts with an empty board,
- Two players take turns placing stones on the board, one stone at a time,
- The players may choose any unoccupied point to play on (except for those forbidden by the "KO" and "no-suicide" rules).
- Once played, a stone can never be moved and can be taken off the board only if it is captured.

The entire game of Go (Little-Go) is played based on two simple rules: Liberty (No-Suicides) and KO. The definitions of these rules are outlined as follows:

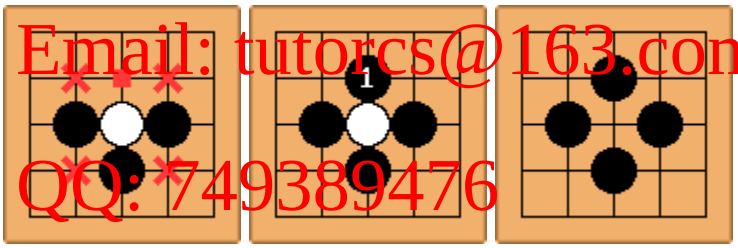
Rule1: The Liberty Ru

Every stone remaining e at least one open point, called a *liberty*, directly orthogonally adjacent ( ), or must be part of a connected group that has at least one such open point (li s or groups of stones which lose their last liberty are removed from the boa

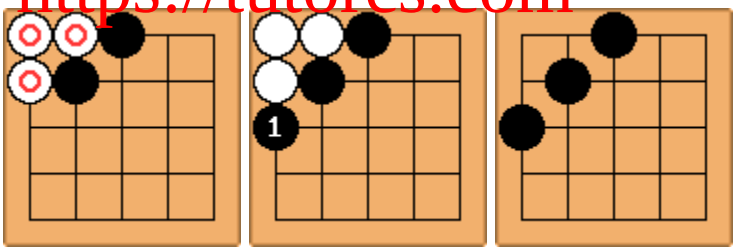
Based on the rule of liberty, **players are NOT allowed to play any "suicide" moves**. That is, a player cannot place a stone such that the played stone or its connected group has no liberties, unless doing so immediately deprives an enemy group of its final liberty. In the latter case, the enemy group is captured, leaving the new stone with at least one liberty.

Examples of capturing:

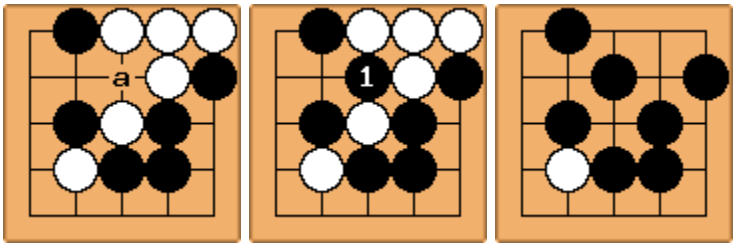
- Example 1. The white stone is captured after Black plays at position 1, because its directly orthogonally adjacent points are occupied.



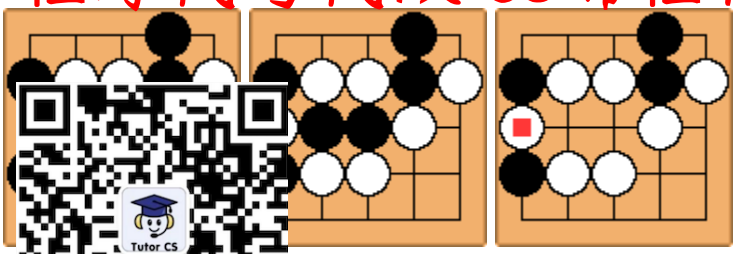
- Example 2. The 3 white stones are captured as a connected group.



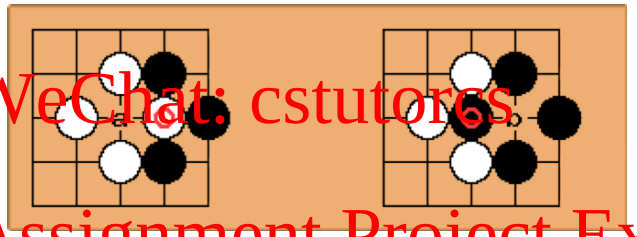
- Example 3. The two groups of white stones are captured.



- Example 4 (Special example). This example illustrates the rule that a capturing stone need not have liberty until the captured stones are removed.



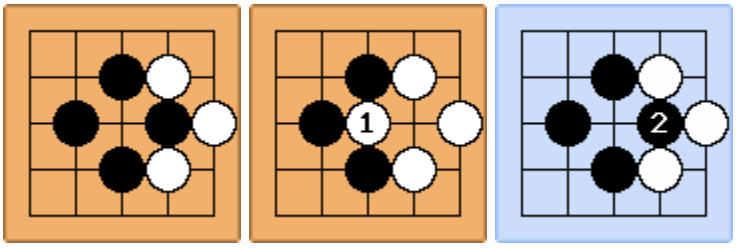
Rule 2: The "KO" Rul



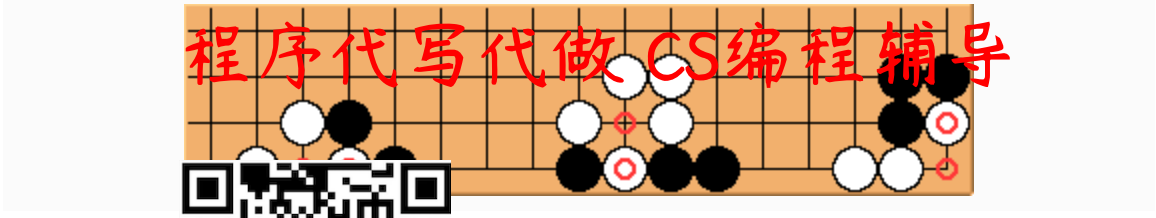
For the position shown on the left board above, Black can capture the ◎ stone by a play at position *a*. The resulting position is shown on the right board above. Without a KO rule, in this position White could recapture the ● stone at position *b*, reverting to the position shown on the left, and then Black could also recapture. If neither player gave way, then we would have Black *a*, White *b*, Black *a*, White *b*, ..., repeated ad infinitum, stalling the progress of the game. This situation is known as **KO**.

The **KO rule** resolves the situation: **If one player captures the KO, the opponent is prohibited from recapturing the KO immediately.**

- Example. Given the initial status on the left below, the white player puts a stone at position 1, which captures a black stone. Black stone **cannot** be placed at position 2 **immediately** after it's captured at this position. Black must play at a different position **this turn**. Black can play at position 2 the **next turn** if this position is still not occupied.



- More examples. KOs need not occur only in the center of the board. They can also show up at the sides or corners of the board, as shown in the diagram below.



Komi

Because Black has the a e first move, awarding White some compensation is called **Komi**. This is in the forr npensation of score at the end of the game. In this homework (a board siz White player is set to be 5/2 = 2.5.

Passing

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. A player may pass his/her turn at any time. Usually, passing is beneficial only at the end of the game, when further moves would be useless or maybe even harmful to a player's position.

End of Game

A game ends when it reaches one of the four conditions:

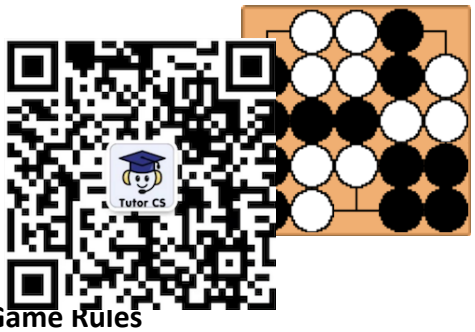
- When a player's time for a single move exceeds the time limit (See Section 6. Notes and Hints).
- When a player makes an invalid move (invalid stone placement, suicide, violation of KO rule).
- When both players waive their rights to move. Namely, two consecutive passes end the game.
- When the game has reached the maximum number of steps allowed. In this homework (a board size of 5x5), the maximum number of steps allowed is (5*5)-1 = 24.

Winning Condition

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

- "Partial" Area Scoring: A player's partial area score is the number of stones that the player has occupied on the board.
- Final Scoring: The Black player's final score is the partial area score, while the White player's final score is the sum of the partial area score plus the score of compensation (Komi).
- Winning Criteria:
    - If a player's time for a single move exceeds the time limit (See Section 6. Notes and Hints), s/he loses the game.
    - If a player makes an invalid move (invalid stone placement, suicide, violation of KO rule), s/he loses the game.
    - If the game reaches the maximum number of steps allowed or if both players waive their rights to move, the winner is the player that has a higher **final score** at the end of the

game. For example, in the following board at the end of a game, White's partial area score is 10 and Black's partial area score is 12. White is the winner because $10 + 2.5 = 12.5 > 12$.



Clarification of the Game Rules

The particular set of rules that have been adopted in this assignment references several popular rule sets around the world, but some changes have been made to best adapt to this project. For example, "Full" Area Scoring is usually used in the 19x19 Go game, which counts the number of stones that the player has on the board plus the number of empty intersections enclosed by that player's stones, but we do not use this rule in our assignment. Go is a very interesting and sophisticated game. Please do some more research if you're interested.

3. Academic Honesty and Integrity

All homework material is checked vigorously for dishonesty using some advanced AI and Data Mining agents and almost all coping pieces of evidence, no matter how they are hidden, would be detected. These agents compare the submissions with previous semesters and similar codes on the internet including GitHub and other similar places. All detected violations of academic honesty are forwarded to the Office of Student Judicial Affairs. To be safe, you are urged to err on the side of caution. Do not copy work from another student or off the web. Keep in mind that sanctions for dishonesty are reflected in *your permanent record* and can negatively impact your future success. As a general guide:

Do not copy code or written material from another student. Even single lines of code should not be copied.

Do not collaborate on this assignment. The assignment is to be solved individually.

Do not copy code off the web. This is easier to detect than you may think.

Do not share any custom test cases you may create to check your program's behavior in more complex scenarios than the simplistic ones that are given.

Do not copy code from past students. We keep copies of past work to check for this. Even though this project differs from those of previous years, do not try to copy from homeworks of previous years.

Do not ask on Piazza how to implement some function for this homework, or how to calculate something needed for this homework.

Do not post code on Piazza asking whether or not it is correct. This is a violation of academic integrity because it biases other students who may read your post.

Do not post test cases on Piazza asking for what the correct solution should be.

Do ask the professor or TAs if you are unsure about whether certain actions constitute dishonesty. It is better to be safe than sorry.

4. Playing against Other Agents

In this homework, your agent will play against other agents created by the teaching staff of this course.

4.1 Program Structur

Figure 1 shows the bas here is one game host and two players in each game. The **Game Host** keeps track gets the next moves from the players in turn, judges if the proposed moves are va stones, and finally judges the winner. Each of the two **Players** must output its given format (in a file called output.txt) with the intended point (row and column me Host. The job of a player is very simple: take the previous and current states of the board (in a file called input.txt) from the host, and then output the next move back to the host.

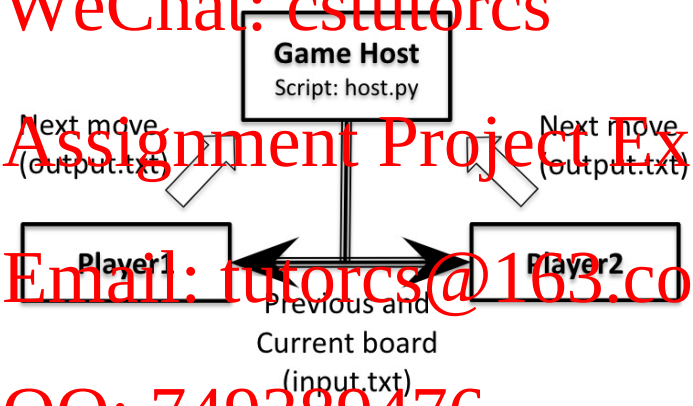


Figure 1: The Program Structure for the Little-Go Game.

4.2 Rule Parameters

The following parameters have been adopted for this homework project:

- In a board, 0 stands for an empty point, 1 stands for a Black stone, and 2 stands for a White stone.
- In board visualizations, X represents a Black stone and O represents a White stone.
- Black always plays first.
- The board size is 5x5.
- The maximum number of moves allowed is $(n * n) - 1$, where *n* is the size of the board. For example, max number of moves allowed for a board of size 5x5 is (5*5)-1=24.
- Komi for the White player is $n/2$. For example, Komi for a board of size 5x5 is 2.5. If White scores 10 and Black scores 12 at the end of the game, then White is the winner ($10 + 2.5 = 12.5 > 12$).

4.3 The Game Board

Coordinate system

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

Board visualization

		X	X	
		O	X	
		O		
	O			

Figure 2: The Format of the Current Board for the Little-Go Game.

The host keeps track of the game board while the two players make moves in turn. We will use a zero-based, vertical-first, start at the top-left indexing in the game board. So, location [0,0] is the top-left corner of the board, location [0,4] is the top-right corner, location [4,0] is the bottom-left corner, and location [4,4] is the bottom-right corner. An example of game state is shown in Figure 2, in which "1" denotes black stones, "2" denotes white stones, and "0" denotes empty positions. For manual players, we visualize the board as in the image on the right where **X** denotes the black stones and **O** denotes the white stones.

4.4 Players and Game Host

AI Players

Different AI Players are available for your agent to play against for the purpose of testing and/or grading. Examples of these existing AI players include:

- Random Player: Moves randomly.
- Greedy Player: Places the stone that captures the maximum number of enemy stones
- Aggressive Player: Looks at the next two possible moves and tries to capture the maximum number of enemy stones.
- Alphabeta Player: Uses the Minimax algorithm (Depth<=2; Branching factor<=10) with alpha-beta pruning.
- QLearningPlayer: Uses Q-Learning to learn Q values from practice games and make moves intelligently under different game conditions.
- Championship Player: This is an excellent Little-Go player adapted from top-performing agents in previous iterations of this class.

Manual Player

To familiarize yourself with the game rules, you can manually play this game on Vocareum as “Player1” against a chosen agent (e.g., the Random Player) “Player2”. When there is at least one manual player,

the game board will be visualized. **On Vocareum, please click the "Run" button to play a YouSelf-vs-RandomPlayer game. You are highly encouraged to do this to get acquainted with how the game works. In case you face issues with the button, please run "bash ../resource/scripts/run.sh" in the terminal to perform the same action.**

Your Programed A

You will need to write y project. Name your agent as my_player.xx, where xx is the conventional extension See Section 6 Notes and Hints), and upload your my_player.xx into the v eum. Then, **when you click the "Submit" button, your uploaded agent will be ost to play against the random player, the greedy player, and the aggressive player. The results of these games will also be reported.**

The Game Host

The Game Host integrates the Go game with the players. During a game play process, the Game Host will perform the following steps:

Loop until game ends:

- Alter the CurrentPlayer
- Clean up any input.txt and output.txt in the player's directory
- Provide the current and previous game boards by creating a new input.txt (see format below)
- Call the CurrentPlayer's agent, which reads input.txt and creates a new output.txt
- Validate the new output.txt, and process the proposed move
    - The validity of your move will be checked. If the format of output.txt is incorrect or your move is invalid as per the rules of the game, your agent loses the game.
- Check if the game has ended. If so, the winning agent is declared.

For testing purposes, your agent can play this game on Vocareum against one chosen (The Random Player) AI agent. **On Vocareum, please click the "Build" button to play a YourPlayer-vs-RandomPlayer game once you have my_player.xx uploaded into the "work" directory. In case you face issues with the button, please run "bash ../resource/scripts/build.sh" in the terminal to perform the same action.**

To help you get started, the basic source code (in Python) for playing the Little-Go game (i.e., the host, the random player, and the read and write functions) are available for you to see. You can download them from HW2/stage1/resource/$ASNLIB/public/myplayer_play onto your local machine and play there. For example, you can:

- Duplicate random_player.py and rename one copy to my_player3.py,
- Download build.sh to the same directory, modify line 36: prefix="./",
- Start a game between two random_players on your machine using the following command:
  $ sh build.sh (or $ bash build.sh)

On Vocareum, you can do the same by copying random_player.py into your work directory, renaming it as my_player3.py, and then clicking on "Build" for the two random players to start playing against each other. You may re-use the provided convenience functions in host.py, read.py, etc; however, **it is your responsibility to make sure that your program runs correctly**.

5. Project Instruction

5.1. Task Description

Your task is to implement [illegible]ayer.xx. Note that in the grading competitions, your agent will play against other [illegible]mportant that your agent inputs and outputs in the exact same format as specified [illegible] (either .json or .txt) or scripts are acceptable, such as files to store Q-value tables or other types of helper functions. These are optional. Please note that only source code files (.java, .py or .cpp) and helper files (.json or .txt) are acceptable.

You are required to use the AI techniques you learnt in class for your implementation. In addition to the regular game playing techniques, such as Minimax with alpha-beta pruning, you are also highly encouraged to use reinforcement learning in this homework in order to defeat the advanced agents in the second stage of this assignment (See Section 5.2). You are also encouraged to research other possible methods or Go tactics that may help you design stronger agents.

In your implementation, ***please do not use any publicly available library calls for minmax, alpha-beta pruning,Q-Learning, or any other search/reinforcement learning algorithm***. You must implement the algorithms from scratch yourself. Please develop your code yourself and do not copy from other students or from the Internet.

5.2. Grading

Your agent will be graded based on its performance against other chosen agents.

The First Stage (80 games, max 90 pts)

In the first stage, you will be graded by playing 20 times against each of the four existing AI players (random, greedy, aggressive, and alpha-beta) in the table below; 10 times as Black and 10 times as White. **Note that when you click the "submit" button, your agent will play against only the random, greedy, and aggressive agents. The alpha-beta agent is reserved for grading.**

Grading Rubrics for the 1st Stage (max 90 points)

WinRate	Opponent Agent and Availability			
	1st Stage			
	Available for Testing on Vocareum			**Reserved for Grading**
	Random	**Greedy**	**Aggressive**	**Alpha-beta**
>=90%	25pts	25pts	25pts	15pts
>=70%	15pts*winRate	15pts*winRate	20pts*winRate	10pts*winRate
<70%	0pts	0pts	10pts*winRate	5pts*winRate

Example:

- VS random player: win 18 games out of 20 games, WinRate=90% → 25pts.
- VS greedy player: win 16 games out of 20 games, WinRate=80% → $15 * 0.8 = 12$pts
- VS aggressive player: win 14 games out of 20 games, WinRate=70% → $20 * 0.70 = 14$ pts
- VS alpha [illegible]mes out of 20 games, WinRate=55% → $5 * 0.55 = 2.75$ pts

So the total po[illegible]ll be: $25 + 12 + 14 + 2.75 = 53.75$pts.

The Second Stage [illegible]0 pts)

In the second stage, your agent will play against two high-performance agents: the Q-Learning agent and the Championship agent, 20 times against each, 10 times as Black and 10 times as White. **Note that when you click the "submit" button, your agent will play only 10 games (5 games as Black and 5 games as White) each against the Q-Learning and Championship agents. The real grading against 20 games each will take place after the assignment due date.** The grading schema is defined as follows:

Grading Rubrics for the 2nd Stage (max 10 points)

WinRate	Opponent Agent and Availability	
	2nd Stage	
	Available for Testing on Vocareum	
	Q-Learning Agent	**Championship Agent**
>=90%	5pts	5pts
>=70%	5pts*winRate	5pts*winRate
<70%	0 pts	2pts*winRate

The format of input and output that your agent processes must exactly match that outlined below in Section 5.3, otherwise, your agent will never win any game. **End-of-line character is LF** (since Vocareum is a Unix system and follows the Unix convention).

5.3. Input and Output

Input: Your agent should read input.txt from the current ("work") directory. The format is as follows:

- Line 1: A value of "1" or "2" indicating which color you play (Black=1, White=2)
- Line 2-6: Description of the previous state of the game board, with 5 lines of 5 values each. This is the state after your last move. (Black=1, White=2, Unoccupied=0)
- Line 7-11: Description of the current state of the game board, with 5 lines of 5 values each. This is the state after your opponent's last move (Black=1, White=2, Unoccupied=0).

For example:

========input.txt========

2

00110

00210

00200

02000

00000

00110
00210
00200
02010
00000
==========

At the beginnin[illegible]t initial values from line 2 - 11 are 0.

Output: To make a mov[illegible]enerate output.txt in the current ("work") directory.

- The format of [illegible] be two integers, indicating i and j as in Figure 2, separated by a comma w[illegible] example:
  ========output.txt=======
  2,3
  ==========
- If your agent waives the right to move, it should write "PASS" (all letters must be in uppercase) in output.txt. For example:
  ========output.txt=======
  PASS
  ==========

5.4. About Stages and Your Agent

All students shall submit their agent named my_player.xxx to Vocareum before the due date. The agent submitted to HW2->**stage1** will be used for grading in the 1st stage. The agent submitted to HW2->**stage2** will be used for grading in the 2nd stage.

For both stages, you are free to choose the AI methods you have learned in the class to build your agent. For example, you may choose to best implement the alpha-beta pruning for the 1st stage and then submit a Q-Learning agent for the 2nd stage. Or, you may build the best agent and use it for both stages.

To facilitate the learning process of your Q-Learning agent, you may use the Game Host to conduct as many training games as you like to train your Q-Learning agent and then submit the well-trained Q-Learning agent for grading. Training can not be performed on Vocareum servers, as there is a limit on how long a given script can run from a student's terminal. Training should be performed on the student's local machine.

6. Notes and Hints

- Please name your program "**my_player.xxx**" where 'xxx' is the extension for the programming language you choose ("py" for python, "cpp" for C++, and "java" for Java). If you are using C++11, then the name of your file should be "my_player11.cpp" and if you are using python3 then the name of your file should be "my_player3.py". Please use only the programming languages mentioned above for this homework. Please Note the highest version of Python that

is offered is Python 3.7.5, hence the walrus operator and other features of higher version Python are not supported.

- To allow for grading the whole class in a reasonable amount of time, each agent is required to complete **60 games** within **5400 seconds** against the 3 AI players when you click the "submit" button on Vocareum stage1 and **30 games** within **1800 seconds** on Vocareum for stage2 (and consequently, within **7200 seconds** in stage1 real grading, and **40 games** within **3600s** in Please note that **running out of time will lead to all points lost.**
- In order to avo ther unexpected situations, **the maximum time for each move is also s** e note that **running out of time for a move will lead to a loss for the o** econd limit per move also doesn't mean that you should fully utilize nds for every move. **Remember that the 5400-second-60-game-per-player policy is also applied on top of it (when you click "submit").**
- The time limit is the total combined CPU time as measured by the Unix **time** command. This command measures pure computation time used by your program, and discards time taken by the operating system, disk I/O, program loading, etc. Beware that it cumulates time spent in any threads spawned by your agent (so if you run 4 threads and use 400% CPU for 10 seconds, this will count as using 40 seconds of allocated time). Your local machine may be more powerful, and thus faster than Vocareum. Therefore, **we highly suggest that you also test your agent on Vocareum later on in this project.**
- **Try first to fully understand the game rules before developing your own code.**
- There may be a lot of Q&A on Piazza. **Please always search for relevant questions before posting a new one. Duplicate questions make everyone's lives harder.**
- **Only submit the source code files (in .java, .py or .spp) and helper files (if any, in .json or .txt). All other files should be excluded.**
- Please submit your homework code through Vocareum (https://labs.vocareum.com/) under the assignment HW2. Your username is your email address. Click "forgot password" for the first time login. You should have been enrolled in this course on Vocareum. If not, please post a private question with your email address and USC ID on Piazza so that we can invite you again.
- You can submit your homework code (by clicking the "submit" button on Vocareum) as many times as you want. Only the latest submission will be considered for grading. After the initial deadline, the submission window will still be open for 5 days. However, a late penalty will be applied as 20% per day if your latest submission is later than the initial deadline.
- Every time you click the "submit" button, you can view your submission report to see if your code works. The grading report will be released after the due date of the project.
- You don't have to keep the page open on Vocareum while the scripts are running.
- If implementing Q-learning, be wary of the size of the state space. Naively storing Q-values for every possible state will lead to an intractably large state space that can not be stored efficiently. Think about the patterns/symmetries that exist in Go that may allow you to reduce the effective size of your state space.
- Be careful and avoid multiple submissions of large files to Vocareum. Vocareum does not allow students to delete old submissions, and in the past, students have run out of space and been unable to use Vocareum until we got in touch with support and asked them to delete files.
- There are some sequences of opening and ending moves that are well studied and known to be strong like opening from the corners first, then edges, and centers at the last. Feel free to explore the internet for tips and tricks of playing Go. Some research papers are
    - Deep Learning and the Game of Go
    - http://erikvanderwerf.tengen.nl/pubdown/thesis_erikvanderwerf.pdf

- Using the techniques learned so far from the class should be sufficient to implement a very good player agent for the game. Students are not required to know or implement more advanced techniques, such as, e.g., deep learning, tensor flow, etc., but if you are interested and have extra time after your basic agent works well, you are welcome to explore on your own.

7. Discussion and Fe

If you have questions a e, the Game Host, or the available AI agents, please feel free to post them on Pi ents must complete their first and second stages on their own. To be fair to all st bout implementation details on Piazza for both stages will be allowed. In addition to this, you should check Piazza frequently for any new announcements for the progress of this project. During the second stage, the teaching staff team may adjust certain parameters of the competition at their discretion.

8. References

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

9. Appendix: An Example QLearningPlayer for the Game of TicTacToe

To assist with the development of your MiniMax/Alpha-Beta or your QlearningPlayer for the Little-Go game, we are also providing you with an example for the game of TicTacToe (not the game of Little-Go). In this example, you can see as an example how a QLearning agent is trained and learns how to play the game of Tic-tac-toe. This example is to assist your development only. Feel free to implement your own program in your own way.

The source code of this example can be obtained from HW2/stage1/resource/startercode on Vocareum.

There are 6 files which are given in this example, and they are:

- Board.py: Tic-tac-toe board, 3 by 3 grid
- RandomPlayer.pyc, SmartPlayer.pyc, PerfectPlayer.pyc (cpython-3.6): Think of them as 3 blackboxes. You do not need to know how things work inside of each player, but it may be helpful to know how they behave (see below).
- QLearner.py: Q-Learning Player that has been implemented for your reference.
- TicTacToe.py: Where all players will be called to play tic-tac-toe games and where your QLearner will be trained and tested. This is similar to the Game Host in Figure 1 (except not using input.txt and output.txt). To play with the TicTacToe games with these agents, you can run the following command line:
  $ python3 TicTacToe.py

The most important functions in QLearner.py include the method movAndLearn(). The parameter GAME_NUM is set to be the number of "exercise" games for training the Q-Learner to learn its Q values for the game. Please see the file QLearner.py for these details. Please also read TicTacToe.py for more details about how the r), and the variable GAME_NUM are used. For example, you would notice that if yo 0 that would train your Q-Learner to be reasonably good to win some games but n sing the number to be GAME_NUM=100000 would enable your Q-Learner to learn player for the Tic-Tac-Toe game. You can change the value of GAME_NUM and ob earning.

You may write your own QLearner.py, but no other files in this directory need to be modified. To get familiar with how the games are played, you can change TicTacToe.py and experiment the process as you like. The three available "opponent" agents are:

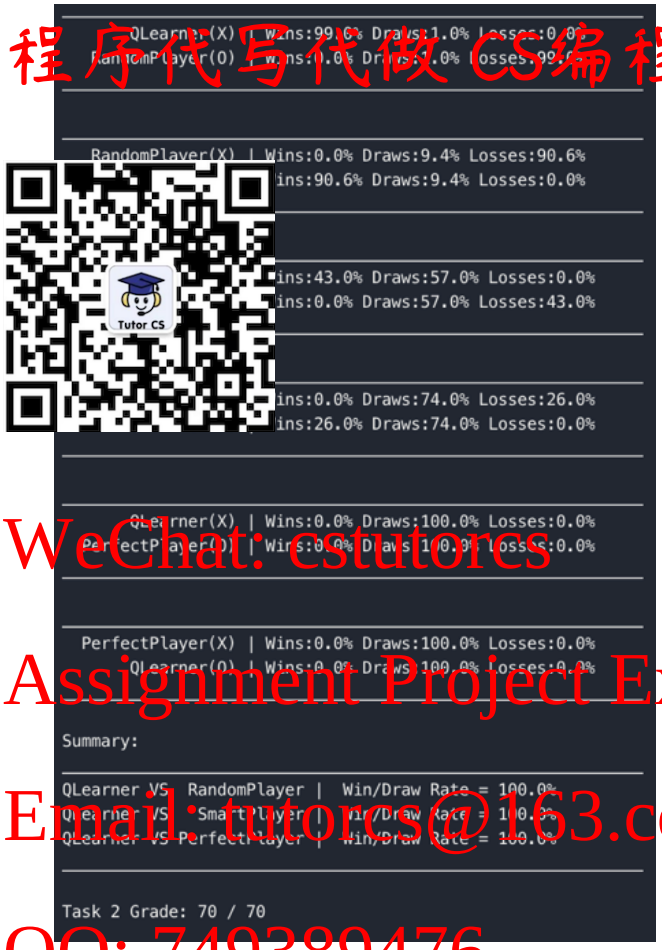
- **RandomPlayer:** Moves randomly
- **SmartPlayer:** Somehow better than RandomPlayer, but cannot beat PerfectPlayer
- **PerfectPlayer:** Never loses

For **Q-Learning,** Recall the formula:

$$Q(s,a) \leftarrow (1\text{- alpha})\ Q(s,a) + \text{alpha}(R(s) + \text{gamma}\ \max_{a'} Q(s',a'))$$

You are free to choose values for all Q-Learning parameters, i.e. the reward values for WIN,DRAW,LOSE, the learning rate ***alpha***, the discount factor ***gamma***, and other initial conditions. *Hint: The rewards will only be assigned for the last action taken by the agent.* Your QLearner agent will be called inside TicTacToe.py to first "learn/train" itself from a number (set by the parameter GAME_NUM) of training games against other agents, and then, the learned/trained agent will be called to play against other agents for competition. Again, please see the details inside the file TicTacToe.py.

After you run python3 TicTacToe.py, the game results will be printed out as follows:



Finally, you are encouraged to experiment and improve the Q-Learner here in terms of speed and performance, and that will help you to get prepared for building your own Q-Learner agent for the game of Little-Go.

We wish you all the very best in this exciting project!