CSCI-561 - Spring 2020 - Foundations of Artificial Intelligence
Homework 2

1. Overview

In this programming homework, you will develop your own AI agents based on some of the learned AI techniques for Search, Game Playing, and Reinforcement Learning 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 AI agents as well as the agents from your fellow classmates. Your agents will be graded and measured by 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-Suicide), and KO. The definitions of these rules are outlined as follows:

Rule1: The Liberty Rule

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*).

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.
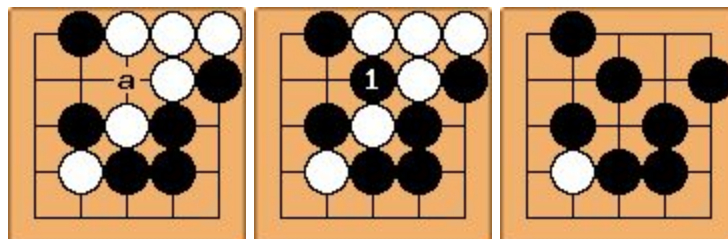
Examples of capturing:
- Example 1. The white stone is captured after Black play at position 1, because its directly orthogonally adjacent points are occupied.
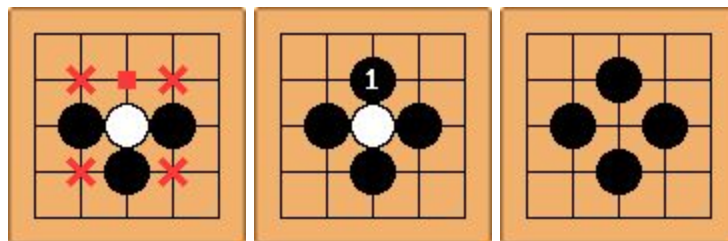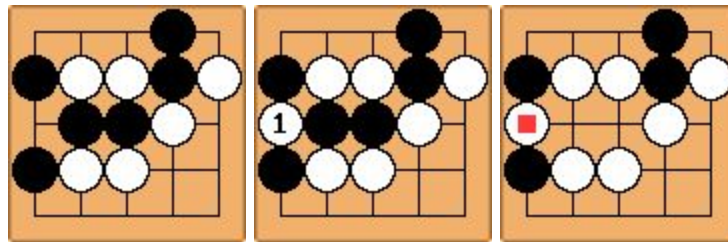
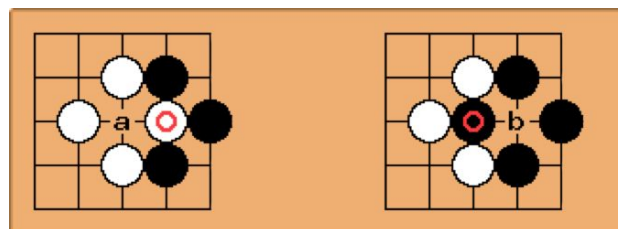- Example 2. The 3 white stones are captured as a connected group.
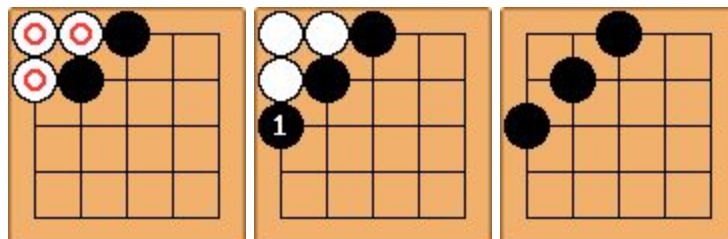
- Example 3. The two groups of white stones are captured.

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


Rule 2: The "KO" Rule

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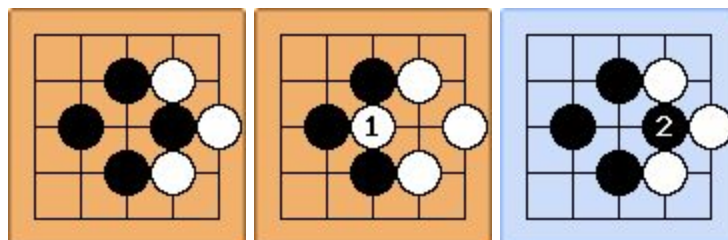
- More examples. KOs need not occur only in the center of the board. They can also show up at the sides or corners of the board, as shown in the diagram below.
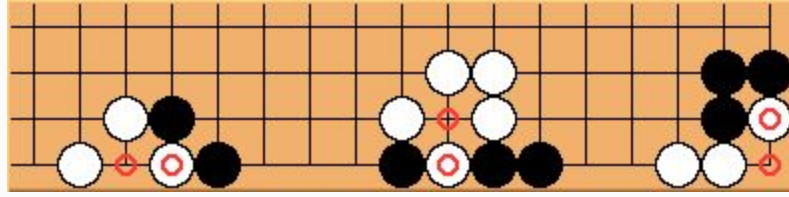
Komi

Because Black has the 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. In this homework (a board size of 5x5), Komi for 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 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 a time limit (See Section 6. Notes and Hints).
- When a player makes an invalid move (invalid stone placement, suicide, violate KO rule, etc.).
- When both players waive their rights to move. Namely, two consecutive passes end the game.
- When the game has reached the maximum steps. In this homework (a board size of 5x5), the maximum steps 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: Black player's final score is the partial area score, while 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 a time limit, s/he loses the game.
    - If a player makes an invalid move (invalid stone placement, suicide, violate KO rule, etc.), s/he loses the game.
    - If the game reaches the maximum steps or 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, on the following board at the end of a game, White's partial area score is 10 and Black's

partial area score is 12 at the end of the game, White is the winner because $10 + 2.5 = 12.5 > 12$.
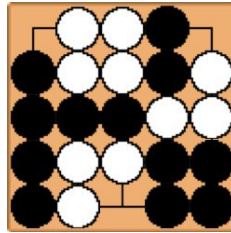
Clarification of the Game Rules

This particular set of rules references several popular rule 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. Go is a very interesting and sophisticated game, please do some more research if you like.

3. Academic Honesty and Integrity

All homework material is checked vigorously for dishonesty using several methods. 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 another agent, either created by the TAs, or created by another student in the class.

4.1 Program Structure

Figure 1 shows the basic program structure. There is one game host and two players in each game. The **Game Host** keeps track of the game process, gets the next move from the players in turn, judges if the proposed move is valid, wipes out the dead stones and finally judges the winner. Each of the two **Players** must output its next move in an exact given format (in a file called output.txt) with the intended point (row and column) coordinates to the Game 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.
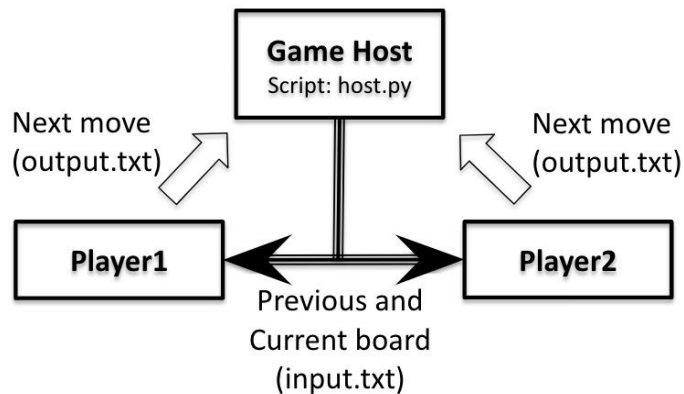
Figure 1: The Program Structure for the Little-Go Game.

4.2 Rule Parameters

The following parameters are adopted in the scope of this homework 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 5x5.
- The maximum movement is $n*n-1$, where n is the size of the game. For example, max movement of a board size 5x5 is 5*5-1=24.
- Komi for White player is $n/2$. For example, Komi of a board 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	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			

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

The host keeps the game board while two players make moves in turn. We will use 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 in which X is black stone and O is white stone.

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 can kill the most stones from the opponent.
- Aggressive Player: looks on the next two possible moves and tries to kill the most stones from the opponent.
- Alphabeta Player: use Minimax algorithm (Depth<=2; Branching factor<=10) with alpha-beta pruning technique.
- QLearningPlayer: use the Q-Learning technique to learn Q values from practice games to make moves intelligently under different game conditions.

Manual Player

To familiarize yourself with the game rules, you can manually play this game on Vocareum as "Player1" against one 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 YouSelf vs. RandomPlayer game. You are highly encouraged to do so, and hope you may win in the end.**

Your Programed Agent Player

You will need to write your own agent in this project. Name your agent as my_player.xx, where xx is the conventional extension used in homework 1 (See Section 6 Notes and Hints), and upload your my_player.xx into the work directory on Vocareum. Then, **when you click the "Submit" button, your uploaded agent will be called by the Game Host to play against the random player, the greedy player, and the aggressive player. The results of these games will be reported for you to view after these games.**

The Game Host

The Game Host integrates Go game with players. During a game play process, the Game Host will do 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 file 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 according to the rules of the game, your agent loses the game.
- Check if the game ends. If so, the winning agent will be declared.

For testing purposes, You player can play this game on Vocareum against one chosen (The Random Player) AI agent. **On vocareum, please click the "Build" button to play YourPlayer vs. RandomPlayer game once you have my_player.xx uploaded into the "work" directory.**

To help you start, 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/$ASNLIB/public/myplayer_play onto your local machine and play there. For example, you can
- Duplicate ranndom_player.py and rename one of them to `my_player3.py`,
- Download build.sh to the same directory, modify line 36: prefix="./",
- Start a game between two random_players on your machine by the following command.
    `$ sh build.sh`

On Vocareum, you can do the same by copying the `random_player.py` into your work directory and rename it as `my_player3.py`, and then click "Build" for the two random_players to play against each other.

5. Project Instructions

5.1. Task Description

Your task is to implement **your agent** in my_player.xx. Note that in the grading and/or the second stage of competition, your agent will play against the agents from other students or TAs, so it is important that your agent inputs and outputs in the exact same format as specified. Other helper files (either .json or .txt) or scripts are acceptable, such as files to store Q-value tables or other types of helper functions, but they are not required. 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 learned from the class for your implementation. In addition to the regular game playing techniques, such as MinMax or Alpha-beta pruning, you are also highly encouraged to use reinforcement learning in this homework in order to win the honorary competition against other students in the second stage. You are encouraged to research other possible methods or Go tactics that may help you defeat your opponent.

In your implementation, ***please do not use any publicly available library calls for minmax, alpha-beta, or Q- Learning***. 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

There will be 2 stages in this project, the 1st stage is for the course grade, and the 2nd stage is for the competition to win a Google interview or other Prizes (for honorary purpose).

The First Stage (100 pts)

In the first stage, you will be graded by playing 100 times against each of the four existing AI players (random, greedy, aggressive, and alpha-beta) in the table below. 50 times as Black and 50 times as White. Note that when you click the "submit" button, your agent will play against only the random, greedy, and aggressive agents. The agent alpha-beta is reserved for the grading purpose.

Grading Rubrics

WinRate	Opponent Agent and Availability				
	1st Stage (for your course grade)				2nd Stage (for Google Interview)
	Available for Testing on Vocareum			Reserved for Grading	Available for testing and preparing the competition
	Random	**Greedy**	**Aggressive**	**Alpha-beta**	**Q-Learning Agent**
>=90%	25pts	25pts	25pts	25pts	For your tests
>=70%	15pts*winRate	15pts*winRate	20pts*winRate	20pts*winRate	For your tests
<70%	0pts	0pts	10pts*winRate	10pts*winRate	For your tests

Example:
- VS random player: win 95 games out of 100 games → 25 pts.
- VS greedy player: win 85 games out of 100 games → $15 * 0.85 = 12.75$ pts
- VS aggressive player: win 80 games out of 100 games → $20 * 0.8 = 16$ pts
- VS alpha-beta player: win 67 games out of 100 games → $10 * 0.67 = 6.7$ pts

So the total points for the 1st stage will be: $25 + 12.75 + 16 + 6.7 = 60.45$ pts.

The Second Stage (for extra pts)

Students' agents that get >=**80 points** will enter the second stage. In the second stage, your agent will compete against other agents that have also entered this stage to win the Google Interview and the honorary prizes.
- Students' agents will play against each other in the second stage.
- For student A and student B, they will play 11 games. Whoever wins more in the 11 games is the winner of A and B, counted as one **WIN** for the winning player.
- The extra chance to move first is decided randomly.
- Each student will play with **all the other students** entering this second stage.
- The student will be ranked according to the WIN counts.

Grading Rubrics (for the 2nd stage)

Rank	Extra Points	Additional Prize for 1-3
1-3	20pts	An Interview at Google (and maybe at Honda as well), plus a chance to ask for a recommendation letter from Professor Wei-Min Shen
4-10	15pts	
10-20	10pts	
After 20	5pts	

5.3. Input and Output Format

The format of input and output of your agent must follow the exact same format, otherwise, your agent will never win any game. The input and output format is described as follows. End-of-line character is LF (since Vocareum is a Unix system and follows the Unix convention).

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 game board, with 5 lines of 5 values. This is the state after your last move. (Black=1, White=2, Unoccupied=0);
- Line 7-11: Description of the current state of game board, with 5 lines of 5 values. This is the state after your opponent's last move. (Black=1, White=2, Unoccupied=0);
  At the beginning of a game, the default initial values from line 2 - 11 are 0.

For example:
```
========input.txt=======
2
00110
00210
00200
02000
00000
00110
00210
00200
02010
00000
======================
```

Output: To make a move, your agent should generate output.txt in the current ("work") directory.

- The format of placing a stone should be two integers, indicating i and j as in Figure 2, separated by a comma without whitespace. For example:
```
========output.txt=======
2,3
======================
```

- If your agent waives the right to move, it should write a "PASS" (all letters are capital) in output.txt. For example:
```
========output.txt=======
PASS
======================
```

5.4. About Stages and Your Agent

All students shall submit their one agent named my_player.xxx to Vocareum before the due date and the submitted agent will be used for grading in the 1st stage. After the grading in the 1st stage is completed, we will inform all those students whose agent has been advanced into the 2nd stage. At that time, the students who are in the 2nd stage will have a one-time chance if they want to update their agent for the competition before the 2nd stage starts.

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 submit the well-trained Q-Learning agent for grading or for competition.

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 the programming languages mentioned above for this homework.
- In order to better recognize who you are in the later games, **please put your USC id and name as a comment at the first line of your my_player.xxx.** The format can be "1234567890:Your Name".
- To allow grading the whole class in a reasonable amount of time, each program is suggested to finish within *60 minutes* (to complete **300 games** against AI players when you click the "submit" button on Vocareum). Please note that **running out of time may lead to all points lost.**
- In order to avoid infinite loop and other unexpected situations, **the maximum time for each move is 10 seconds.** Please note that **running out of time may lead to points lost for the ongoing game.** This doesn't mean you should fully utilize these 10s. **Please note the 60-minute-300-game-per-player policy is on the top of it (when you click "submit").**
- The time limits mentioned above are the actual CPU time on Vocareum. Your local machine may be more powerful, thus faster than Vocareum. **It's highly suggested to make your plan of testing on Vocareum at the later stage of 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. Duplicated questions make everyone's life hard.**
- Some source code (e.g. host.py and random_player.py) are provided for your reference. **You are NOT allowed to directly copy/use the source code.**
- **Only submit the source code files (in .java, .py or .cpp) 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. Useranme 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 will 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. Late penalty will be applied as specified in the syllabus if your latest submission is after the deadline.
- Every time you click the "submit" button, you can view your submission report to see if your code works, while the grading report will be released after the due date of the project.
- You don't have to keep the page open while the scripts are running.

7. Discussion and Feedback

If you have questions about the Little-Go game, the Game Host, or the available AI agents, please feel free to post them on Piazza. However, all students must complete their first and second stage on their own. To be fair for all students, no discussion about implementation details on Piazza for both stages would be allowed. In addition, you should check Piazza frequently for any new announcements for the

progress of the project. During the second stage, the teaching staff team may adjust certain parameters of the competition according to the actual situation.

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 the development of your MinMax/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 learn how to play the game of Tic-tac-toe. This example is to assist your development only, and 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 `move()` and `learn()`. 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 methods `move()`, `learn()`, and the variable `GAME_NUM` are used. For example, you would notice that if you set `GAME_NUM=1000` that would train your Q-Learner to be reasonably good to win some games but not all games. But increasing the number to be `GAME_NUM=100000` would enable your Q-Learner to learn and become the perfect player for the Tic-Tac-Toe game. You can change the value of `GAME_NUM` and observe its effects on Q-learning.

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 lose

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.

```
      QLearner(X) | Wins:99.0% Draws:1.0% Losses:0.0%
   RandomPlayer(O) | Wins:0.0% Draws:1.0% Losses:99.0%


   RandomPlayer(X) | Wins:0.0% Draws:9.4% Losses:90.6%
      QLearner(O) | Wins:90.6% Draws:9.4% Losses:0.0%


      QLearner(X) | Wins:43.0% Draws:57.0% Losses:0.0%
   SmartPlayer(O) | Wins:0.0% Draws:57.0% Losses:43.0%


   SmartPlayer(X) | Wins:0.0% Draws:74.0% Losses:26.0%
      QLearner(O) | Wins:26.0% Draws:74.0% Losses:0.0%


      QLearner(X) | Wins:0.0% Draws:100.0% Losses:0.0%
  PerfectPlayer(O) | Wins:0.0% Draws:100.0% Losses:0.0%


  PerfectPlayer(X) | Wins:0.0% Draws:100.0% Losses:0.0%
      QLearner(O) | Wins:0.0% Draws:100.0% Losses:0.0%

Summary:

QLearner VS  RandomPlayer |  Win/Draw Rate = 100.0%
QLearner VS   SmartPlayer |  Win/Draw Rate = 100.0%
QLearner VS PerfectPlayer |  Win/Draw Rate = 100.0%


Task 2 Grade: 70 / 70
```

Finally, you are encouraged to make experiments and improvements of 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 in the first or the second stage.

We wish you the best in this exciting and rewarding project!