In Tic-Tac-Toe game, there are two players: A and B. The board is 3x3 and player A puts down X on the board and then player B put down O on the empty place of the board. The players play alternatively. The following is a snapshot of Tic-Tac-Toe playing.

	X	O
X	O	
O	X	

If a player fills out a row, a column or a diagonal of the board, she wins. In the figure, player A wins.

1. **Implement Minimax algorithm for Tic-Tac-Toe game, and answer to the followings with your program.**

   (a) Where is the best place for the first play of Player A?

➔ 최초의 A 의 모든 수에 대해 minimax algorithm 은 무승부를 예측한다. 이때 board 의 좌상단부터 탐색이 진행되므로 (1,1)을 최선의 play 로 결정한다.

   (b) Let's assume that Player A put down X to where you recommend in (a), where is the best place for Player B, next?

➔ (2,2) 이외의 위치에 두었을 경우 모두 패배하게 된다. 따라서 (2,2)의 위치가 B 의 최선의 play 다.

2. **Implement alpha-beta pruning for Tic-Tac-Toe game, and answer to the followings with your program.**

   (a) Where is the best place for the first play of Player A?

➔ Minimax 와 마찬가지로 (1,1)을 최선의 play 로 결정한다. 이때 minimax 보다 탐색의 회수가 줄어듦을 알 수 있다.

   (b) Let's assume that Player A put down X to where you recommend in (a), where is the best place for Player B, next?

➔ Minimax 와 마찬가지로 (2,2)를 최선의 paly 로 결정한다.

Now, let's expand the board to 4x4.

3. **Implement Minimax algorithm for Tic-Tac-Toe game, and answer to the followings with your program.**

   (a) Where is the best place for the first play of Player A?

   (b) Let's assume that Player A put down X to where you recommend in (a), where is the best place for Player B, next?

   (c) If your program fails to find out answers of (a) and (b), why do you think that it fails?

➔ Minimax algorithm 은 가능한 모든 경우를 탐색하게 된다. 따라서 4x4 의 board 에서는 runtime 의 한계로 인해 시간내에 답을 구할 수가 없다.

4. **Implement alpha-beta pruning for Tic-Tac-Toe game, and answer to the followings with your program.**

   (a) Where is the best place for the first play of Player A?

(b) Let's assume that Player A put down X to where you recommend in (a), where is the best place for Player B, next?

(c) If your program fails to find out answers of (a) and (b), why do you think that it fails?

➔ Alpha-beta pruning 을 통해 탐색하는 범위를 줄였음에도 여전히 실행시간이 오래 걸려 답을 구할 수 없다.

5. Implement Monte Carlo Tree Search for Tic-Tac-Toe game, and answer to the followings with your program. Perform only 100 random tries for each play.

(a) Where is the best place for the first play of Player A?

➔ Random play 의 무작위성으로 인해 매번 다른 결과가 나오지만 (2,2)의 위치가 평균적으로 가장 높은 승률을 보인다.

(b) Let's assume that Player A put down X to where you recommend in (a), where is the best place for Player B, next?

➔ 매번 결과가 다르게 나온다.

6. Implement Monte Carlo Tree Search for Tic-Tac-Toe game, and answer to the followings with your program. Perform only 10000 random tries for each play.

(a) Where is the best place for the first play of Player A?

➔ (3,3)

(b) Let's assume that Player A put down X to where you recommend in (a), where is the best place for Player B, next?

➔ (4,0)

(c) Do you think that the two places are better than those found in (5)? Why?

➔ 두 경우 모두 상대방은 random play 를 한다고 가정하였다.
우선, 100 회의 random tries 에서는 유망한 경로에 대한 탐색이 깊숙이 이뤄지지 않는다. 최초의 빈 board 의 상태를 game tree 의 root 로 생각했을 때 첫 수를 두는 경우의 수가 16 가지이다. 따라서 100 회의 random try 는 부족함을 알 수 있다.
10000 회의 random try 는 root 노드에서 16 가지의 경우의 수에 대해 대부분 200 회 정도의 탐색이 일어나고, 유망한 노드에 대해서는 1000~6000 회 정도의 탐색이 발생했다. 따라서 100 회의 try 보다는 훨씬 충분한 탐색이 일어났고, monte carlo search 는 매 탐색 시, 새로운 노드를 span 하므로 더욱 깊숙이 탐색이 발생했다고 볼 수 있다.
따라서, 10000 회의 random try 가 탐색한 위치가 더욱 정확하다고 볼 수 있다.

Submit the source code for 3, 4, 6, and the answer to the questions.