How to Run:

Simply open up the index.html file in your favorite browser. (Tested on Chrome) You will see buttons to run the different games, as well as a speed input. Speed controls the amount of time between turns. You can run bulk games by clicking the bottom row of buttons. The board output during bulk games will not be displayed, but you will see the aggregated results.

1.) Architecture

Architecture still consists of a Game object, which keeps track of player turns, board state, and whether the game has finished or not. New to the Game object is the addition of being able to query how to follow certain strategies. Like blocking condition, win conditions, and so on.

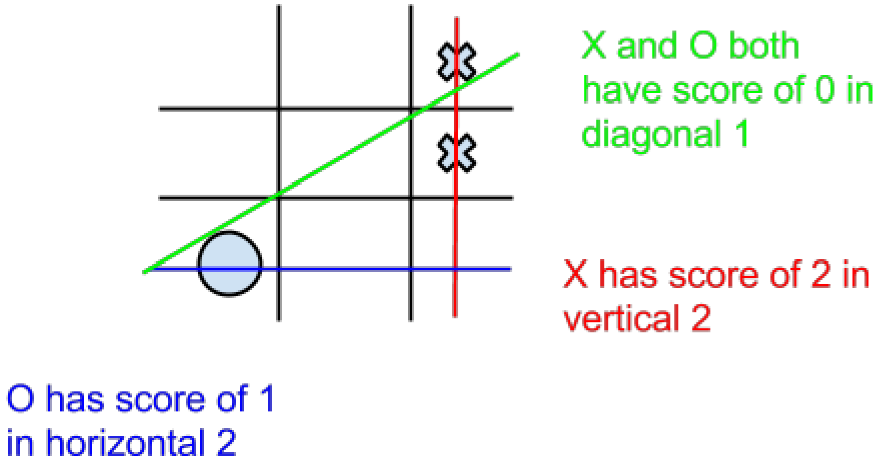
There is a naive player who once again chooses spaces at random. Nothing has changed in this regard since Project #2.

The smart agent, however, has. It has a prioritized set of strategies to follow, and now only one function that implements them. These strategies are “win”, “block”, and “build”. It will check the game information to see the conditions for winning, blocking, and building.

Strategy to Representation

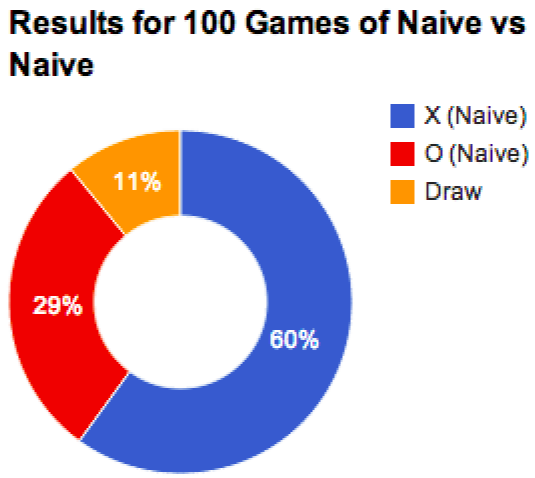
These conditions are formatted in the same representation used in project #2. It worked beautifully here. All the strategies end up boiling down to a certain score which corresponds to the scoring representation used earlier. For example a “win” strategy is to look for scores that are near the win condition for the game, and then open spaces in those rows.

Here’s how scoring works, as a refresher:

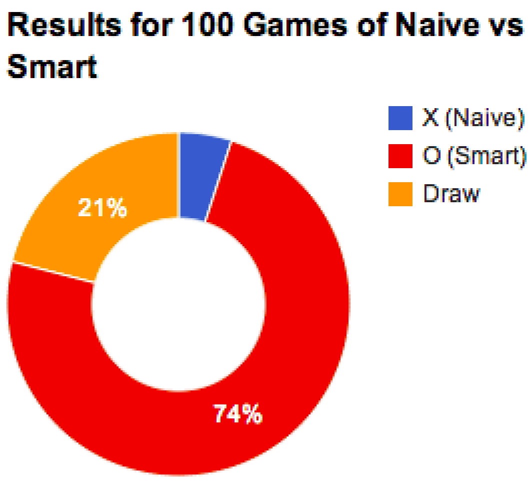

X and O both have score of 0 in diagonal 1
X has score of 2 in vertical 2
O has score of 1 in horizontal 2

2.) Results and Analysis

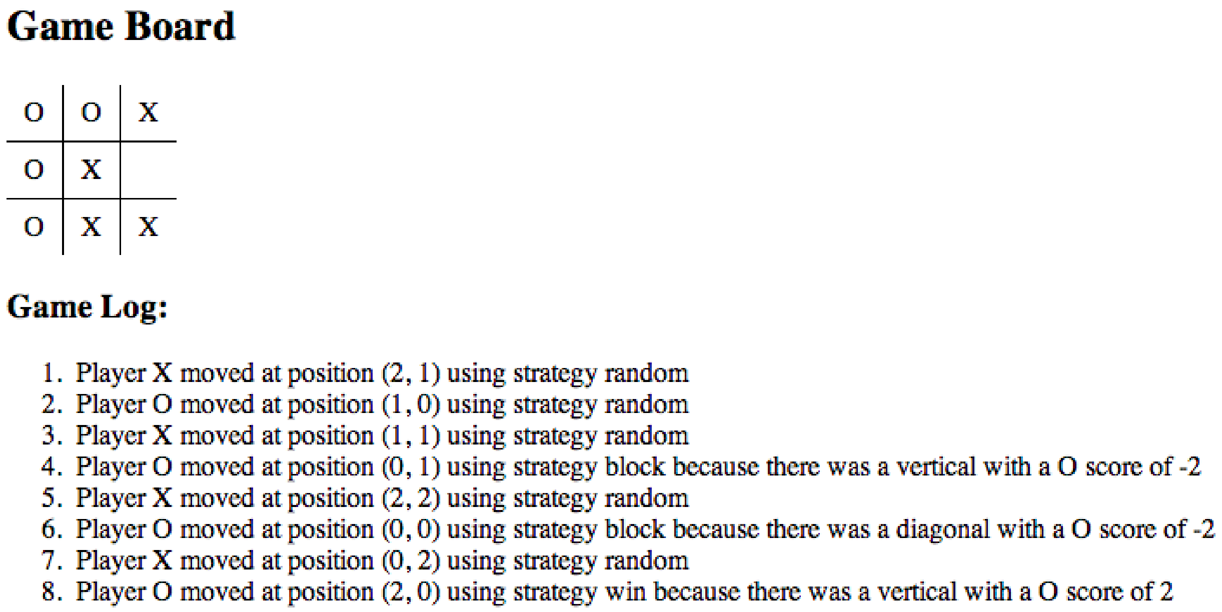
Naive vs Naive: Again uninteresting with regards to this project. The naive agent is purely random. X wins a bit more often than O, probably because of moving first.



Naive vs Smart: Draws and smart wins. Mostly smart wins. Out of 100 games, I saw 21 draws, 5 wins for naive, and 74 wins for smart.



Here’s an example of a Naive vs Smart game:



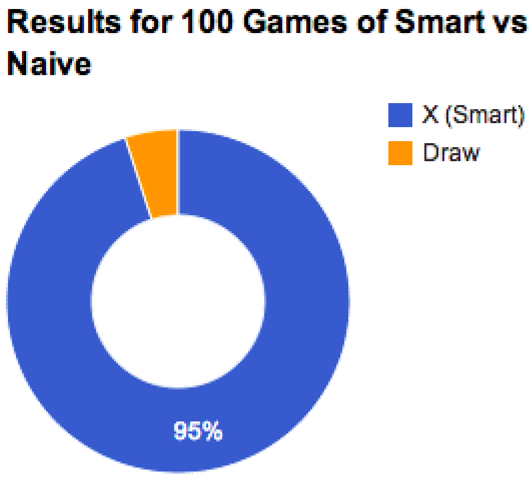
Game Board

O	O	X
O	X	
O	X	X

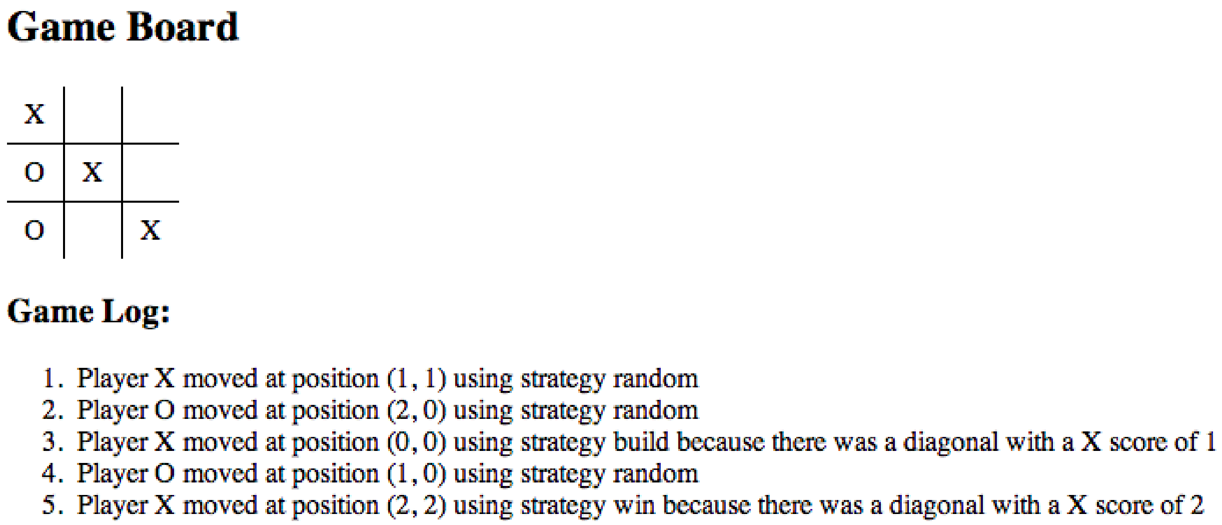
Game Log:

1. Player X moved at position (2, 1) using strategy random
2. Player O moved at position (1, 0) using strategy random
3. Player X moved at position (1, 1) using strategy random
4. Player O moved at position (0, 1) using strategy block because there was a vertical with a O score of -2
5. Player X moved at position (2, 2) using strategy random
6. Player O moved at position (0, 0) using strategy block because there was a diagonal with a O score of -2
7. Player X moved at position (0, 2) using strategy random
8. Player O moved at position (2, 0) using strategy win because there was a vertical with a O score of 2

Smart vs Naive: Again, smart almost always wins. The naive agent can almost never make enough moves to draw the game. The results for 100 games were 5 draws, 95 wins for X (Smart), and 0 wins for O (Naive).


Results for 100 Games of Smart vs Naive

Here’s an example of a Smart vs Naive game:

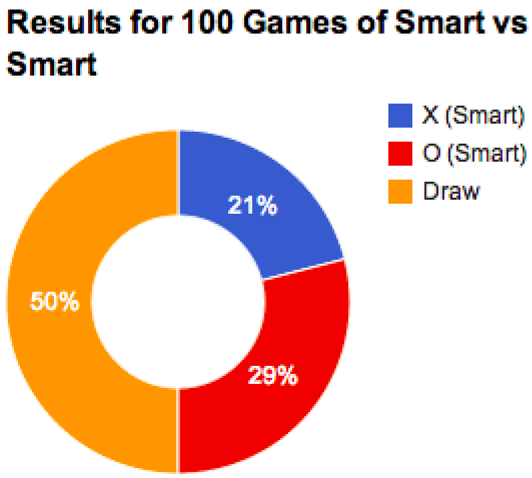

Game Board

X		
O	X	
O		X

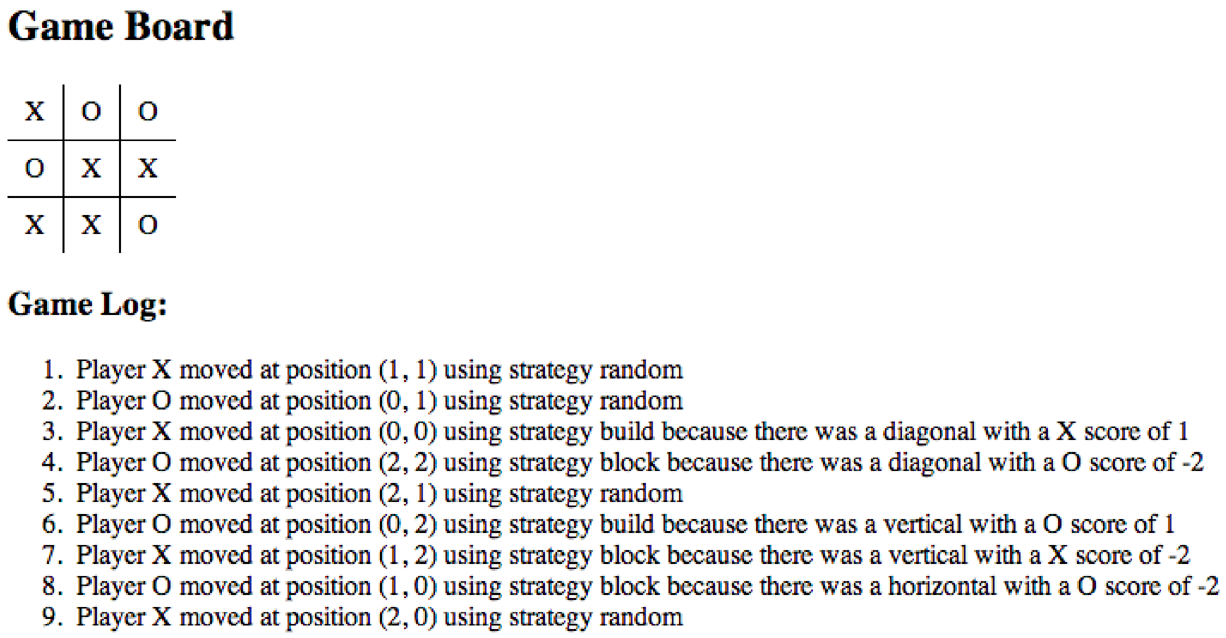
Game Log:

1. Player X moved at position (1, 1) using strategy random
2. Player O moved at position (2, 0) using strategy random
3. Player X moved at position (0, 0) using strategy build because there was a diagonal with a X score of 1
4. Player O moved at position (1, 0) using strategy random
5. Player X moved at position (2, 2) using strategy win because there was a diagonal with a X score of 2

Smart vs Smart: Had the most draws, which was expected based on the results of Project #2.



Sometimes wins occur, as in Project #1 and #2, because of inadvertent trap making. This has already been explored at length in previous reports. A majority of the strategies followed (discounting initial random moves) were blocking moves.



Game Board

X	O	O
O	X	X
X	X	O

Game Log:

1. Player X moved at position (1, 1) using strategy random
2. Player O moved at position (0, 1) using strategy random
3. Player X moved at position (0, 0) using strategy build because there was a diagonal with a X score of 1
4. Player O moved at position (2, 2) using strategy block because there was a diagonal with a O score of -2
5. Player X moved at position (2, 1) using strategy random
6. Player O moved at position (0, 2) using strategy build because there was a vertical with a O score of 1
7. Player X moved at position (1, 2) using strategy block because there was a vertical with a X score of -2
8. Player O moved at position (1, 0) using strategy block because there was a horizontal with a O score of -2
9. Player X moved at position (2, 0) using strategy random

A majority of meaningful strategies followed was indeed blocking.

Questions for the Agent

Some questions the agent could answer is “when do you block?” and it would say when there is a score of -2 by introspecting into the precedents of “block”. Or you could ask it why it moved to a certain spot, and it could tell you it was because it was following some strategy.

Differences with Project #2

I abstracted the knowledge of strategies out, leaving only string representations of the strategies in the player object. The player object must now query for knowledge on each strategy to figure out what to look for in the representation.