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.

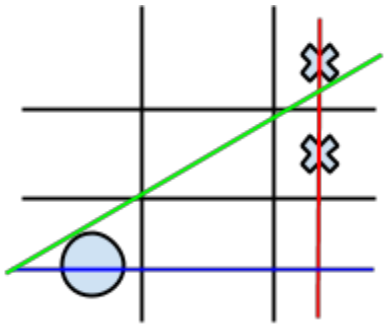
1.) Architecture

Architecture still consists of a Game object, which keeps track of player turns, board state, and whether the game has finished or not.

There is a naive player who once again chooses spaces at random. Nothing has changed in this regard since Project #1.

The smart agent, however, has. Although there is still a ruleset it follows, the information presented to the agent is different. No longer does it evaluate the entire board state but rather it is fed a summary of "scores" from the Game object. It looks through these scores and finds potential "slots" to play in. The most desirable slots being those that will bring a victory, then slots that will block an opponent, then slots that will build towards victory. This is a similar ruleset as Project #1 (with some rules taken out because they weren't using the board knowledge) but now the agent makes decisions based off of a representation of the board.

Here's how scoring works:



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

The smart agent receives a representation that looks like this:

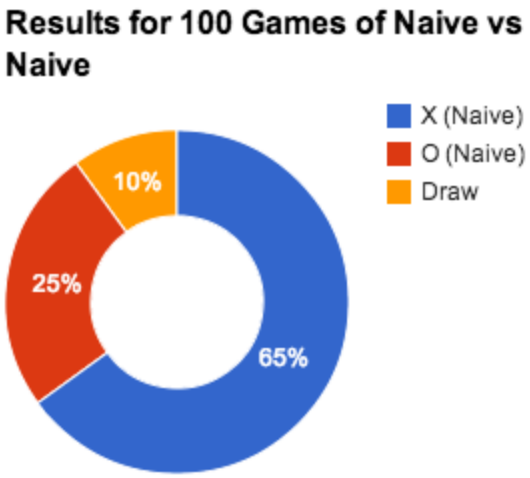
Player	Slot	Score
“X”	Horizontal, 1	1
“X”	Vertical, 2	2
“O”	Diagonal, 1	0
“O”	Vertical, 2	-2

...And so on until all slots are represented for all players.

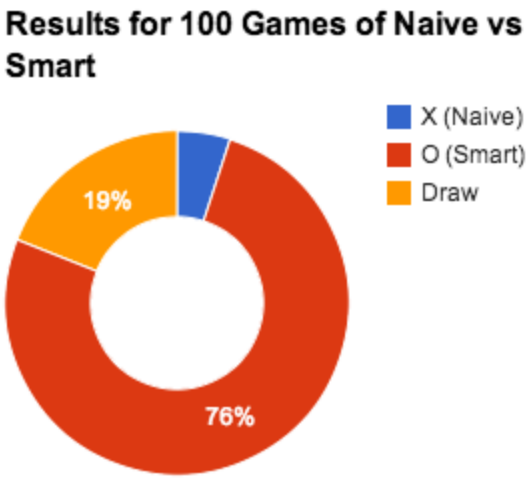
The agent knows of a winning condition provided by the game. In the case of TicTacToe, it’s a score of 3. If the board was to be expanded, however, the agent would still know to look for winning moves, so long as the winning score information was provided by the game.

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 19 draws, 4 wins for naive, and 76 wins for smart.



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

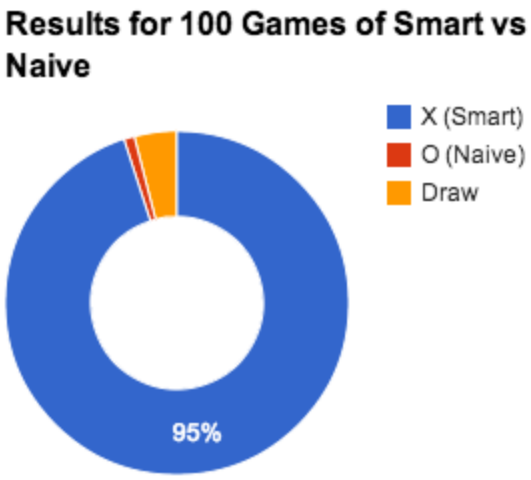
Game Board

O	X	
O	X	X
O	O	X

Game Log:

1. Player X moved at position (1, 2) using rule random
2. Player O moved at position (2, 1) using rule 3
3. Player X moved at position (1, 1) using rule random
4. Player O moved at position (1, 0) using rule 1 because there was a horizontal with a O score of -2
5. Player X moved at position (2, 2) using rule random
6. Player O moved at position (0, 0) using rule 1 because there was a diagonal with a O score of -2
7. Player X moved at position (0, 1) using rule random
8. Player O moved at position (2, 0) using rule 0 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 4 draws, 95 wins for X (Smart), and 1 win for O (Naive).



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

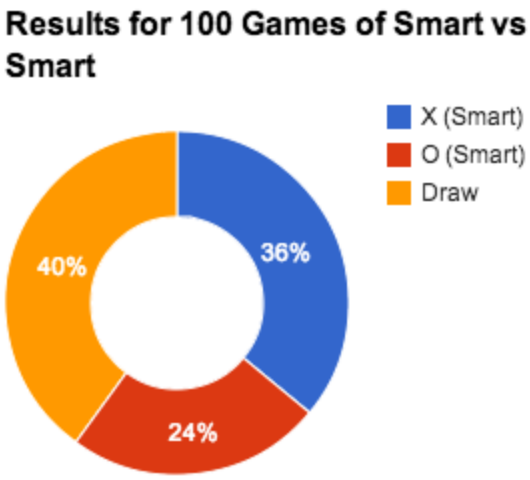
Game Board

		X
O		X
	O	X

Game Log:

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

Smart vs Smart: Had the most draws, which was expected based on the results of Project #1, however unlike Project #1 (probably due to the removal of the "start" rules), there were a decent chunk of wins for X and O.



Interestingly, most of the moves made in a Smart vs Smart game (past the initial ones) are blocking moves. In the process of blocking, new winning plays open up to be blocked by the other player, and so on. Because they are both trying to play optimally, the natural ending is a draw, however sometimes in the process of blocking an agent may setup a “trap” on accident. Here’s an example of a typical draw game:

Game Board

X	O	X
X	O	X
O	X	O

Game Log:

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

And here is a game where a “trap” got constructed inadvertently:

Game Board

O	X	
O	O	X
X	X	O

Game Log:

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

As you can see, by simply moving to block X, O creates a conundrum in which X cannot block O from winning. This is not intentional, but rather a lucky side effect of reacting to the board states

Differences with Project #1

After reviewing I thought that the additions I made at the end of Project #1 (prioritize center and corners) were not applicable to this project, because I didn’t have a way of favoring them in my representation. So I took those rules out from the ruleset, in the interest of keeping with the assignment.

The biggest difference is now it is easier to see *why* moves were taken by the smart agent, due to the more representation based architecture. The agent performance is a little worse now as well because the initial move is random for the smart agent.