Our Magical Adventure through Unity and Tic Tac Toe

Alexander Feng and Jamie Ip

April 29, 2020

What we accomplished

- Documentation for Gamesman Unity
- Fixing Gamesman Unity bug of remote solver unable to finish games
- Identified Gamesman Unity bug of not running after building because of http/https request issues
- Miscellaneous other Gamesman Unity bug fixes
- Python solver for 3x3x3 Tic Tac Toe, unfortunately takes hours to run
- Python solver for 3x3x2 Tic Tac Toe, optimized to solve the initial position in 0.004 seconds by indexing from a pre-generated file as opposed to 1984.034 seconds solving using memoization and symmetries
- Hash function for converting 3D Tic Tac Toe positions to an integer index, slightly redundant in that it requires storing unvisited positions
- Python solver for Wild Tic Tac Toe, a variant of Tic Tac Toe where players can choose to use X or O on each move
- Analysis on the remoteness of Tic Tac Toe, Misère Tic-Tac-Toe, and Wild Tic Tac Toe versus the average number of pieces on the board

Gamesman Unity

(a) When we picked up Gamesman Unity from Github, we were pleasantly surprised to find it in mostly working order. The graphics were finished, but the local and remote solvers could only solve 3x3x1 Tic Tac Toe. The project also had zero documentation and quite a few bugs.

(b) We began by adding documentation to Gamesman Unity to both better understand the project and to help any future students who want to continue the project. Our goal for the semester was to make it capable of fully solving 3x3x3 Tic Tac Toe (with a hole in the center) to fully take advantage of Unity's 3D interface.

After creating a 3x3x3 solver, we realized the scale this would entail (3^{26} = 2.54e12 positions), so we turned to 3x3x2 Tic Tac Toe instead. 3x3x2 proved to be much more manageable, but still took 30 minutes to fully solve. So any position could be solved instantly, we wrote a hash function to convert positions into integers, then stored the value of each position, indexed by hash, into a file. The file is unfortunately very large (369 MB), as our indexing scheme requires every possible position to be stored, even those that are unvisited. Still, this allows for instantaneous solving.

At this point, the only issue remaining was that our Unity game wasn't able to successfully send requests to the nyc server without bugs. Additionally, we had to write in a new file that would handle our 3x3x2 board. After figuring out how to fix those bugs, we encountered another problem where you can't request through http if you're being hosted by https. This lead to some frustration as we didn't really understand how web requests worked.

(c) At its current state, Gamesman Unity is located on the GamesCrafters github and can be cloned for immediate startup. For editing code, using a normal text editor is fine. However, modifying games elements requires using the Unity Engine Editor which can be found online. As a worthy note, the code is all written in *C#*. (Code hasn't been pushed yet, but should be within the next week or so.)

Currently, the only remaining bugs for Gamesman Unity are that:

- Putting in a board for a non existing remote solver crashes the unity game.

- The local solver is still an option even when such a solve is impractical.

Overall Improvements to be made are:

- to fix the above bugs
- make the UI more appealing
- perhaps make animations
- Simplify the code
- optimize 3D TTT hash function
- reduce the amount of space 3D TTT takes (ie. removing impossible to achieve boards)

Tic Tac Toe

(a) We also decided to create a solver for Wild Tic Tac Toe, a variant of Tic Tac Toe where players can choose whether to place an X or O every turn. Inspired by Dan's idea of analyzing remoteness compared to pieces on the board, we attempted a brief analysis of comparing Tic Tac Toe, Misère Tic Tac Toe, and Wild Tic Tac Toe.

(b) Remoteness and # of Positions
(Symmetries are on for all games)

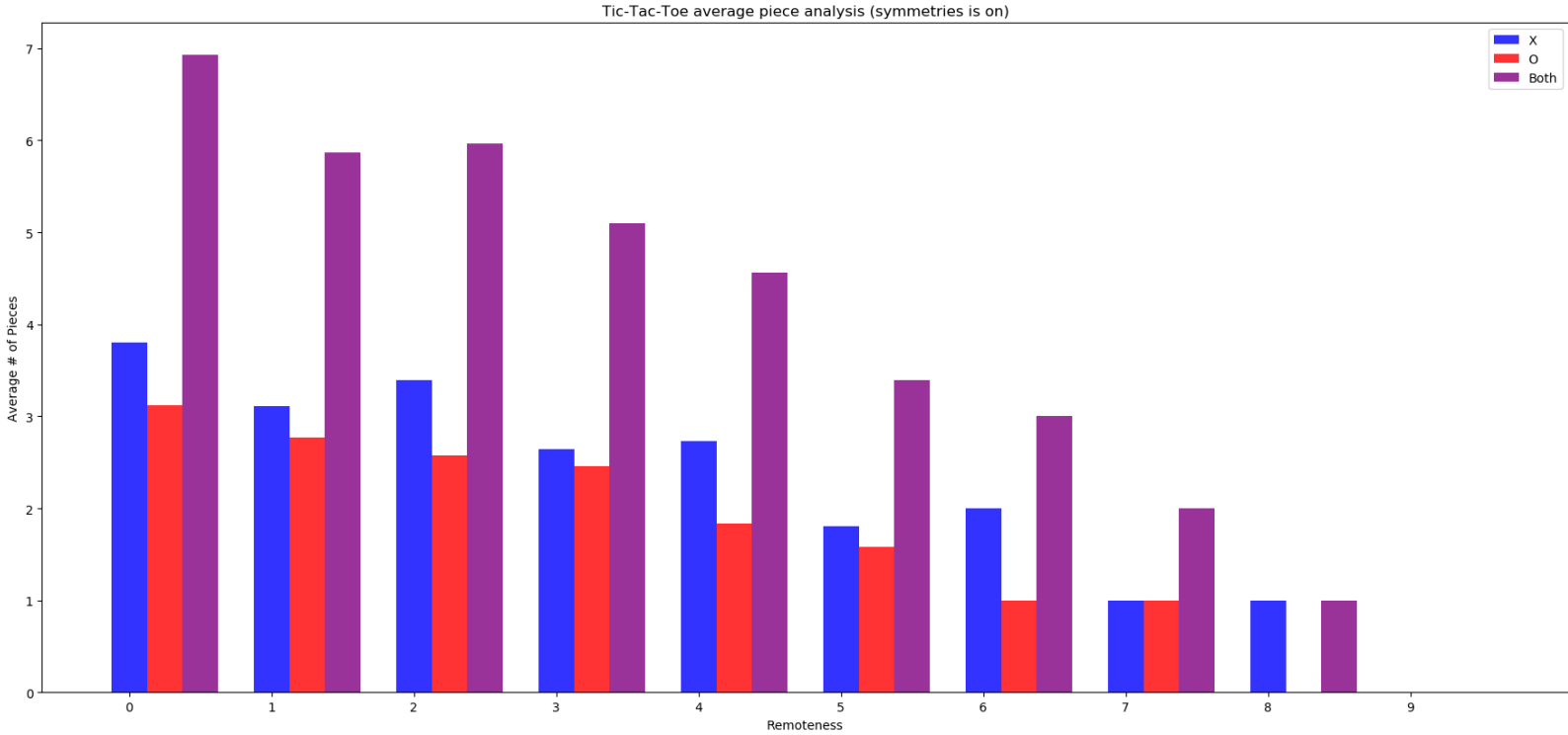
	Tic Tac Toe				Misère Tic Tac Toe				Wild Tic Tac Toe			
R	Win	Lose	Tie	Total	Win	Lose	Tie	Total	Win	Lose	Tie	Total
9	0	0	1	1	0	0	1	1	0	0	0	0
8	0	0	3	3	2	0	1	3	0	0	4	4
7	0	0	5	5	0	4	7	11	1	0	8	9
6	0	0	21	21	32	0	6	38	0	2	32	34
5	18	0	18	36	0	61	31	92	2	0	34	36
4	0	17	35	52	136	0	20	156	0	2	32	34
3	51	0	27	78	0	100	43	143	10	0	18	28
2	0	72	27	99	116	0	17	133	0	8	10	18
1	321	0	11	332	0	39	11	50	1414	0	5	1419
0	0	135	3	138	135	0	3	138	0	1106	6	1112
Total	390	224	151	765	421	204	140	765	1427	1118	149	2694

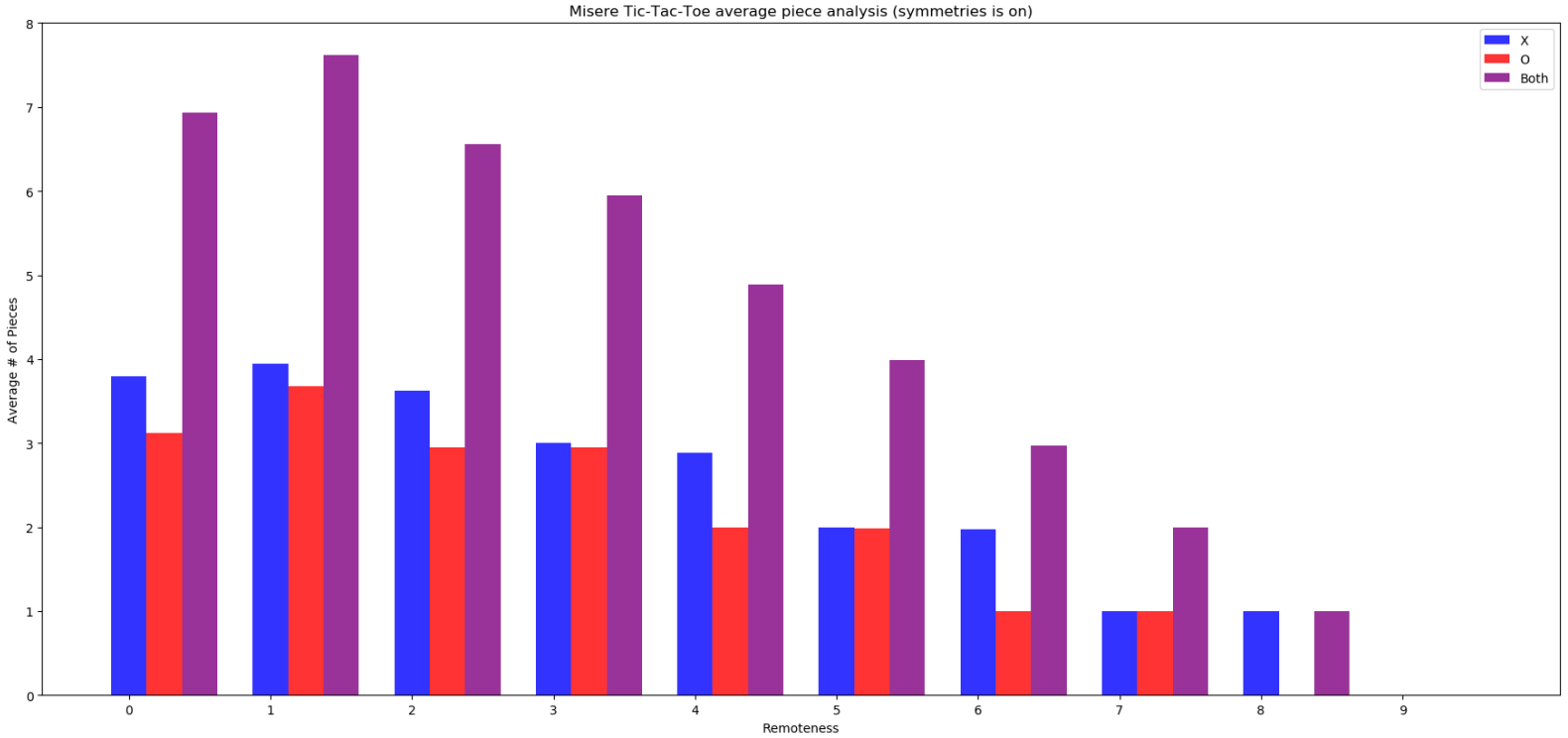
Remoteness and Average # of Pieces

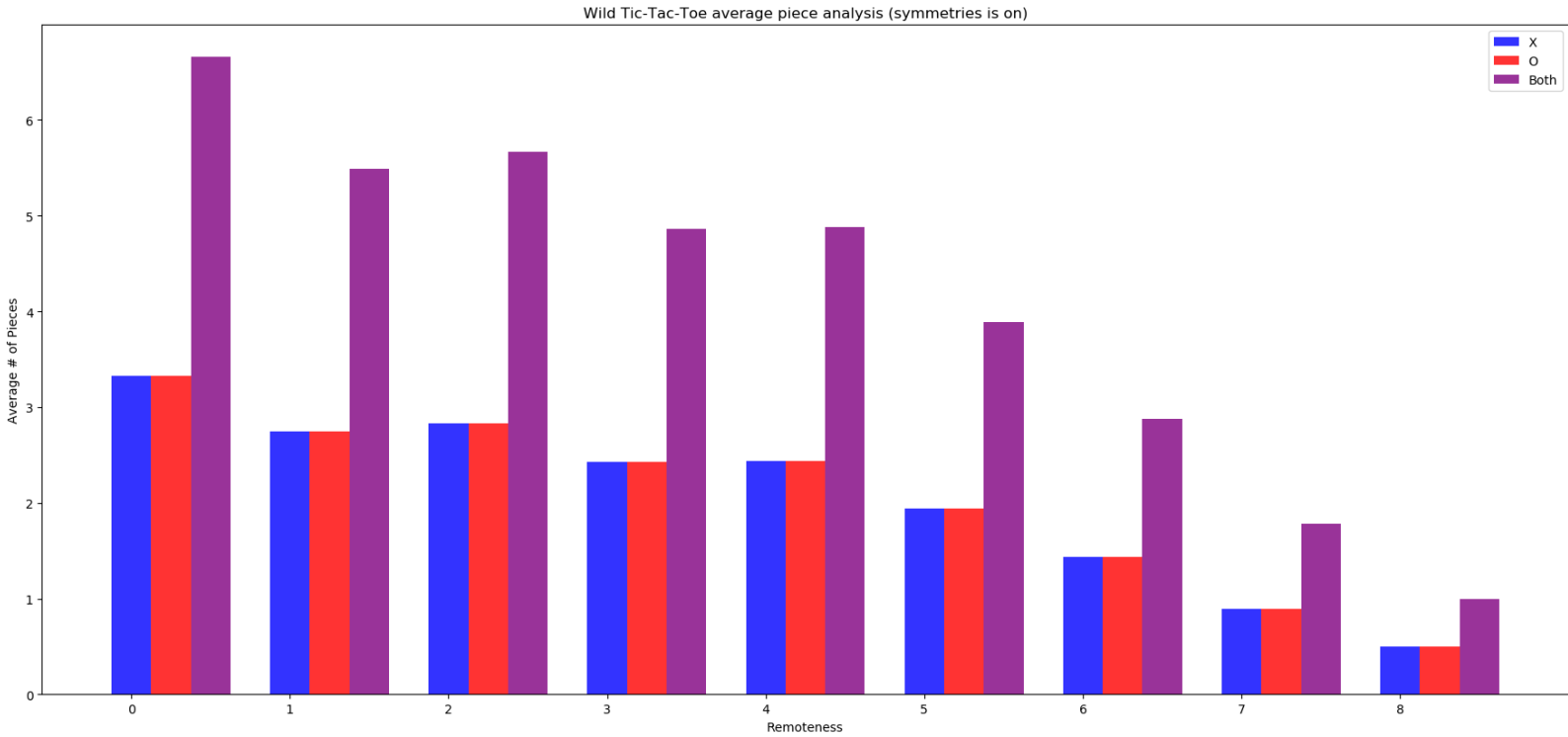
	Tic Tac Toe			Misère Tic Tac Toe			Wild Tic Tac Toe		
R	Avg X	Avg O	Avg All	Avg X	Avg O	Avg All	Avg X	Avg O	Avg All
9	0.0	0.0	0.0	0.0	0.0	0.0		N/A	
8	1.0	0.0	1.0	1.0	0.0	1.0	0.5	0.5	1.0
7	1.0	1.0	2.0	1.0	1.0	2.0	0.89	0.89	1.78
6	2.0	1.0	3.0	1.97	1.0	2.97	1.44	1.44	2.88
5	1.81	1.58	3.39	2.0	1.99	3.99	1.94	1.94	3.89
4	2.73	1.83	4.56	2.89	2.0	4.89	2.44	2.44	4.88
3	2.64	2.46	5.1	3.0	2.95	6.56	2.43	2.43	4.86
2	3.39	2.57	5.96	3.62	2.95	6.56	2.83	2.83	5.67
1	3.11	2.77	5.87	3.94	3.68	7.62	2.75	2.75	5.49
0	3.8	3.12	6.93	3.8	3.12	6.93	3.33	3.33	6.66
Avg	2.15	1.63	3.78	2.32	1.87	4.19	2.06	2.06	4.12

The average number of pieces in Wild Tic Tac Toe has a very nice symmetry between X's and O's, given that a player can choose which piece to play instead of the default X going first.

The average number of pieces at remoteness 9 is 0 for TTT and Misere TTT, since the empty board is the only one at 9. In contrast, the empty board for Wild TTT is a win, and it has remoteness 7.


Tic-Tac-Toe average piece analysis (symmetries is on)
X
O
Both
Average # of Pieces
Remoteness
0
1
2
3
4
5
6
7
8
9


Misere Tic-Tac-Toe average piece analysis (symmetries is on)
X
O
Both
Average # of Pieces
Remoteness
0
1
2
3
4
5
6
7
8
9



The graphs seem to show a decreasing linear curve, which indicates how the average number of pieces increases as the game is closer to ending. The fact this curve is linear intuitively seems to make sense, as one piece is played per turn.

(c) Future ideas: 1) Place Wild Tic Tac Toe on the website. 2) I would love to see our work combined with analyses of other variants of Tic Tac Toe, and a more sophisticated analysis could be derived from a larger dataset as well. There could also be further analysis done on Remoteness vs Average Number of Pieces based on whether a position is a Win, Loss, or Tie.

My code for generating the tables and graphs shown above can be found here: github.com/J4MIE1P/GamesCrafters/blob/master/TTTsolver.py. I've added comments and designed it so that it could be easily run on any TicTacToe Python file that follows the basic format.