# Stochastic AI in a N,N,N Game

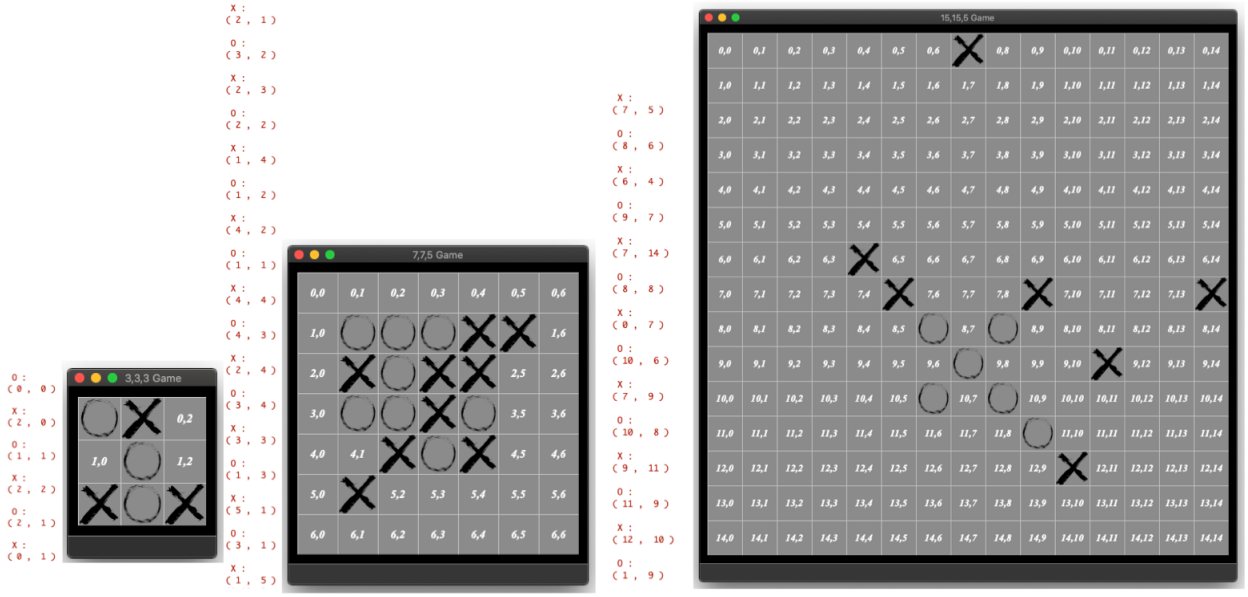
Alonso Vega

August 11, 2020

## 1 What is It?

A $N \times N \times K$ adversarial binary board game is a generalization of the classical $3 \times 3 \times 3$ game known as Tic-Tac-Toe. We implement this abstract game using a variety of data types and mathematical manipulation tools. Computational efficiency and generalization was paramount upon implementation. It is our hope that this software is used by those who have the desire to further its beauty and share it for further improvement.

Below is an example of a $3 \times 3 \times 3$ (Tic-Tac-Toe), an arbitrary $7 \times 7 \times 5$, and a $15 \times 5$ game (Gomoku).




We started out with the idea of modeling a Tic-Tac-Toe game as a connected graph from which we could obtain numerical metrics to feed to a machine learning algorithm (this is one of our current task). So far we have modeled the state of our system effectively and implemented well-known reinforcement algorithms to search for optimal moves ($minimax()$,$\alpha\beta\_search()$,$monte\_carlo\_tree\_search()$). One can easily play against the AI with anyone of these procedures or a mix of them.

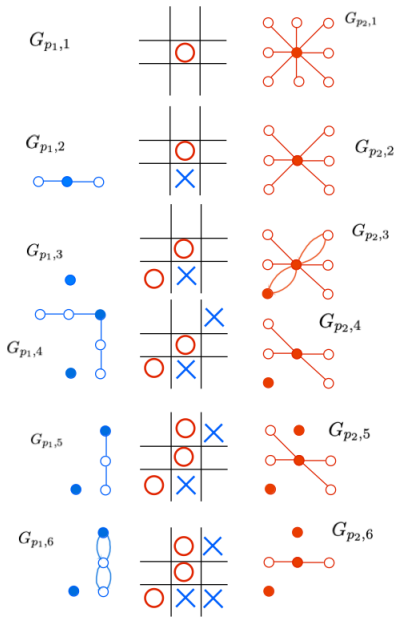
We denote the state of the board as $\boldsymbol{x}_t \in \{-1, 0, 1\}^{N \times N}$, with transitions given by

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + p\mathbf{e}_{i_2}^{i_1} \ , \quad (i_1, i_2) \in [N] \times [N]$$

,where $[N] := \{0, 1, ..., N\}$ and $\mathbf{e}_{i_2}^{i_1}$ is an element of the standard basis for $M_{N\times N}(F)$; $F$ is some field. The integer $p$ represents the player $-1$ or 1. Since this is a simple binary adversarial board game, the two players share a control/action space $U := \{\mathbf{e}_0^0, \mathbf{e}_1^0, ..., \mathbf{e}_{N-1}^N\}$.

The initial state $\boldsymbol{x}_0$ is obviously $\mathbf{0}_N$ with uniformly selected first player $p \sim Unif(\{-1, 1\})$. The utility of each player's win is simply $p$, $ELSE$, 0 (tie).

We are currently working on developing a EVALUATION function for the $\alpha$-$\beta Search()$ subroutine. From a given state of the board $x_t$, this function should robustly be able to estimate the expected utility, $\hat{u} : X \to [-1, 1], \boldsymbol{x}_t \mapsto \hat{u}(\boldsymbol{x}_t)$. An idea is to model the current state of the discrete board as two Cartesian graphs, $(G_{p_1}, G_{p_2})$, and identify the features that may imply an optimal control. Below is an example in a game of Tic-Tac-Toe.




Consider the state $\boldsymbol{x} = [\boldsymbol{x}_1\ \boldsymbol{x}_2\ ...\ \boldsymbol{x}_N] = [\tilde{\boldsymbol{x}}_1\ \tilde{\boldsymbol{x}}_2\ ...\ \tilde{\boldsymbol{x}}_N]^T \in \{-1, 0, 1\}^{N\times N}$ can be decomposed as rows or columns vectors, where the tilde corresponds to row vectors. Thus, if there is only actions of one player in those vectors, then we form a winning path. Same goes for the diagonal and the anti-diagonal $(diag_+(\boldsymbol{x}), diag_-(\boldsymbol{x}))$. The parallel edges between nodes $\{(i_0, j_0), (i_1, j_1), ..., (i_N, j_N)\} \subset V(G_{p_k,t})$ imply a reinforced possible winning path. Each node of each player's graph can be either filled $(filled((i_1, j_2)) = 1_2 \Leftrightarrow \nexists \mathbf{e}_{i_2}^{i_1} \in U_t)$, denoted a past action input, or empty, implying a possible future action$(\exists \mathbf{e}_{i_2}^{i_1} \in U_t)$. These unfilled nodes can be destroyed by the other players moves at a current time.

We also wish to further generalize this $N, N, K$ game into a $M, N, K$ game. That is, to a board of size $M \times N$ where either player must obtain $K$ input actions in physical neighboring sequence to win.

More importantly, our AI begins to statistically fail significantly when the dimension of the board increases from about 7; thus a proper estimate of the utility of the state at a cut-off point in the search is of the essence. Also, we need to formulate a bias search on the control space at

the expansion points. These heuristics are needed, for the control space is severely huge at the beginning of the high dimensional games.

# 2 How to Use It?

The software can be either used with or with out the GUI. If using the GUI, you will need to install QT and have all relative libraries included in $MainWindow.cpp$. If not, you can make your own *.cpp* file and include the $MCTree.h$ and $MiniMax.h$ files.

## 2.1 TTT_state.h

| **Variable Members** | **Type** | **Summary** |
|---|---|---|
| *state* | *array* | Integer matrix that encapsulates $\boldsymbol{x}_t$. |
| *time* | *int* | Current play step. Initially it is 0. |
| *controlSpace* | *set* | Set of tuples, where each tuple $(i, j)$ illustrates allowable actions on the board. |
| *termina* | *bool* | Property of state that signifies terminal state. |
| *player* | *char* | Current player. |
| *outcome* | *string* | Current outcome: Tie, X won, O won. |

| **Methods** | **Returns** | **Summary** |
|---|---|---|
| $in\_controlSpace(i_1, i_2)$ | $b \in \{0, 1\}$ | Checks if $\mathbf{e}_{i_2}^{i_1} \in U$. |
| $print\_controlSpace()$ | $U$ | Prints out $U$ to console. |
| $transition()$ | $-$ | Transitions current state $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$ with uniformly sampled control. |
| $transition(i, j)$ | $-$ | Transitions current state $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$ with control $\boldsymbol{u}_t := \mathbf{e}_i^j$ . |
| $terminal(i_c, j_c, p, K)$ | $-$ | Calculates if current input control $\boldsymbol{u}_t$ leads to a terminal state. |
| $get\_termina()$ | $b \in \{0, 1\}$ | Checks if current state is a terminal state. |
| $get\_player()$ | $c \in \{'X', 'O'\}$ | Gets current player. |
| $print\_board()$ | $-$ | Prints state of board onto console. |