Gimme an X, Gimme an O: A Digital Implementation of Tic Tac Toe

Sanat Khurana and Zeb Mehring

Introduction

Tic Tac Toe is a 2-player game of strategy in which players take turns populating a 9x9 grid with their game pieces in an effort to create a consecutive sequence of three along any vertical, horizontal, or diagonal path on the board. The first player to make such a combination wins the game, or if no such streak occurs when the board is filled, the game results in a tie.

This project implements a digital version of the game where players take turns pressing buttons corresponding to positions on the 9x9 grid. When a valid move is made, an LED corresponding to the chosen position lights up and the players switch turns. Different colored LEDs are used to indicate the different players (Xs are green and Os are blue, using the convention that X always moves first). If a winner emerges, the winning sequence of LEDs blinks and the next button press will reset the game; in the event of a tie, the next button press will perform the reset without blinking anything.

A heuristic-based AI was implemented so that the game could be played by only one player. Different programs are loaded for the two game types.

Game Types

Multiple programs were developed to allow for both 1-player and 2-player scenarios. The 2-player version is as described above: two players take turns making moves until someone is defeated or the game results in a tie. Any further action once either of these conditions is met resets the game. If a player attempts to make an invalid move (i.e. take a space already claimed by themselves or by another player), nothing happens and only the next valid move is recorded. Players may decide amongst themselves who goes first.

In the 1-player version, 3 different levels of difficulty are available. The AI uses a heuristic based analysis to pick the optimal move, and will use this analysis at a rate proportionate to the difficulty level (based on the results of a random number generator). For Level 1, the AI will perform the optimal move 50% of the time, for Level 2 it will do so 75% of the time, and for Level 3, the AI will always make the optimal move. If the optimal move is not made, the AI simply takes the next available space on the game board (traversing row-wise from the upper left corner).

Implementation Strategy

The game was implemented on an Arduino UNO with a program written in C++. The UNO takes digital input from the 9 buttons on the board through pins 2-7, 9-10, and 13, and outputs the game state to the shift registers (which are chained together) through pins 8 and 11-12. The state of the game is stored as a global two-dimensional array with entries that correspond to the color of the LED in each position (or lack thereof). Additional global variables keep track of whose turn it is, whether or not the game is over, and whether or not the game has ended in a tie.

The main loop of the program polls the input pins sequentially to detect a button press. A small delay of half a second is mandated between presses to debounce the signal. When a button is pressed, the game state is updated and the entire board is scanned to detect a win or a tie (the number of possibilities is small enough that all can be checked in negligible time). An invalid move triggers no change. After these updates and checks are done, the results are written to the LEDs via the shift registers, updating the display to the player.

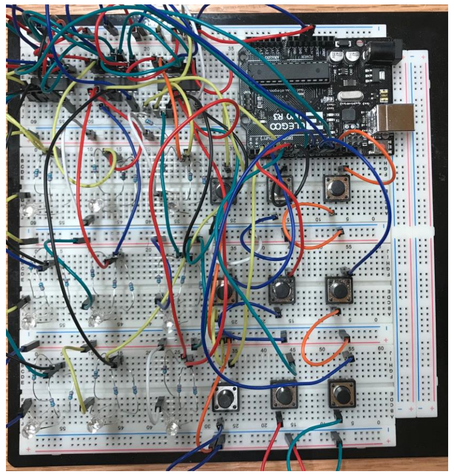
Heuristic-Based AI

The AI uses a set of heuristics to make the best possible move, simulating scenarios (up to 2 moves away) to decide the best position to place its next piece. It iterates through a hierarchy of heuristics using a lengthy if…else statement that breaks when a good move is possible. For each condition, each free space on the board is considered. The conditions tested (in sequential order) are:

1. If a winning move is possible, make it.
2. If the opponent can make a winning move, block it.
3. If a move enables two possible winning sequences (a “fork”), make it.
4. If the opponent has gone first and a move enables a winning sequence and the move will not allow the opponent to make a winning fork, make it.
5. If the opponent can make a winning fork, block it.
6. If the opponent has gone first and it’s the AI’s first turn and the middle space is free, take it.
7. If the opponent has gone first and it’s the AI’s first turn and the middle space is not free, take the upper left corner position.
8. If the AI goes first and it’s the AI’s first turn, take the lower right corner.
9. If all the above conditions fail, take the first available space based on a row-wise traversal of the board from the top left corner.

Bill of Materials

- 1 Arduino UNO Board
- 9 RGB LEDs
- 27 220-Ohm Resistors
- 9 Push Buttons
- 3 74HC595 Shift Registers
- 1 Giant Breadboard
- **Many** Jumper Cables



Circuit Diagram