Battleship!

T.P.S Report

COVER SHEET

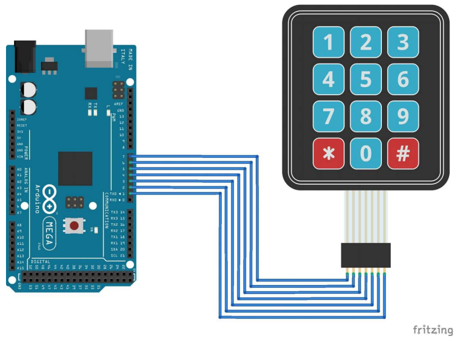
Tad Miller
Danny Nsouli
Systems Programming

Introduction

We chose to program Battleship because we thought it would be a practical challenge to the developing programmers we are. The game begins with a number of pre-set ships of different sizes that both players are able to move, rotate, and place around their individual RGB LED displays. Once both players have set down their ships, the game begins, following the classic mechanics of Battleship. Players move their cursor with a control pad to determine where they fire. When a player has fired on all spots at which a single ship is placed, it is removed from the game, or “destroyed”. The first player to destroy all of the opposition’s ships wins the game.

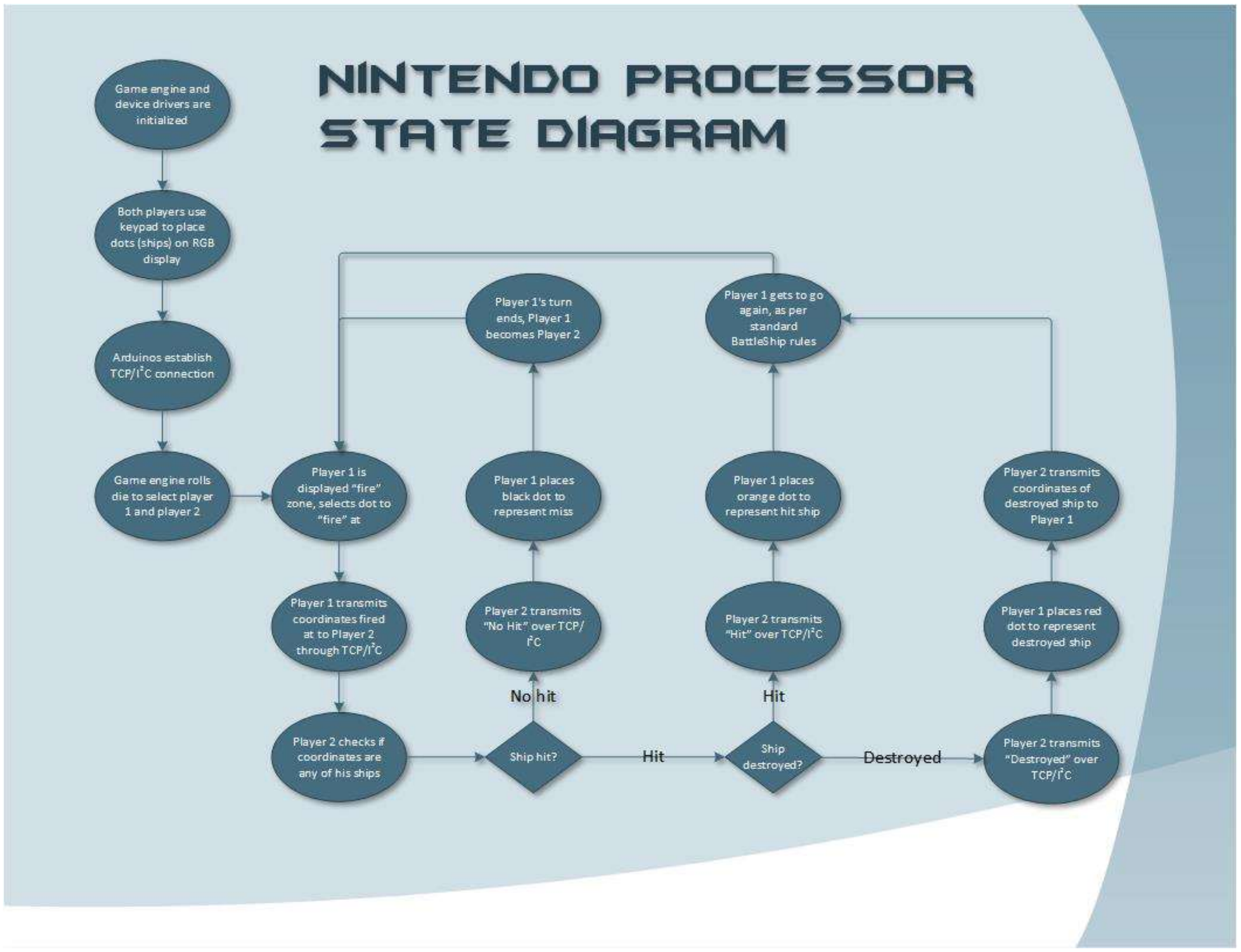
Implementation

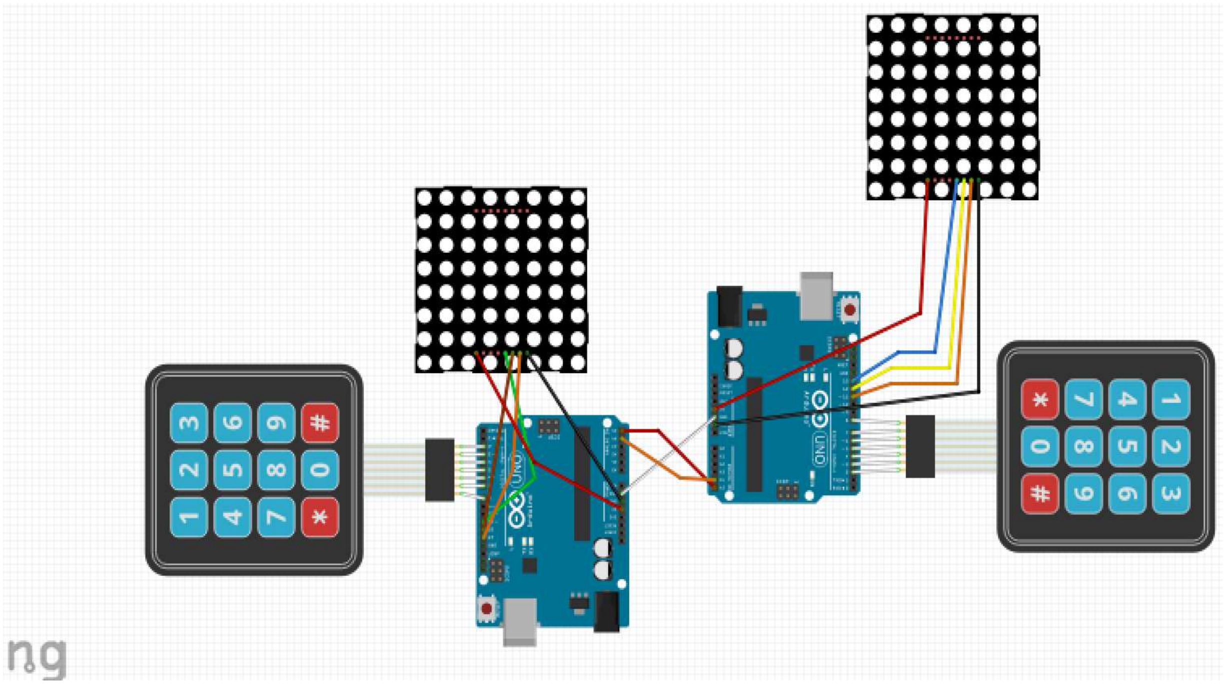
To control a game move, each of the two players has a keypad connected to their individual Arduino. The keypad acts similar to a directional pad on a Nintendo controller.



Controls are as follows: 2 is to move up, 8 is down, 6 is right, and 4 is left. Pressing 7 will allow a currently placing ship to rotate back and forth between vertical and horizontal orientations. Pressing 5 will permanently set down a ship during the setup phase and will later be the “FIRE” button to attack ships during the game. After each player sets the locations of their ships, they are each given a randomly generated number between 1 and 80. The player with the higher number gets to make the first move. A single point will appear on the display which the player will be able to move to choose the space they think the other player has their ships stationed. The screen will be blank and if an opponent’s ship is hit then an orange space will display on the board. If the player misses they will see a black space on the board (shut off light). If a player hits a ship, they are allowed another turn to try to hit again. When a ship is totally destroyed as in all its hit markers are orange, it will turn entirely red. Once all ships are destroyed, the winner’s screen will flash green.

Finite State Machine Blueprint and Fritzing Diagram


NINTENDO PROCESSOR STATE DIAGRAM
Game engine and device drivers are initialized
Both players use keypad to place dots (ships) on RGB display
Arduinos establish TCP/I²C connection
Game engine rolls die to select player 1 and player 2
Player 1 is displayed "fire" zone, selects dot to "fire" at
Player 1 transmits coordinates fired at to Player 2 through TCP/I²C
Player 2 checks if coordinates are any of his ships
Ship hit?
No hit
Player 2 transmits "No Hit" over TCP/I²C
Player 1 places black dot to represent miss
Player 1's turn ends, Player 1 becomes Player 2
Hit
Ship destroyed?
Hit
Player 2 transmits "Hit" over TCP/I²C
Player 1 places orange dot to represent hit ship
Player 1 gets to go again, as per standard BattleShip rules
Destroyed
Player 2 transmits "Destroyed" over TCP/I²C
Player 1 places red dot to represent destroyed ship
Player 2 transmits coordinates of destroyed ship to Player 1



Testing Procedure

Testing first began with trying to correctly implement the directional pad controls with the keypads in order to allow movement of the cursor. After that was completed we tried to create ships. At first we allowed the player to only make one space markers for their ships and then every marker that is adjacent to one another would count as a single boat all together. However, we changed the code in order to give the player a set amount of ships of set sizes in order to more align with the real game. After that we tested setting up the ships and if we could rotate them with a button on the keypad. Once that was completed, we had to figure out the communication code between the arduinos, which took the longest time.This came along with trying to get both arduinos to send the same coordinates to each other in order for them to truly register if the player made a hit or miss on the other player's display. We tested by trying to destroy all ships from one side many times, which always resulted in the game either freezing or not registering what kind of move was made from one arduino to the other. The game also lagged a lot at times so lots of changes needed to be made in order for the coordinates to be transmitted faster. The colors of the display also sometimes did not change or changed to the wrong color depending on what kind of move was made. Also certain rows, especially on the outer border of the display would sometimes be the only spaces that did not cooperate with the arduino's communication by giving row and column numbers in the 100s, so we tested hitting almost every row and column in order to make sure all the

coordinates would translate correctly in the end from one side to the other. In addition, when picking a random number to choose which player would move first, there were a lot of complications with the ASCII values of other commands that would interfere resulting in the same number being chosen for certain players each time not making it random. The range needed to be adjusted to control this issue.

Presentation

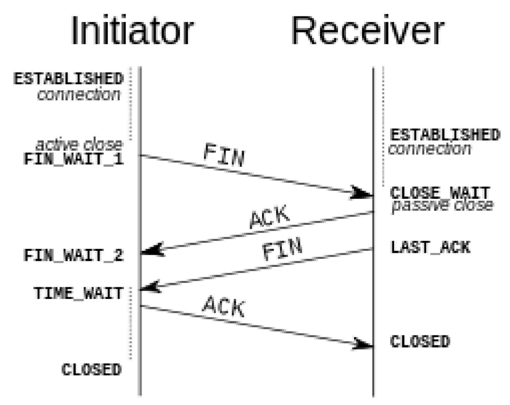
To make the gaming experience fun and user friendly, we constructed an arcade cabinet design out of cardboard that would allow the players to easily understand that the keypad is the controller and, more importantly, for the displays to not to be seen by opposing players in order to more accurately emulate the experience of playing Battleship. We did this by wiring the arduinos, displays, and keypads with one box and using another to mount the displays so that they would be at players' eye levels. Long wires were used to get from the arduinos to the top of the towering box holding the displays. The wires that are plugged into the computers are neatly wrapped around the sides with the use of paper clips to easily access them. The following diagram demonstrates the games' representations of ships through the 8x8 RGB Matrix Display:

RGB Matrix Dot Representation

Color	Game Object
Green	Battleships
Blue	Water/Empty Spaces
White	Hit Cursor
Orange	Hit
Red	Destroyed Battleship
Black	Miss

TCP/I^2C Protocol

One of the largest issues we came across with this project is the ability to reliably transmit data between Arduinos. We developed our own I2C protocol to manage the transmission of ship coordinates in a manner that would be a guaranteed send/receive each time. The best way to describe this protocol would be through the TCP networking protocol, from the initiator/receiver diagram. In our case with Battleship, the initiator sends across a status that they will be transmitting coordinate data. Once the receiver processes data, they transmit back an acknowledgement. The initiator will then send both the x and y coordinates individually through this process, using acknowledgements. This guarantees that the Arduinos will be synchronized throughout the entire game of Battleship.


Initiator
Receiver
ESTABLISHED connection
active close FIN_WAIT_1
FIN
ESTABLISHED connection
CLOSE_WAIT passive close
ACK
FIN_WAIT_2
FIN
LAST_ACK
TIME_WAIT
ACK
CLOSED
CLOSED