

Abstract -

This project is a computer creation on the classic battleship game developed by Milton Bradley in 1967. The game features two players that each has a 10x10 grid to place their 5 ships. The ships have different lengths and therefor will take different number of hits in order to sink. The ships include; the Aircraft carrier that takes up five blocks, the battleship that takes up four blocks, the submarine that takes up three blocks, the cruiser that takes up three blocks, and the destroyer that takes up two spots. Before the game begins the players will secretly arrange their ships on their personal grids. The ships can only be placed either vertically or horizontally. Once the ships have been placed the players take turns trying to guess the positions of the other player's ships. The game has two grids for each player (four total) one is for placing their own ships and keeping track of where their opponent has attacked, the other is for the player to keep track of his own moves and where he has already attacked. The game continues one with each player marking their boards and guessing at the others until all of one player's ships are destroyed. Once all the ships from a player are destroyed the remaining player is declared the winner.

Introduction-

This project was decided upon because it was figured that programing something like a game would be more fun and interesting to do then some type of database or computational analysis. This way the game could be enjoyed as it is being created. The goal of this project was to create a battleship game that could be played on the computer over a network to challenge other players that would be on that network. It was also designed to let just a single player play against the

computer to simulate as the second person. Another part of this project involved introducing a type of controller to play with so the key board would not have to be used. The controller in this case is the circuit with the buttons connected to the ts7250 boards in the Linux lab. A challenge with implementing some type of hardware involves creating a way for the hardware to communicate to the program.

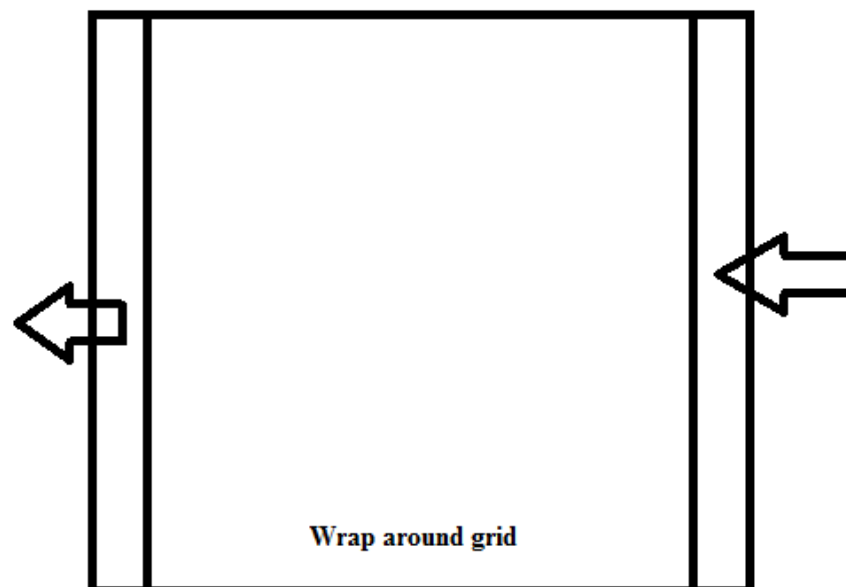
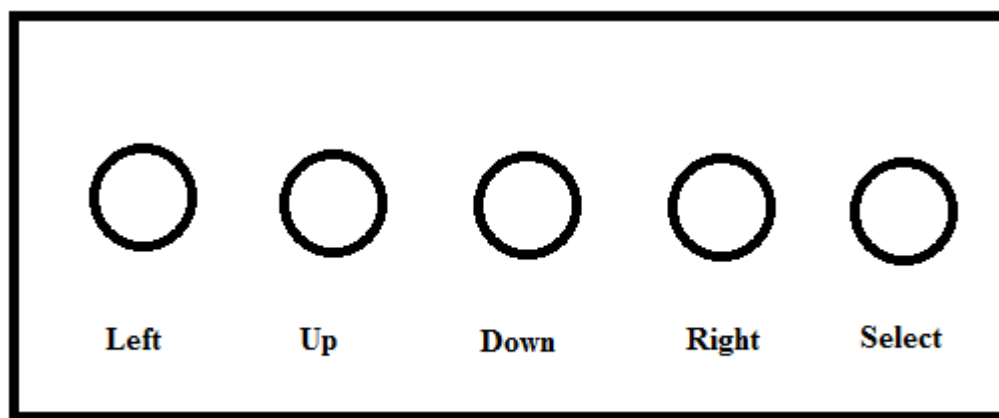
Background-

This project uses the TCP networking protocol used to make a direct connection from one device to another and allows them to send messages back and forth from each other in the form of sockets. If one device losses connection they game will terminate. It also uses a kernel module that must be installed prior to being used to allow the controller to function in the program. This controller uses a fifo to communicate with the user space program and allow the user to attack different positions of the grid and for those attacks to be deciphered and recognized on the other side. Using interrupt handlers allowed for the buttons to be acknowledged as they were pressed in an efficient way. A random number generator was used to make the computer player seem to be attacking the player in various positions.

Implementation-

This program needs in its command lines the port number that the players wish to play on and for the client program before the port number the IP address of the player operating on the server is required to be placed. This allows the connection of the devices to be quickly connected and verified. Once the connection is verified the game will ask each player to place their ships using the

controller connected to the ts7250 board. The buttons from left to right all have a specific purpose. Starting from the left; button one moves the coordinate to the left, button two moves the coordinate up, button three moves the coordinate down, button four moves the coordinate to the right, and button five verifies the move to allow for the next instruction to start. If the player is already on the farthest left column and presses the button to go to the left again the coordinate will swing around and be placed on the right. This works the same in all directions .



However the wrap around grid is only for the convenience of the player the ships that will be placed will not be able to wrap around the top, bottom, left, and right walls will act as boundaries and if one is crossed while placing a ship it will be asked for the ship to be placed again. The fifo created will send the coordinates to the user space program that will then print out the set of coordinates every time a button is pressed so the player knows where the cursor is currently pointing. Once the ships are in place the game will start and the person running the client program will have the first attack. The attack process works just like the process of placing the ships. The player just presses the corresponding buttons and once he is finished presses the fifth button to launch his attack at the other player. These coordinates will be sent through a socket to the other player's terminal in the format of "1.1\0" where the first number is the x coordinate for attack and the second is the y coordinate of attack. The receiving program will determine if the attack was a hit or a miss then send a 12 (if miss) or 11 (if hit) to the program that sent the attack. The game boards then print out for both players so they can see where they have each attacked. The next player will then attack and send his attack through the socket as did the first. This process will repeat until one player has sunk all the opponent's battleships. That player will be declared the winner then the programs will terminate.

If a single player wishes to play the game by his/her self they may select the option at the beginning of the game to play against a computer. The game will play just as before with the player setting their ships and attacking at the opponent. However, this time the opponent is just a computer that randomly placed its ships and will attack you in random positions until someone is declared the winner.

Results-

The program as I tested it seems to work as intended the many functions used inside the program solves the many errors that could come from a program that has a high number of user inputs. The only place that I did not allow for error checks is the main menu when the user must type a 1 or a 2 to either play with another player or play against the computer. Through testing this code many times I found and corrected tiny bugs that showed up that while I was coding were missed. The wrap around grid works as planned to make getting to the correct position easier for the player. The boundaries placed by the top, bottom, left, and right walls as well as other ships also work as intended when the player is placing the ships. The work was divided into four smaller parts that then were placed together to make the final working programs. The first part was to create the simple battleship program with the boundaries and error checking that allowed ships to be placed on a grid and then attacks to be implemented on the grid. The second part involved creating a way to test part one by creating a computer to play against to make sure the game fully functioned and to create a couple of functions that took a, s, d, and w as characters to move the cursor on where the attacks were placed as well as the ships. The third part involved creating the TCP network connection so the two players could play against one another on separate machines. The four part was creating a kernel module that using interrupt handlers would allow for an interrupt to be triggered when a button is pressed but however for it to not mess up the user space programs when it is not needing to be pressed but is pressed anyway. Once these four parts were created they were then put together in order to make the final working project. The game screens will after every move print out the two game boards being used by that player (shown below).

```
computer missed your ship

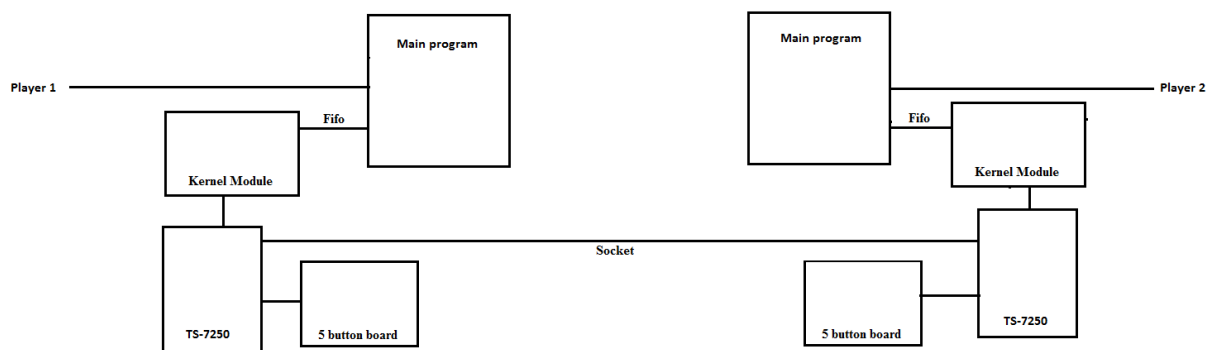
Your game board
  1      2      3      4      5      6      7      8      9      10
1  . . . . . . . . . . . .
2  . . . . s . s . . . . .
3  . . . . x . . . . . . .
4  . . . . . . . . . . . .
5  . . s . x . s . . . . s .
6  . . . . . . . . . . x . . s
7  . . . . . . . . . . s . . s
8  . . . . . s . . . . . . s
9  . . . . . s . . . . . . s
10 . . . . . . . . . . . x . s

Oponents gameboard
  1      2      3      4      5      6      7      8      9      10
1  . . . . x . . . . . . .
2  . . . . . . . . . . . .
3  . . . . . . . . . . . .
4  . . . . . . . . . . . .
5  . . . . x . . . . . . .
6  . . . . . . . . . . . .
7  . . . . s . . . . . . .
8  . . . . . . . . . . . .
9  x . . . . . . . . . . .
10 . . . . . . . . . . . .

Enter attack coordinates

```

The final working game communicates with other devises as shown.



Conclusion-

The battle ship game was a success. After many long mishaps with the socket and the fifos created the game fully functions and works as intended. The socket portion of the code took a while to understand and figure out because we never used the TCP connection in class. It was really similar to the UDP connection however there were a few important differences. Because there was not an easy way to move an integer array through the socket some string manipulation had to be used in order to send coordinates back and forth between programs, it was also true using the fifo to communicate with the five button board. With this in mind basic string manipulation was once again used to transport messages. The only part of the program that I was not able to finish was the use of the sega genesis controller to play the game on instead of the 5 button boards connected to the ts7250 boards. This would have made the game less annoying to play because the sega controller had arrow keys to use instead of four buttons horizontally placed out. I had tried many ways to get the sega controller to operate but could not get the inputs from it in the allotted time to finish the project.