MIDDLE EAST TECHNICAL UNIVERSITY

EE-447

Laboratory Project: Battleship Game Console

Final Report

Contributors,
Abdullah Aslam (2107407)
Mahmoud ALAsmar (2126027)

INTRODUCTION

In this project we are required to make a two player game, in which player one places four ships. The ships are of two types, civilian ship and battleship. After placement of ships, player two has to destroy only the battleships, within a limited time, using mines. Player two can use at max four mines. There are some gameplay restrictions and requirements that are needed to be considered to construct the game. In this report we show how we fulfilled those requirements. Figure 1 shows our setup.

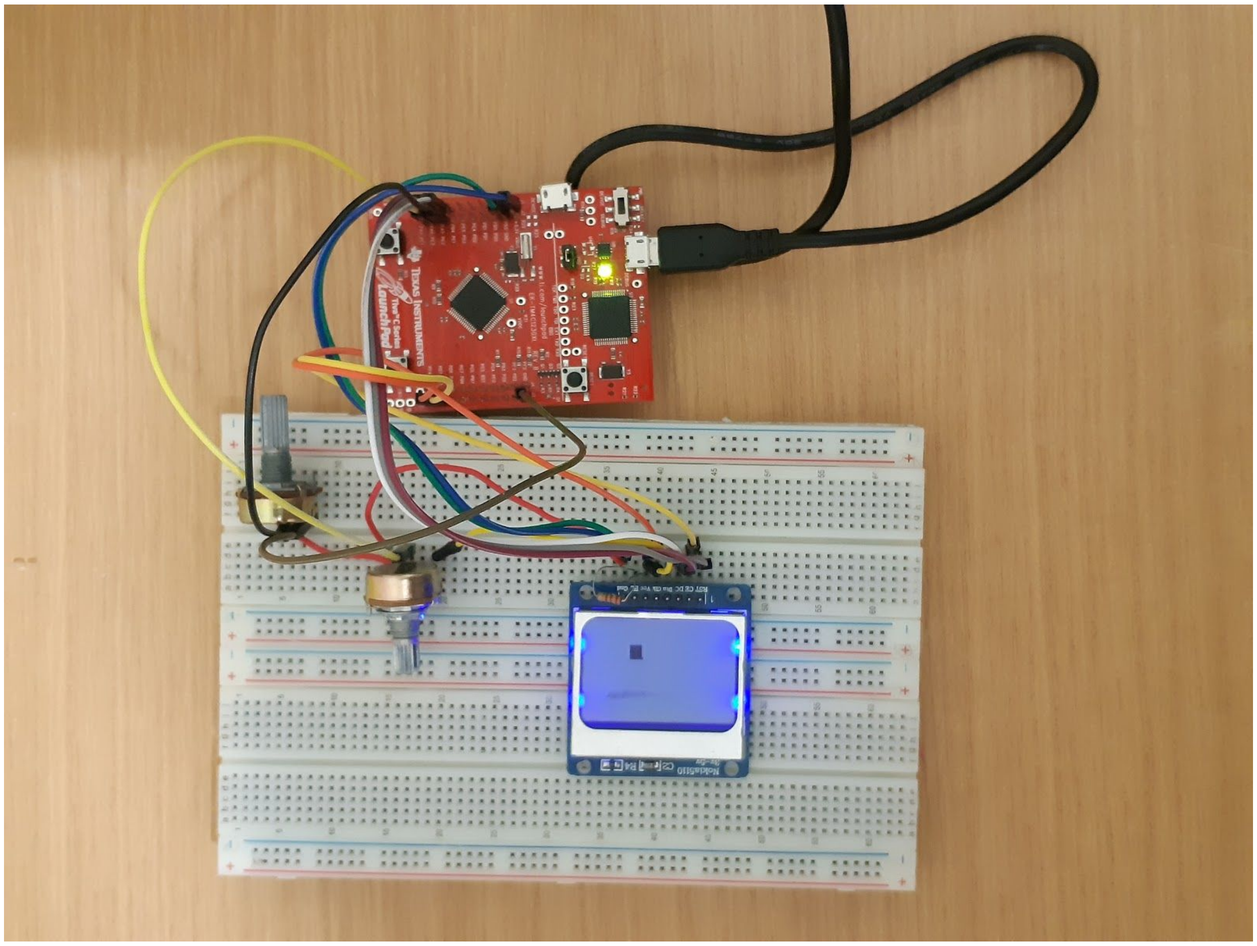


Figure1: The hardware used

Throughout this project the following module will be used:

- ADC : conversion from two different channels , converts inputs of POT1 (vertical movement) and POT2 (horizontal movement). Sampling rate is 125 Ksps , one sample from each channel.
- SPI : Interface with LCD. SCLK = 381KHz, Clock Polarity is High, Clock Phase is second edge, transmitted data are 8 bits.
- General Purpose Timers : used for timing operations, 0.5 second timer and 1 second timer.
- Systick : used for delays.
- Display Module : Controls Display operations. Moves cursor according to given x and y coordinates and draw required icons according to clicked buttons.
- Calibrating Module : generates x and y coordinates of the screen by proper mapping of POT1 and POT2 inputs.
- Decision Module : Compares positions of placed mines with positions of battleships and civilian ships and generates final result of the game.

STAGE 1 : DEPLOYMENT PHASE

REQUIREMENTS

- **Gameplay Area : 64x32 box**
- **Move in all possible directions within play area**
- **Player 1 must place 4 ships**
- **Pointing pixel**
- **Pointing pixel points to top left corner of the bounding box of the ship icon**
- **Ships must be present on screen till the end of the deployment phase**

At this stage player 1 is supposed to place his ships either civilianships or battleships by controlling the cursor and pressing the proper button (SW1 or SW2) for ships deployment.
Play area is restricted to be within 64x32 box. To achieve play area restriction, the code scales the inputs of the 2 potentiometers, which are used by player 1 to control X and Y position of the cursor, to range of numbers. Three informations will be obtained from mapping the input of the potentiometers, first is X position , then Y position and finally a value that indicate which pixel within the Y section. For X axis the range is starting from 6 to 61 so that a ship with 8 pixels in width can be placed at right most position without exceeding the boundary of the game.
For Y axis the input received from the potentiometer is scaled to a number between 1 and 3 so that a ship with 8 pixels in height can be placed at y = 4 without exceeding the boundary of the game. With proper scaling for the input of the potentiometer the cursor can move only within the restricted play area. In our case the cursor size is 3x3 with cross shape and ships size is 8x8 square shape with the square being filled for civilian and unfilled for attack ship. A sample is captured from the potentiometer approximately every 8 usec , the screen will be cleared once a sample is obtained, map the input of the potentiometers, set the cursor position and the draw the cursor. Once a button is pressed either (SW1 or SW2) , the system goes to the handler of buttons. In the handler X , Y , the value within Y section, this value can be one of the following either 1,2,4,8,16,32,64 or 128 and it indicates which pixel within Y section player 1 is aiming to point to, and type of the ship will be stored, there is a separate location in the memory to store these 4 associated informations with each ship. Obtained X , Y and which pixel within Y section information will allow player 1 to move the cursor in all possible points within play area. A counter which is used to count the number of ships being placed will be incremented and the ships being placed will be displayed, if the value of the counter becomes 4 then the systems stops deployment operation and wait for the confirmation of player 1 to proceed to stage 2 of the game, the confirmation is done by pressing SW2 to clear screen and proceed to stage 2. Finally in the deployment phase each time the screen is being refreshed it will be cleaned first (zeros are sent as data to the display to clean the screen), then update the cursor,draw it and draw the already placed ships. Each of cursor drawing and ships drawing are implemented on a different subroutines, each time screen is refreshed to main program branches to these subroutine , in this

way player 1 can keep tracking placed ships and the cursor. In our design the cursor has cross shape, the middle pixel represents the pointing pixel. The obtained informations from potentiometers, i.e X ,Y and value within Y section will be used to indicate the position of middle pixel of the pointer or pointing pixel,also once a ship is placed , the ship is started to be drawn from its top left corner , coordinates of its top left corner is the same as the coordinates of the pointing pixel of the cursor and the rest of the pixels of the either ship are based on these coordinates.

Figure 2 shows the cursor, and figure 3 shows the deployment of ships. Filled square implies civilian ship. The empty square implies battleship.

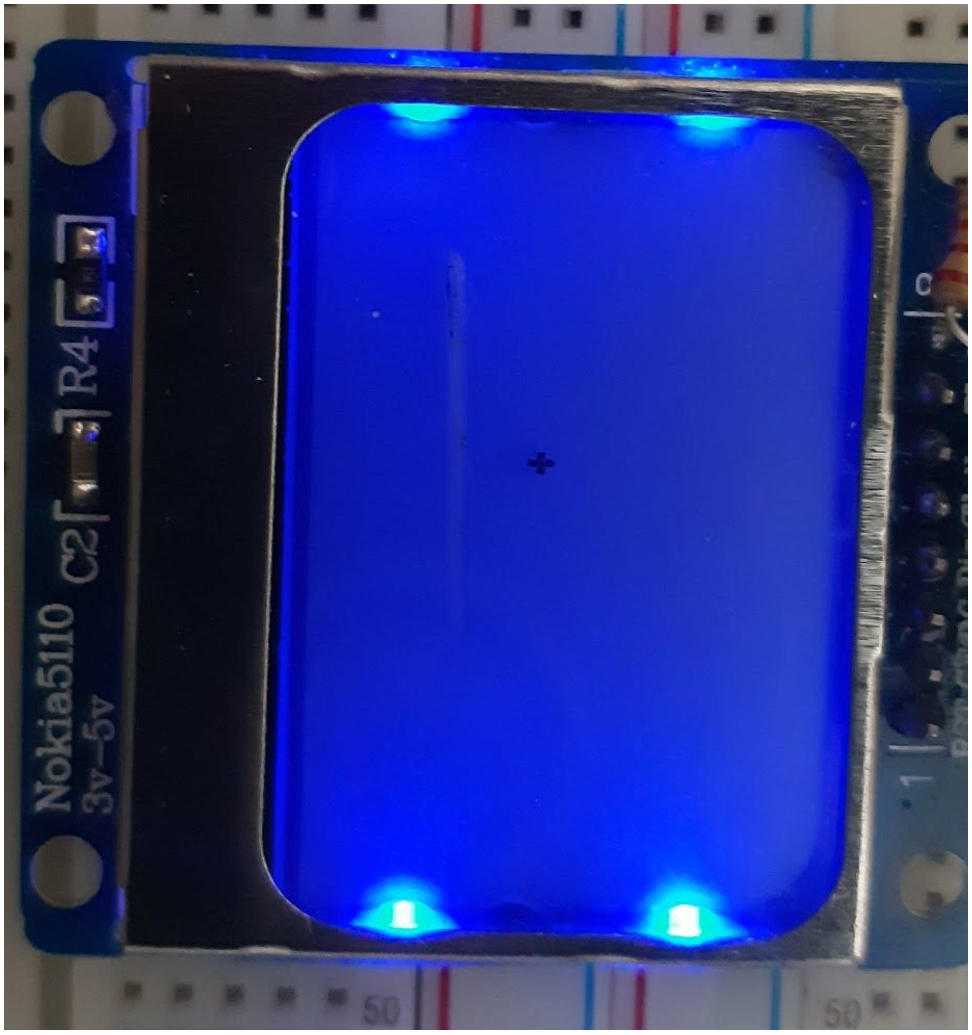


Figure2: Cursor

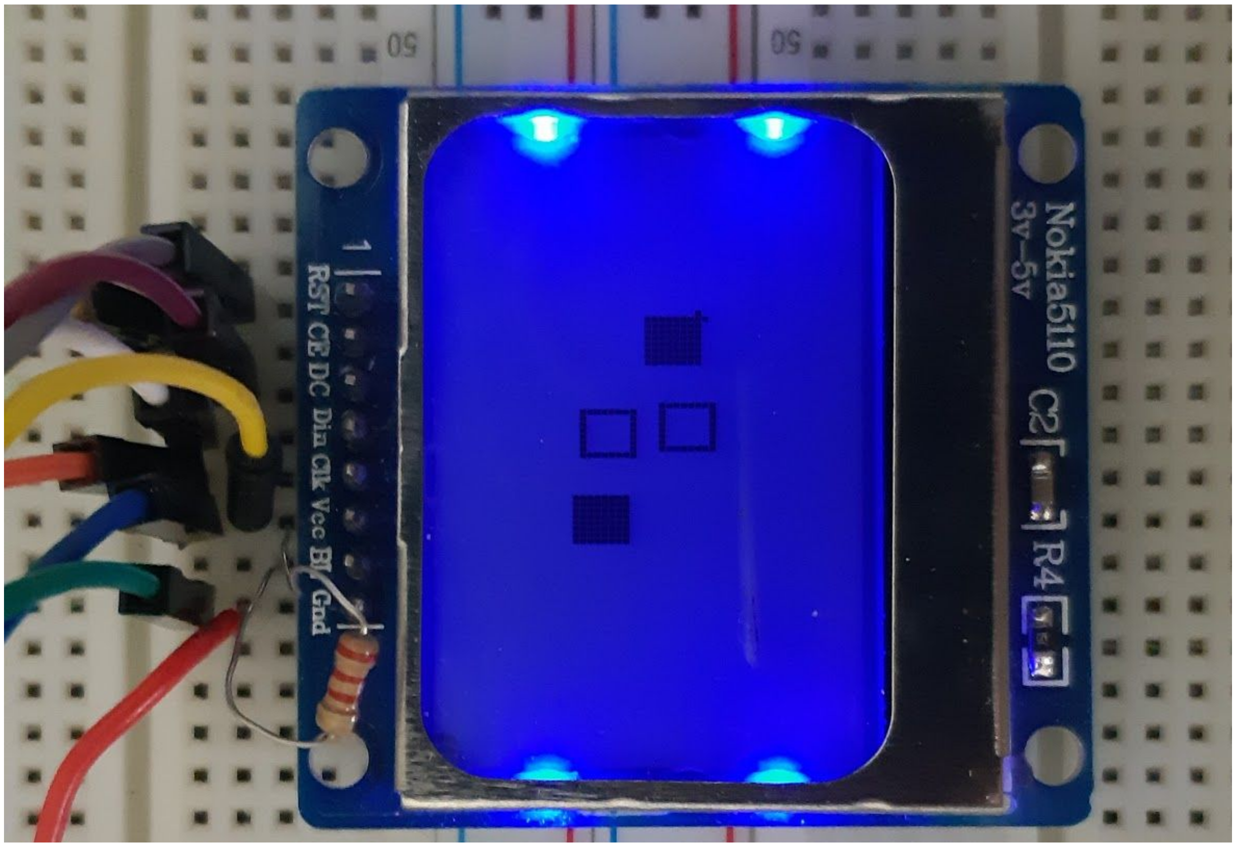


Figure3: deployment of ships

STAGE 2: ATTACK
REQUIREMENT

- **Exact timings (remaining time clock(20 sec), flashing ships(0.5)).**
- **Up to four mines can be placed.**
- **Decrementing Clock at the top right corner.**
- **Wait for 20s to pass to make decision.**

Stage 2 starts with a blank screen, and waits for the interrupt from button 2 (SW2) in order to display placed ships at stage 1 for 0.5 seconds. A general purpose timer is used to count down from a specific value at frequency of 50 kHz to count 0.5 second. Once the time out flag of the timer is set, the screen will be cleared. SysTick will be initialized to count 1 second and register R7 will hold the the value 20 and it will be decremented each time the handler of SysTick is activated. Then from the handler of SysTick the program branches to subroutine that prints the number being stored in R7. To print the number, coordinates X = 69 and Y = 0 will be sent to LCD screen to set the position of "countdown(CD) " and then two digits will be displayed each digit is 7x8 , these digits are same as the digits contained in R7, in this way player 2 can see the time that is remaining. Player 2 will be able to move the cursor within an area of 64x32 to place his mines, each mine is a pixel. Player 2 can place a mine by pressing SW1, each time SW1 is pressed the program goes to the handler of buttons , stores the location of the mine , same as principle of storing the ships using the three information : X , Y and pixel within Y section. There is a separate location in the memory to store these three information that are associated with each mine. After storing the mine it will be displayed on the screen so that player 2 can keep tracking of the placed mines. Player 2 can place up to 4 mines so counter is used to count the number of mines being set. This counter will be incremented at the buttons handler and if it is equal to four then player 2 will not be able to set any other mines but he can place less than four. Once R7 is equal to zero the program branches directly to decision module indicating that stage 2 has been done.

STAGE 3: DECISION MAKING
REQUIREMENT

- **Check for match.**
- **Make Decision.**
- **Display Message.**
- **Start Over.**

In this stage, the decision of player two winning the game or losing the game is made. In order to achieve this we compare the saved position data of all four ships with saved position data (X,Y section, Y position) of the mines. Decision making sub routine checks whether the match is made or not using vicinity check subroutine. In vicinity subroutine a virtual bounding box is created around the ship and the mine location is checked if it is found on or within the bounding box or not. For the bounding box, it first compares the x coordinate of the mine (Xm) with x coordinate of ship(Xs). The first condition to be satisfied is Xm<=Xs and then Xm>Xs-1. Any of these conditions fail a negative flag is set, that implies not in vicinity.If previous conditions are satisfied, y section of mine (Yms) and y section of ship (Yss) is compared, first Yms with Yss and then Yss+1, because a ship can be in two sections Yss and Yss+1. Again, if mine is not found in either of these two sections vicinity subroutine gives a negative flag. After this, the value within the y section (Yposition) is compared, which implies the position of a pixel in a section considered vertically (0, being LSB and top, 7 being MSB and bottomy). Note that, y position implies that where is the pixel located within a section, in the case of ship it means the starting position of ship in the section. If mine is in Yss, then to be in the vicinity its y position (Ymp) has to satisfy this, Ymp>Ysp (ship's y position). This implies mine is located lower than the ships top. If mine is in Yss+1, then to be in the vicinity its y position (Ymp) has to satisfy this, Ymp<Ysp. This implies mine is located above the bottom of ship. So, if Y position condition is satisfied also, then the mine is in vicinity of the ship. Vicinity subroutine sets a flag accordingly (match is made or not) and goes back to decision subroutine. The flow of decision program goes this way, it considers one ship first and checks for all the mines in its vicinity and then same for the next three ships. In this procedure we consider the following option for losing,

- P2 loses if a civilian ship is hit, a mine is found in vicinity of civilian ship by vicinity subroutine.
- P2 loses if there is a battleship but not hit (not all the battleships are hit), no mine is found in the vicinity of that ship by vicinity subroutine.
- P2 loses if there is no mine placed within 20 seconds, definite loose.

If the above conditions are not met, then player 2 wins. However, it is based on the assumption that at least a battleship is placed by player 1. If player 2 wins, this message is displayed

‘WOW!! YOU WON’. If player 2 loses, this message is displayed ‘YOU LOSE LOSER !!’, using display subroutine. The message is displayed for two seconds and then the program is looped back to the deployment phase.

Figure 4 and figure 5 show the win and loose message respectively.

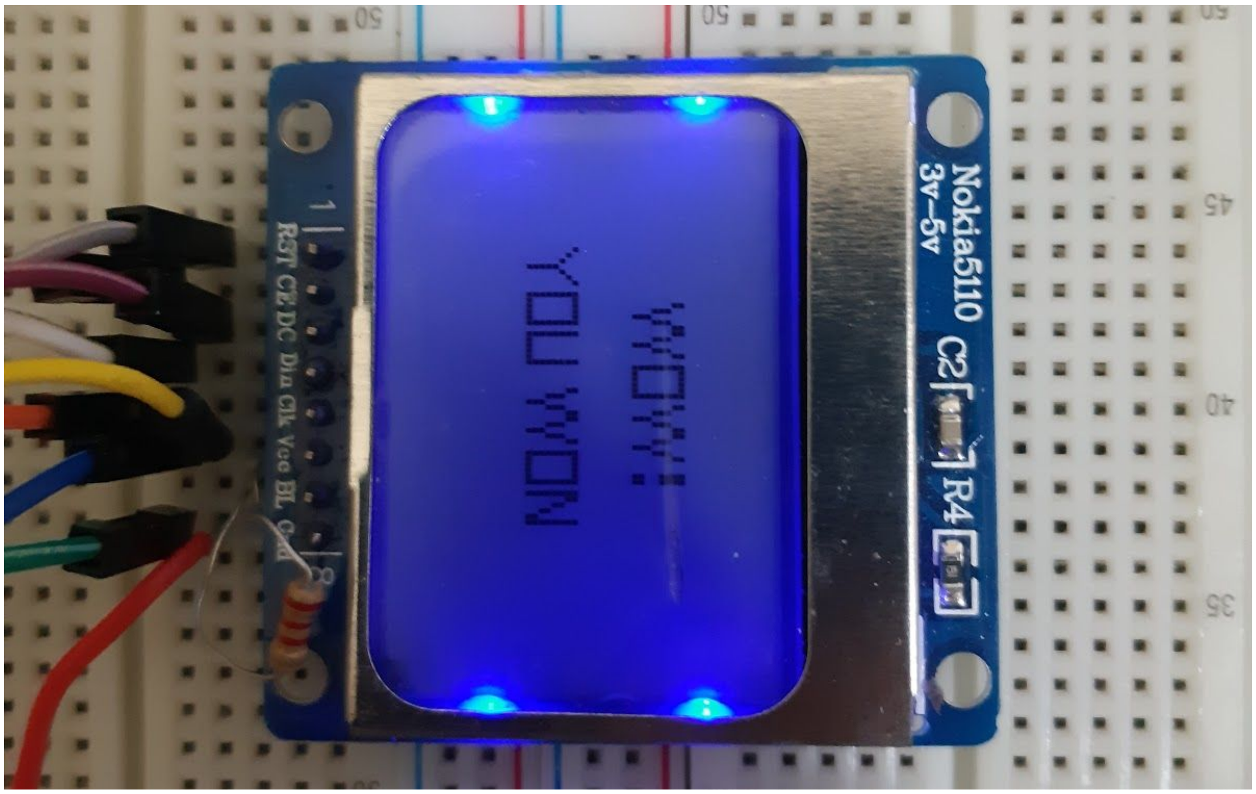


Figure4: win message

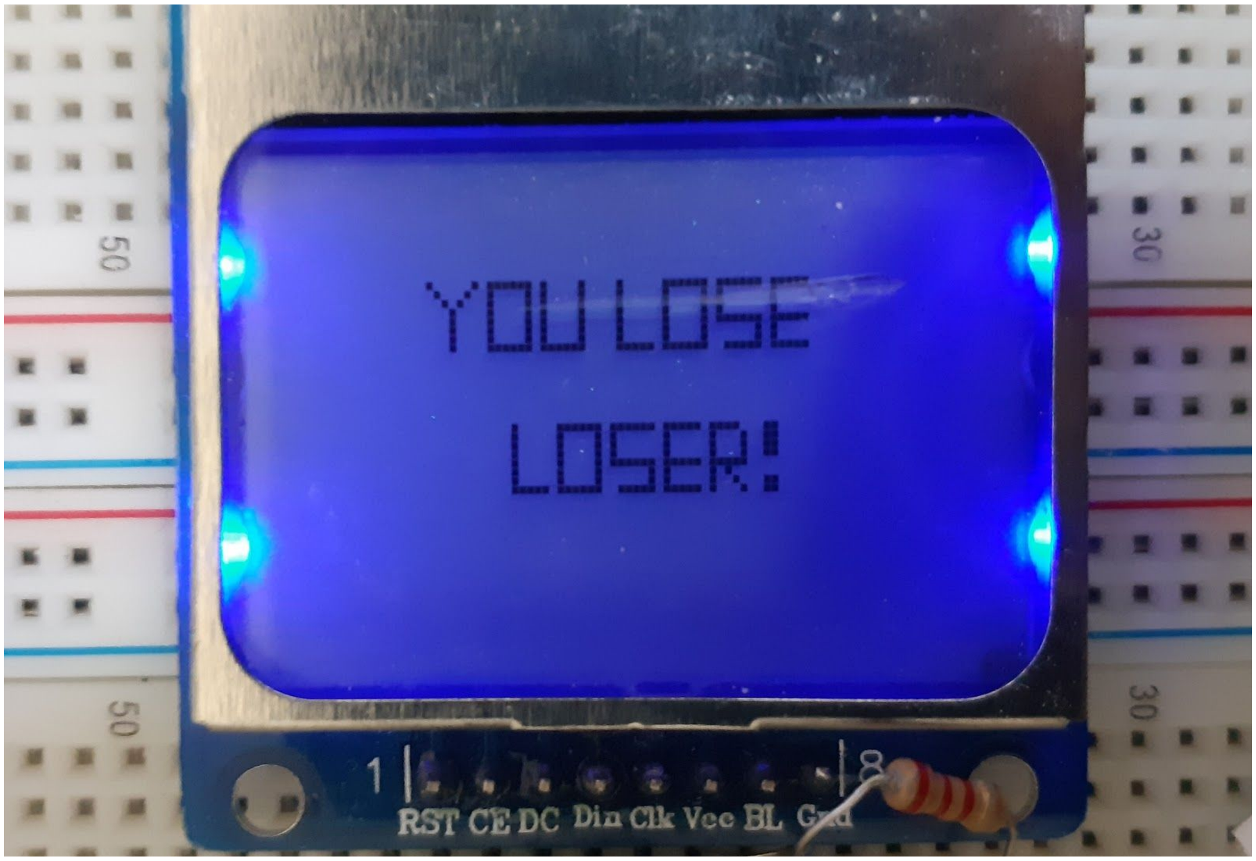


Figure5: defeat message

BONUS

P1 places at least one battleship

In order to achieve this, we force the player 1 to put the fourth ship as battleship if the first three ships are civilian ships. Every time a civilian ship is placed with pressing button 2, interrupt takes us to the handler for the button and the civilian ship location is saved and the ship is displayed. Introducing a counter in handler counts number of times it went to handler. If the counter is greater than three, placing ship is skipped and it simply gets out of the handler. So, the fourth shipl player one places is the battleship.

APPENDIX

ICONS

We will use the max limit of icons,

ICON	**SHAPE**	**SIZE**
Cursor	plus sign, pointing pixel at centre.	5x5 pixels
Battleship	unfilled square	8x8 pixels
Civilian Ship	filled square	8x8 pixels

FLOWCHART

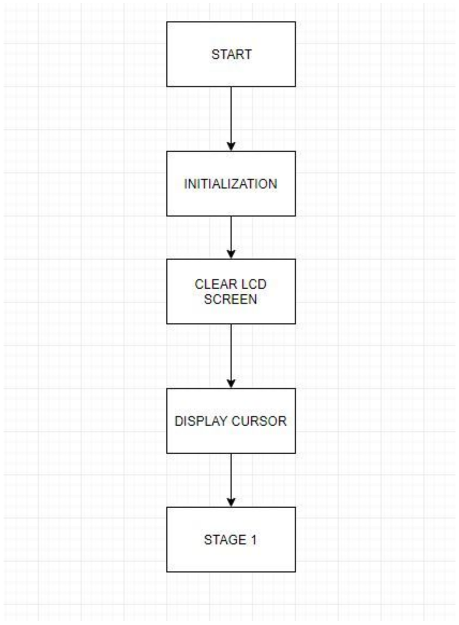


Figure 1 :Game Start flowchart

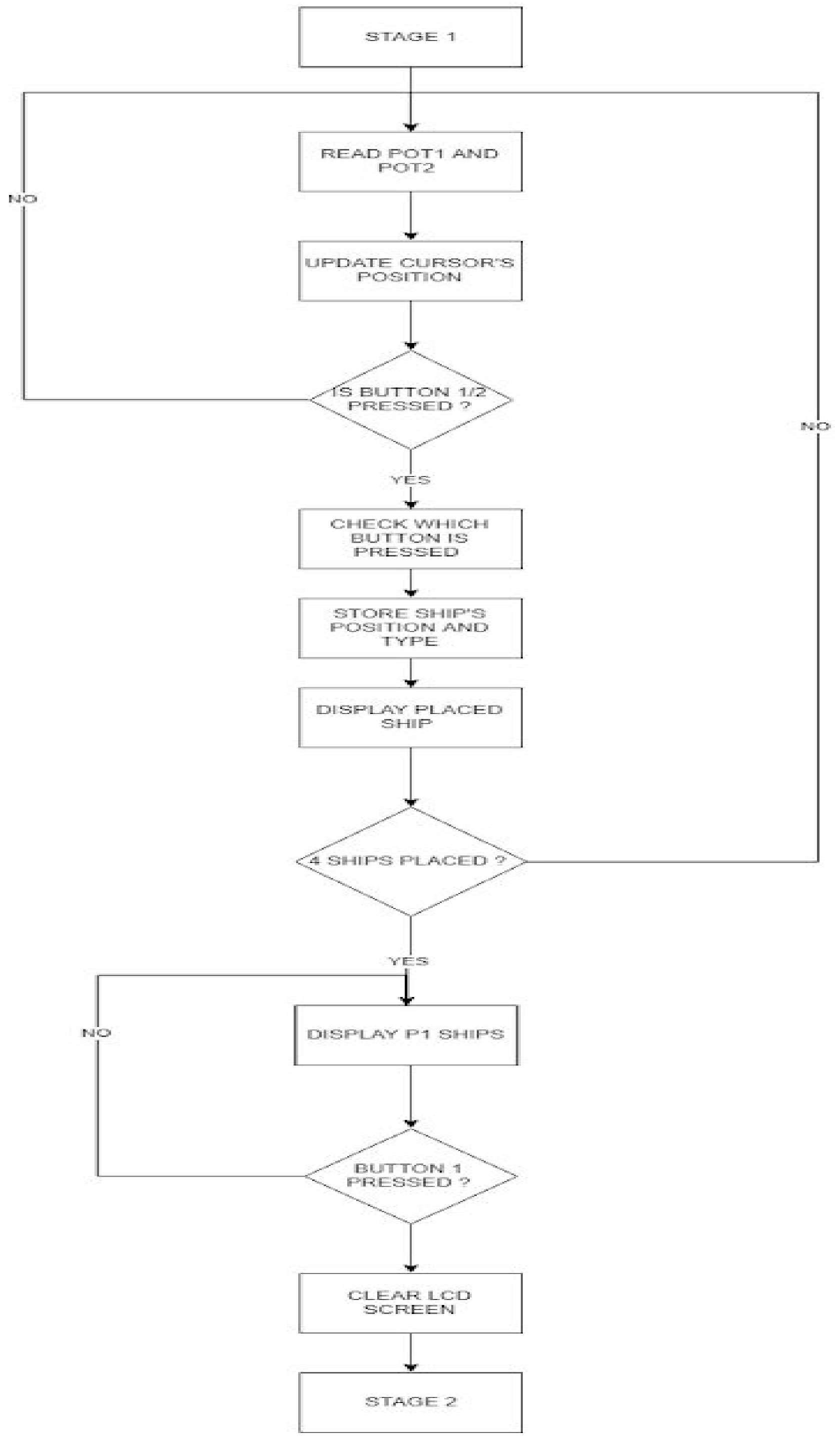


Figure 2: Stage 1 flow chart

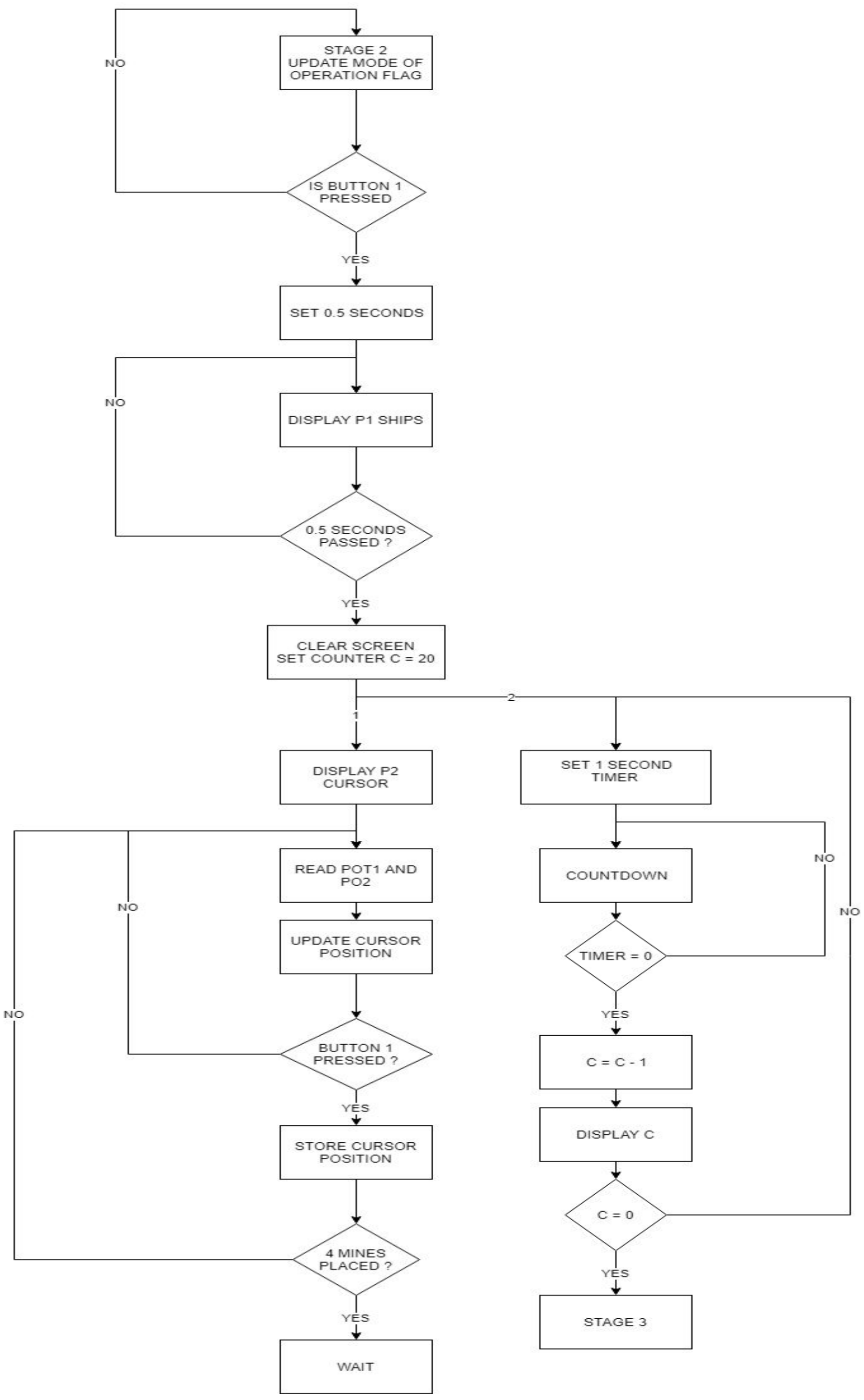
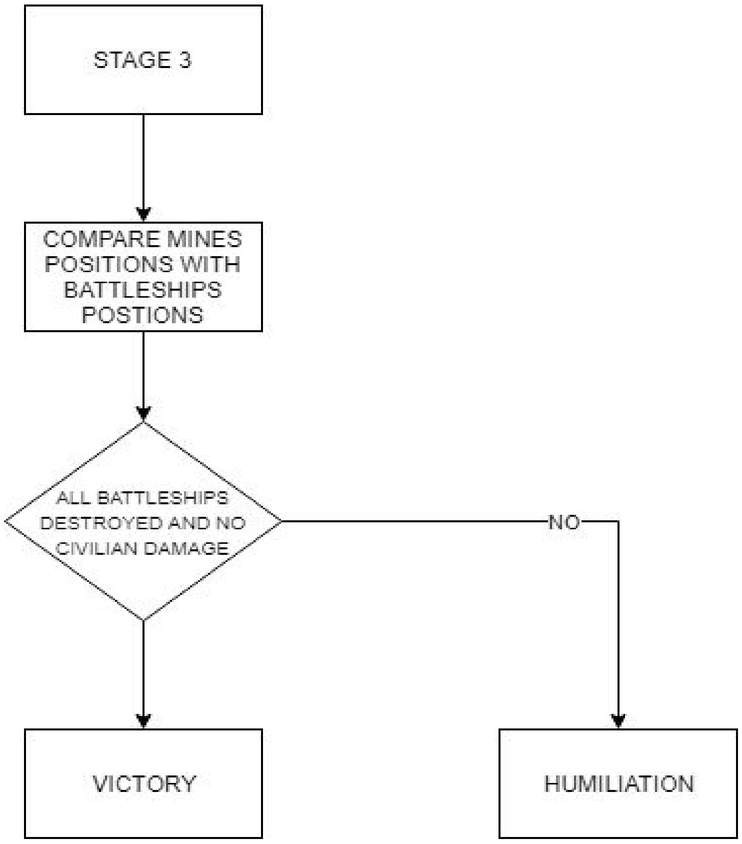


Figure 3: Stage 2 flow chart


STAGE 3
COMPARE MINES POSITIONS WITH BATTLESHIPS POSTIONS
ALL BATTLESHIPS DESTROYED AND NO CIVILIAN DAMAGE
NO
VICTORY
HUMILIATION
Figure 4: Stage 3 flow chart