Project 2

<Battleship Game>

CSC-5 40651
Name: John Olmos
Date: 2/7/18

Introduction

Battleship is a two-player guessing game. The game is played on two square grids on which players place a number of ships. Players take turns picking elements within the grid to try and "hit" the opposing player's ships until all elements have been hit and their fleet is destroyed.

Game Rules

A typical game starts out with two 10x10 grids per player using numbers and letters to help identify elements within the grid. One grid is used to arrange the player's ships and record their opponents picks while the second grid is used to record the player's own picks. Before the game starts each player arranges ships of varying size in secret within the grid making sure not to overlap with any other ship.

After each player has placed their ships the game starts in a series of rounds. During a round each player picks and element within the opponent's grid in an attempt to hit one of their ships. Their opponent then announces if the pick was a "hit" or a "miss" on one of their ships and the result is recorded on a grid. If all of the elements of a ship have been hit it is considered sunk. The game ends once all ships have been hit and sunk.

Development Summary

V1

For simplicities sake the board was reduced to 5x5 grids with 3 ships. The first obstacle was creating a way to allow the user to place a ship and then validating the ship placement within the board. Once this was done the process was mirrored and adjusted to allow the program to act as an opponent. Most time was spent on correctly updating the board and validating inputs.

V2

With the program essentially functional, v2 was focused entirely on incorporating the missing elements of the checklist into the program. This resulted in some minor changes to existing functions and the addition of a few new ones.

V3

The last version of the program focused on including the remaining items on the checklist resulting in a few additional functions. Notably this resulted in the creating and addition of a system for logging and updating game records.

V4

The final version revolved entirely on improving comments for pseudocode.

Example Inputs/Outputs

```
Set your Battleship (4 spaces long)
Select a starting space (ex B2): B2
Place vertically? (y/n): y

  Defensive Board
     1  2  3  4  5
    --------------
A |
B |    S
C |    S
D |    S
E |    S
```

Placement

The first prompt in the game asks users to place and their ship within the grid by entering a letter and a number. Once the position is submitted a prompt asks for confirmation on an orientation before placing the ship on the board.

In the left image the program has output the player's board after the user entered B2 for placement and yes for a vertical orientation. The letter "S" is used to indicate the ship within the board.

Turns

Once all ships have been set the game asks users to enter a position within the grid to try and hit the opponent's ship. Once entered, the program outputs both the player and opponent choices along with the defensive and offensive boards.

If a pick manages to hit a ship, then the board outputs an "X" the selection is marked with an "O" on the board. The image on the right shows the status of each board after two turns. Here we see that the computer managed to hit the user during both turns while the user had one hit and one miss.

```
 Defensive Board
    1  2  3  4  5
   --------------
A | S  S  S
B |    X
C |    S
D |    S     S
E |    S     X

 Offensive Board
    1  2  3  4  5
   --------------
A | ?  ?  ?  ?  ?
B | ?  ?  X  ?  ?
C | ?  ?  ?  ?  ?
D | O  ?  ?  ?  ?
E | ?  ?  ?  ?  ?
```

Results

Once all ships have been sunk on either board a winner is declared and a set of statistics is generated as shown below.

```
You Won!

Player Statistics
-----------------
Rounds   = 12
Misses   = 3
Hits     = 9
Hit rate = 75.00%
Total play time in integer seconds = 1287

RUN SUCCESSFUL (total time: 30m 39s)
```

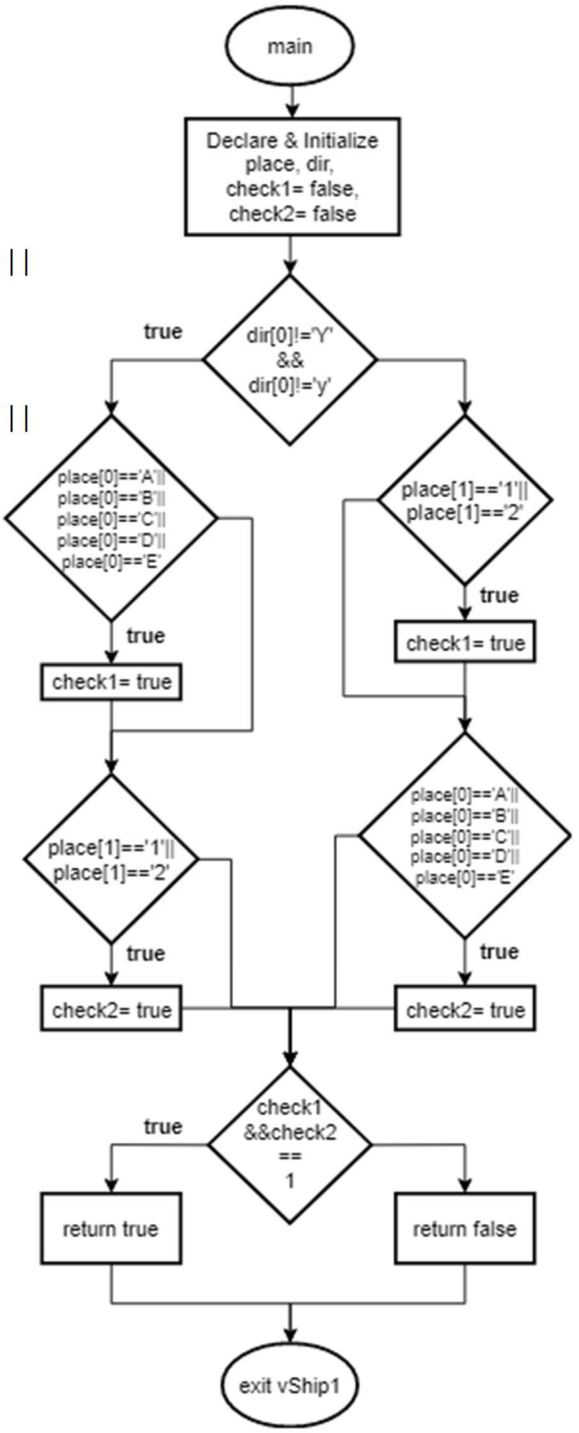
Flowchart Snippets

The following is a set of flowcharts to illustrate the logic flow at varying points within the code.

Battleship Validation

▪ Function ▪ If (independent) ▪ If-else (dependent) ▪ Switch

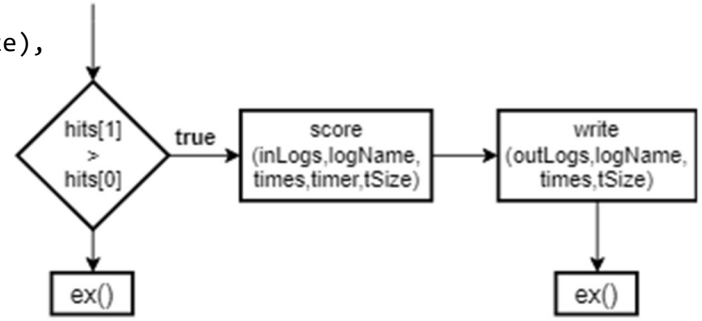
```
bool vShip1(string place, string dir){
    //Declare & Initialize checks
    bool check1=false;
    bool check2=false;

    //Validate ship placement on the board
    if(dir[0]!='Y'&&dir[0]!='y'){
        if(place[0]=='A'||place[0]=='B'||place[0]=='C'||
           place[0]=='D'||place[0]=='E')check1=true;
        if(place[1]=='1'||place[1]=='2')check2=true;
    }else{
        if(place[0]=='A'||place[0]=='B')check1=true;
        if(place[1]=='1'||place[1]=='2'||place[1]=='3'||
           place[1]=='4'||place[1]=='5')check2=true;
    }

    //Return True if ship fits, else return False
    switch(check1&&check2){
        case 1:return true;break;
        default:cout<<"\nError, ship does not fit\n";
                return false;break;
    }
}
```



Update scores

- Ternary operator

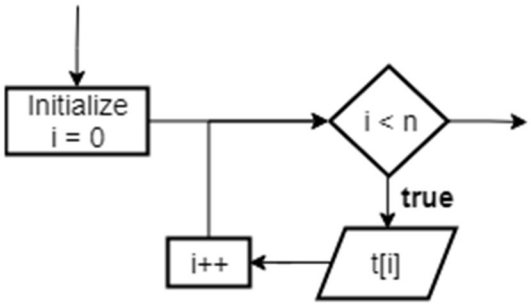
```
//Check if user won
(hits[1]>hits[0])?
//If true
    //Open record file
    score(inLogs,logName,times,timer,tSize),
    //Update record file
    write(outLogs,logName,times,tSize),
    //Exit Function
    ex():
//Else
    //Exit Function
    ex();
```



Write scores to file

- For loop

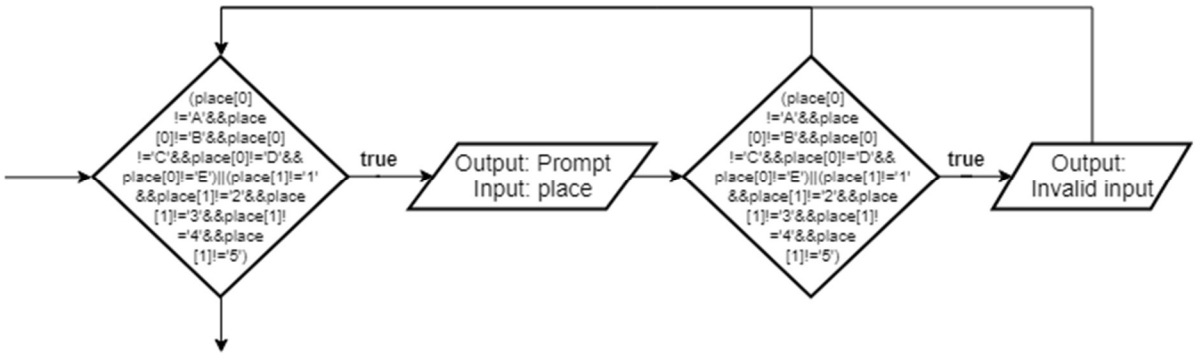
```
//Output new score array to file
for(int i=0;i<n;i++){
    output<<t[i]<<endl;
}
```



Input validation

▪ While Loop

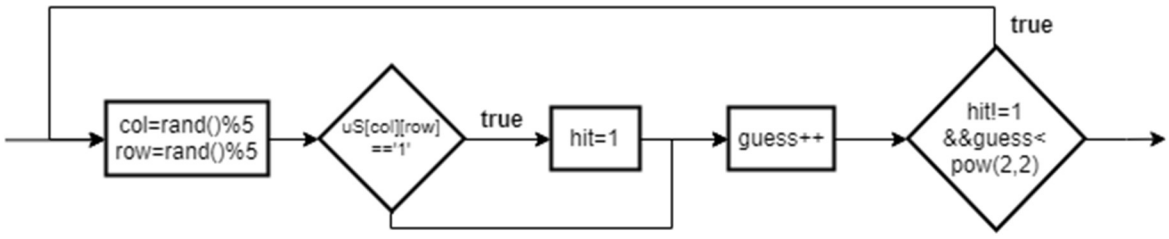
```
//User Pick, loop until input is valid
while((place[0]!='A'&&place[0]!='B'&&place[0]!='C'&&place[0]!='D'&&
       place[0]!='E')||(place[1]!='1'&&place[1]!='2'&&place[1]!='3'&&
       place[1]!='4'&&place[1]!='5')){
    cout<<"\nSelect a space to target (ex B2): ";
    cin>>place;
    //Display error if invalid
    if((place[0]!='A'&&place[0]!='B'&&place[0]!='C'&&place[0]!='D'&&
       place[0]!='E')||(place[1]!='1'&&place[1]!='2'&&place[1]!='3'&&
       place[1]!='4'&&place[1]!='5')){
        cout<<"\nInvalid entry\n";
    }
}
```



Computer pick

▪ Do loop

```
do{
    col=rand()%5;//Pick row [0,4]
    row=rand()%5;//Pick column [0,4]
    if(uS[col][row]=='1')hit=1;
    guess++;
}while(hit!=1&&guess<pow(2,2));
```



Pseudo-Code

```
/* Opening Comments */

//System Libraries
//User Libraries
//Global Constants
//Function Prototypes

//Begin Main
    //Set the random number seed

    //Declare/Initialize Variables

    //Initialize strings & counters

    //Clear & Set Boards
    //Defensive Board
    //Offensive Board

    //Display Intro

    //Place computer ships

    //Display Defensive Board

    //Place user ships

    //Display Game Start

    //Start Game Timer

    //Begin and loop rounds until the
    //user or computer hit all targets
        //User turn
        //Increment hits if ship was hit
        //Computer turn
        //Increment hits if ship was hit
        //Output round choices
        //Display Defensive board
        //Display Offensive board
        //Increment rounds

    //Stop Game Timer

    //Display Game statistics

    //Check if user won
    //If true
        //Open record file
        //Update record file
        //Exit Function
    //Else
        //Exit Function
```

```
//Begin Display intro
    //Output text

//Begin Set Defensive Board
    //Clear board markers
    //Set ship locations to 0

//Begin Set Offensive Board
    //Set board markers to "?"
    //Set ship locations to 0

//Begin Computer placement
    //Open the file
    //Read ship placement data into array
    //Close the file

//Begin User placement
    //Declare & Initialize variables

    //Battleship Location
    //Loop until location is valid
      //Loop until inputs are valid
          //Prompt user for input
          //Validate Input
        //Validate Placement
    //Place Battleship on Defensive Board
    //Display Defensive Board

    //Submarine Location
    //Reset validation
    //Loop until location is valid
      //Loop until inputs are valid
          //Prompt user for input
          //Validate Input
        //Validate Placement
    //Place Submarine on Defensive Board
    //Display Defensive Board

    //Destroyer Location
    //Reset validation
    //Loop until location is valid
      //Loop until inputs are valid
          //Prompt user for input
          //Validate Input
        //Validate Placement
    //Place Destroyer on Defensive Board
    //Display Defensive Board

//Begin Validate Battleship location
    //Declare & Initialize checks
    //Validate ship placement on the board
    //Check Vertical placements
    //Check Horizontal placements
    //Return True if ship fits, else return False
```

```
//Begin Validate Submarine location
    //Declare & Initialize checks
    //Validate ship placement on the board
    //Check Vertical placements
    //Check Horizontal placements
    //Return True if ship fits, else return False

//Begin Validate Submarine location
    //Declare & Initialize checks
    //Validate ship placement on the board
    //Check Vertical placements
    //Check Horizontal placements
    //Return True if ship fits, else return False

//Begin Validate ship placement
    //Declare & Initialize variables
    //Clear test board
    //Place ship on test board
    //Search for matching elements between the test and active boards.
    //If the search finds an equal element then ships would overlap.
      //Check Vertical placements
          //Display error/return false if occupied
      //Check Horizontal placements
          //Display error/return false if occupied

//Begin Set Ship
    //Declare & Initialize
    //Change Board to place ship
      //Vertical placements
        //Horizontal placements

//Begin Display Start
    //Output text

//Begin User Turn
    //Declare & Initialize
    //User Pick, loop until input is valid
        //Display error if invalid
    //Output choice
    //Convert string to char
    //Update offensive board
    //If choice hit
        //Change element to "X"
        //Return true
    //If choice missed
        //Change element to "O"
        //Return false

//Begin Computer Turn
    //Declare & Initialize

    //Multiple choices are generated to improve enemy odds.
    //Choice loops until it hits one of the user's ships or once
    //four attempts have been made (Note: If the final choice
```

```
    //matches that of a previous round the counter is reset).

    //Only display the final choice
    //Update defensive board
    //If choice hit
        //Change element to "X"
        //Return true
    //If choice missed
        //Change element to "O"
        //Return false

//Begin Display Board
    //Display Header
    //Display Grid

//Begin Display Game Results
    //Display win/loss/tie and statistics

//Begin Score
    //Open the file
    //Read and copy the data from the file into array
    //Selection sort array
    //Add game score to end of array
    //Bubble sort new array
    //Close the file

//Begin Write
    //Open the file for writing
    //Output new score array to file
    //Close the file

//Begin Selection sort
    //Declare variables
    //Selection sort

//Begin Bubble sort
    //Declare Variables
    //Bubble Sort

//Begin calc
    //Calculate %

//Begin calc
    //Subtract

//Begin exit function
    //Exit function with defaulted argument
```

Check-off Sheet

Introduction to C++

Chap	Section	Topic	Line Location	Pts
2	2	cout	Throughout	
	3	libraries	9-16	5
	4	variables/literals	Throughout	
	5	Identifiers	Throughout	
	6	Integers	23, 52, 58, 59, 64-67, 72, 145, 146, 155, 156, 168, 169, 180-182, 364, 365, 374, 383, 402, 407, 460-462, 497, 506, 533, 555, 565, 571, 585, 590	1
	7	Characters	60-63, 358, 359, 396, 445, 446, 480, 481, 504, 505	1
	8	Strings	70, 71, 183, 425	1
	9	Floats No Doubles	524, 601	1
	10	Bools	184, 282, 283, 306, 307, 331, 332, 584	1
	11	Sizeof *****		
	12	Variables 7 characters or less	Throughout	
	13	Scope ***** No Global Variables		
	14	Arithmetic operators	Throughout	
	15	Comments 20%+	Throughout	2
	16	Named Constants	23, 58, 59, 180-182	
	17	Programming Style ***** Emulate		
****** **Not required to show**			Total	12

Expressions and Interactivity

Chap	Section	Topic	Line Location	Pts
3	1	cin	Throughout	
	2	Math Expression	Throughout	
	3	Mixing data types ****		
	4	Overflow/Underflow ****		
	5	Type Casting	52, 480, 481, 505	1
	6	Multiple assignment *****		
	7	Formatting output	498, 523	1
	8	Strings	70, 71, 183, 425	1
	9	Math Library	474	1
	10	Hand tracing ******		
****** **Not required to show**			Total	4

Making Decisions

Chap	Section	Topic	Line Location	Pts
4	1	Relational Operators	Throughout	
	2	if	193, 224, 256, 287, 289, 291, 292, 311, 313, 315, 316, 336, 338, 341, 343, 377, 386, 434, 472, 475, 572	1
	4	If-else	286, 310, 321, 335, 348, 373, 401, 449, 485	1
	5	Nesting	286-294, 310-318, 335-345, 373-391	1
	6	If-else-if	515	1
	7	Flags *****		
	8	Logical operators	115, 193-195, 198-200, 206, 224-226, 229-231, 238, 270, 256-258, 261-263, 286-289, 291-293, 297, 310-313, 315-317, 321, 335-339, 341-344, 348, 428-430, 434-436, 474, 475,	1
	11	Validating user input	192, 204, 223, 235, 255, 267, 285, 309, 334	1
	13	Conditional Operator	125	1
	14	Switch	297	1
****** **Not required to show**			Total	8

Loops and Files

Chap	Section	Topic	Line Location	Pts
5	1	Increment/Decrement	114, 145, 146, 155, 156, 168, 169, 364, 365, 374, 383, 402, 407, 440, 473, 497, 503, 506, 533, 555, 568, 571, 590	1
	2	While	428	1
	5	Do-while	99, 187, 188, 218, 219, 250, 251, 468, 469, 588	1
	6	For loop	145, 146, 155, 156, 168, 169, 364, 365, 374, 383, 402, 407, 497, 503, 506, 533, 555, 568, 571	1
	11	Files input/output both	163-176, 528-548, 550-557	2
	12	No breaks in loops ******		
****** **Not required to show**			Total	6

Functions

Chap	Section	Topic	Line Location	Pts
6	3	Function Prototypes	26-47	4
	5	Pass by Value	143, 153, 163, 178, 280, 304, 329, 356, 394, 423, 458, 494, 513, 528, 550, 563, 582, 601, 605	4
	8	return	602, 606	4
	9	returning boolean	298, 300, 322, 325, 349, 352, 379, 388, 451, 454, 487, 490	4
	10	Global Variables	23 (2-D Array Column Dimension)	XXX
	11	static variables	426	4
	12	defaulted arguments	47	4
	13	pass by reference	163, 513, 528, 550	4
	14	overloading	601+605	5
	15	exit() function	609	4
******** Not required to show**			Total	37

Arrays

Chap	Section	Topic	Line Location	Pts
7	1-6	Single Dimensioned Arrays	66, 72	3
	7	Parallel Arrays	60+62, 61+63	2
	8	Single Dimensioned as Function Arguments	528, 550, 563, 582	2
	9	2 Dimensioned Arrays	60-63	2
	12	STL Vectors	64	2
		Passing Arrays to and from Functions	143, 153, 163, 178, 356, 394, 423, 458, 494, 528, 550, 563, 582	5
		Passing Vectors to and from Functions	513, 528	5
******** Not required to show**			Total	21

Searching and Sorting Arrays

Chap	Section	Topic	Line Location	Pts
8	3	Bubble Sort	544	4
	3	Selection Sort	538	4
	1	Linear or Binary Search	373	4
******** Not required to show**			Total	12