Project 2

Mastermind & Battleship

Joseph Levin
CSC-17A
Section 43950
Spring 2015
6/5/2015

This write up contains an introduction which consists of a description of Mastermind and Battleship and why these games were chosen for the project. It summarizes the project with statistics including satisfaction of criteria, what variables, constructs, and processes were implemented, and notes on exceptional areas of the project(i.e, challenges encountered during the development process). It describes the development process with example inputs/outputs, flowcharts and pseudocode, and details of major variables. The write up lists which concepts were used from the textbook, and finally includes the documented code.

	Table of Contents
1	Introduction
2	Summary 1. Program Statistics 2. Areas of Note in Development
3	Description 1. Program Walkthrough 2. Flowcharts 3. Variables/Structures/Classes 4. Concepts Implemented
4	References
5	Code Documentation

1 Introduction: Mastermind

Mastermind is a two-player board game in which turns are taken by alternating between the roles of a *codebreaker* and a *codemaker*. The codemaker chooses a pattern—be it a pattern of colored marbles or simply numbers—and the codebreaker attempts to guess the pattern within a limited number of turns. The codebreaker is given clues in the form of being told how many correct elements (i.e, red marbles or the number three) his guess has as well as how many of those elements are in the correct position of the pattern. It's important to note that a guess could include all the right colors or numbers, but none of them could be in the right position.

The game of Mastermind makes a good fit for a computer game because the role of the codemaker is easily adapted to be played by a computer. A computer is perfectly capable of coming up with random patterns, and is just as fit at telling a person how close to that pattern they are. A computer would also never take out its frustrations on the codebreaker if he/she demonstrations proficiency at decipher its patterns. As a matter of fact, a computer may be preferable to playing with a friend, if one has lousy friends.

Mastermind is an outwardly simple game. Given a secret pattern, figure out the pattern…it is the kind of thing a person might accidentally get correct without realizing they were playing a game in the first place. However as with any good game, it offers an experience that leaves the player satisfied and feeling good. Everybody likes being correct, especially when it comes packaged with a slight taste of having bested someone. With Mastermind, the codebreaker is given just this kind of opportunity.

Battleship

Battleship is another two player board game. Players each have a grid, in which they secretly place ships. The players take turns guessing at the location of each other's ships in an attempt to destroy all of their opponent's ships before theirs are destroyed.

Battleship shares the advantages for adapting into a computer game that Mastermind does. It's easy to imagine the computer taking on the role of an opponent taking guesses at where you hid your ships. While this iteration of Battleship is a simple system with random guesses taken by the computer, it also has strong potential for developing an AI based around educated guesses.

An area of note about this version of Battleship is the lack of multi-length ships. All ships are 1 spot "long" as opposed to various sizes that a typical Battleship game has. While this was admittedly due to time constraints, it's worth mentioning that there do exist variations of Battleship that feature a majority of 1 length ships (namely the variation called Salvo).

2 Summary

2.1 Statistics	
Program Length (lines of code).	1059
# Classes	3
# Structures	4
# Variables	32
# Functions	29

2.1 Areas of Note in Development

At the end of the day the most significant obstacle in the development of the project was my own indecisiveness. I debated with myself over a few different ways of implementing how to handle a user's guesses as well as where to keep track of certain elements such as how many attempts the user has taken. I settled on the current iteration for Mastermind because I felt the structure-within-structure approach suited the logic behind a user's guesses very well. It also allowed for a simple system for keeping track of accuracy of guesses, and comparing how many guesses have been taken to how many are allowed. Also, structures were very beneficial as it made implementing a form of stat tracking almost trivial when combined with binary file I/O.

As for Battleship, I enjoyed working with classes in terms of implementation. Having less experience with objects the start was disorienting but as the member functions began to take shape it became increasingly easy to test my code as I wrote it as opposed to constantly having to pass test parameters into function calls in a driver. There was a certain level of satisfaction to implementing concepts such as polymorphism. The act of overriding abstract and base cases of member functions gave a feeling of complexity I hadn't seen yet and made it fun to work with.

The modular nature of objects made the project significantly easier to alter as problems with my initial planning popped up. For example, my current system of having the BaseBS class more or less by the player class and the DerivBS class be the computer class led me to mistaken have the player target their own board due to having written the target member function for BaseBS in terms of the player taking a turn. Once I realized my mistake, it was as simple as copy and pasting the contents of the two versions of target in order to swap the logic and get it working perfectly.

Considering the amount of carry-over from the first project, this second project didn’t take unreasonably long to accomplish. I started working on it about a week a half ago, although most of the heavy work was done over the weekend.

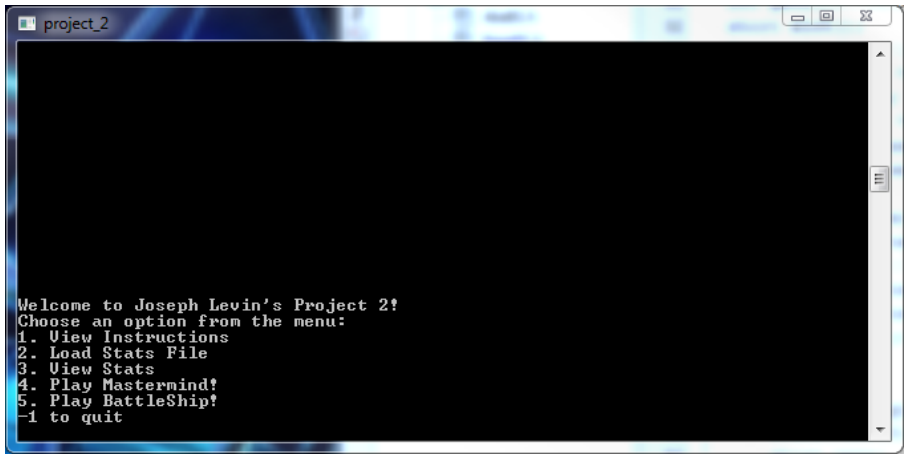
3 Program Description

3.1 Program Walkthrough

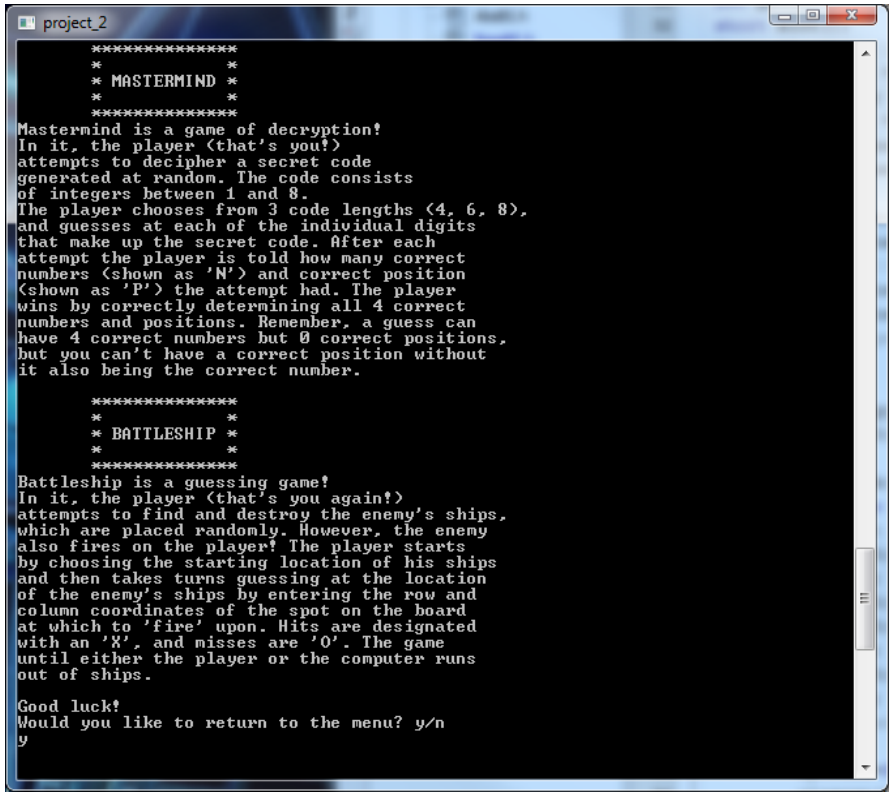
The main menu offer s5 options to the user.

1. View Instructions
2. Load Stats File
3. View Stats
4. Play Mastermind
5. Play Battleship

-1 to quit at any time



If the user inputs 1, they are shown a brief explanation of how Mastermind is played. It should be noted that the original game allows 6 different colors, but I forgot this and chose 8 instead.



Choosing 2 prompts the user to enter the name of a stats file to load.

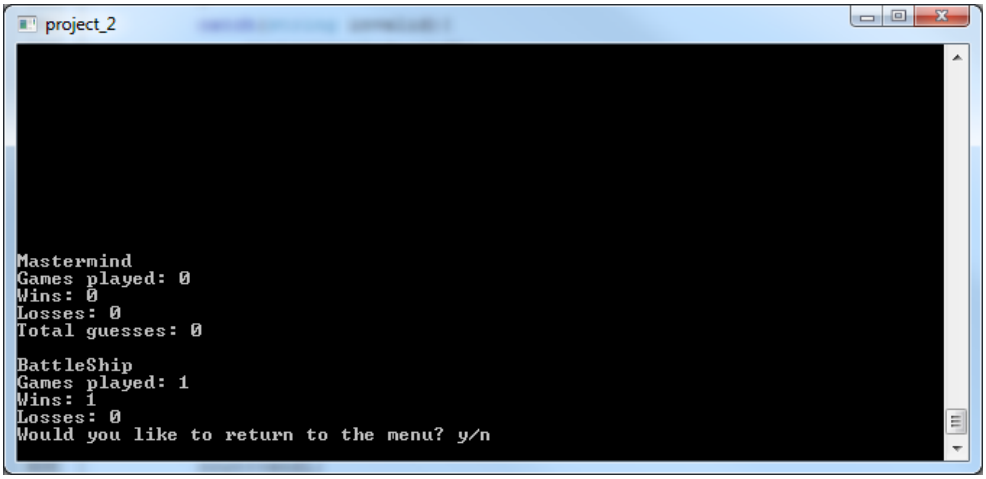
Error displayed if not found

If found, file is read into a Stats struct via binary file IO

Choosing 3 displays the contents of a loaded Stats struct

Default values are all zero

The stats structure is declared at the top of main and is initialized to default values and deleted at the end of the program

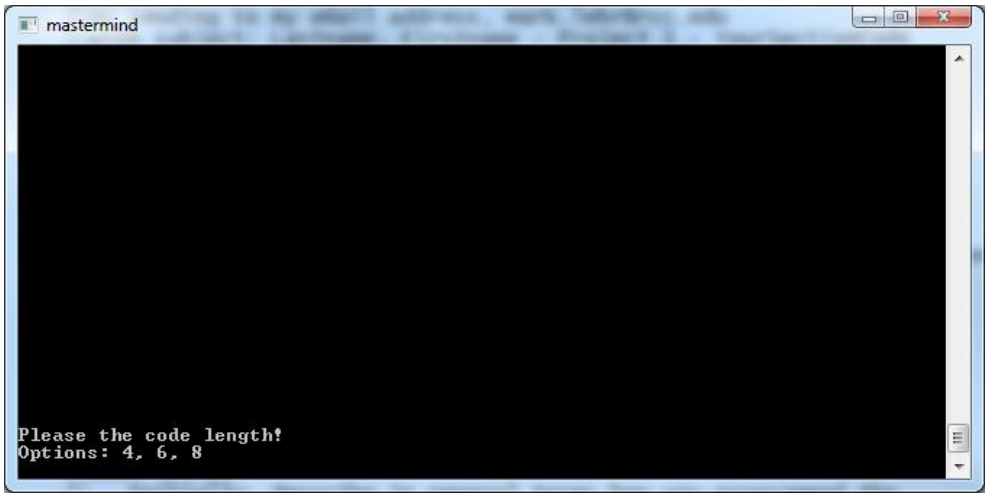


Choosing 4 launches Mastermind

Player is prompted to choose code length

Note: if length=l, then odds of getting right answer first try is 1/(8^l).

i.e, l=4, odds=1/(8^4)=0.0002

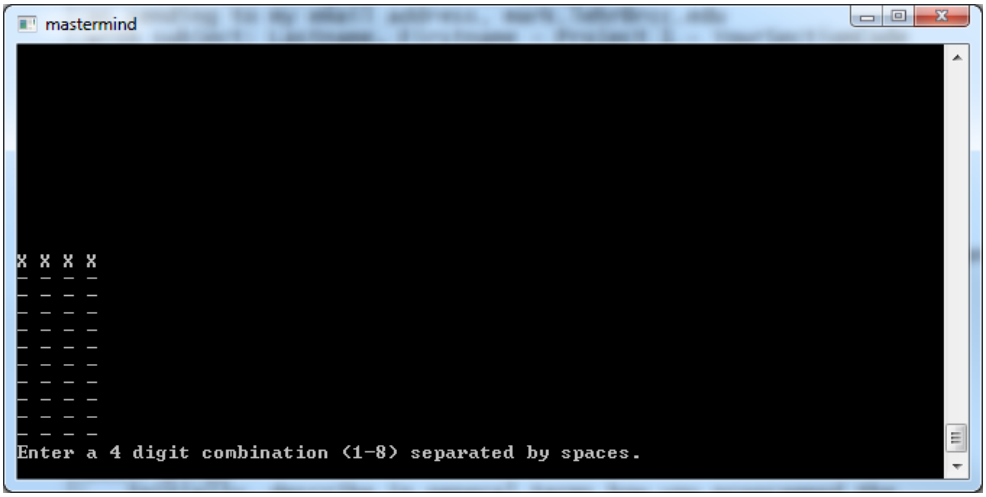


After choosing length (4) board is displayed

Input will only accept 4 digits separated by spaced

i.e, input = 1 2 3 4

the is told how many correct numbers (N=#) and correct positions (P=#)

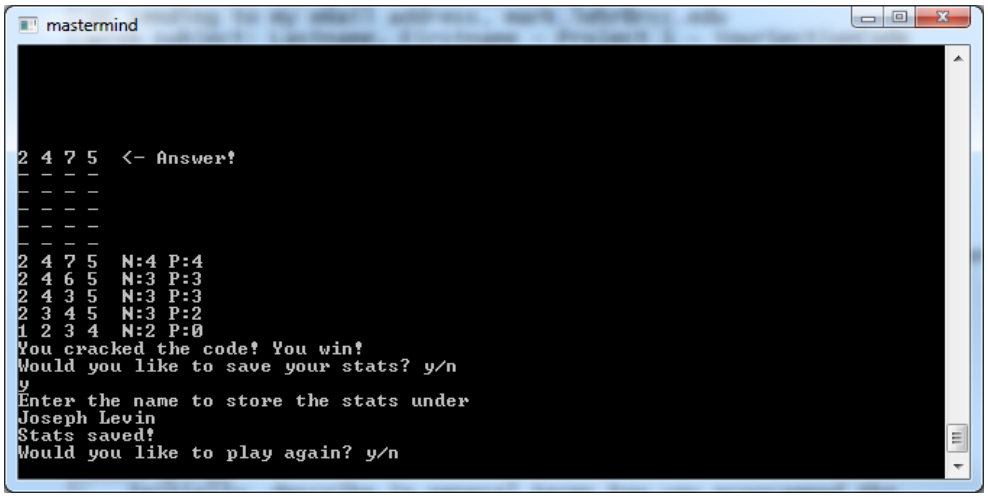


Upon correctly guessing, answer is revealed and a victory message is displayed

User is prompted to save stats

i.e, input y to save

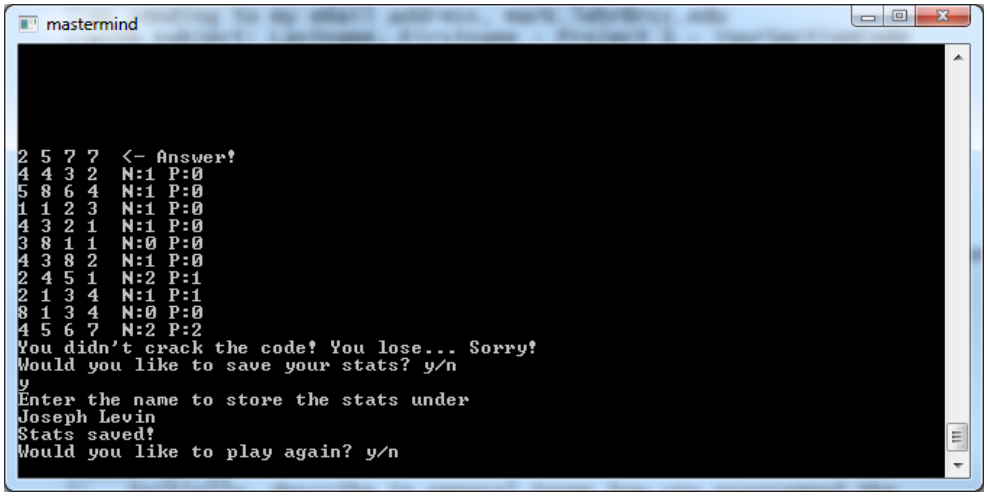
input <file name>



Similarly, losing reveals answer and a defeat message is displayed

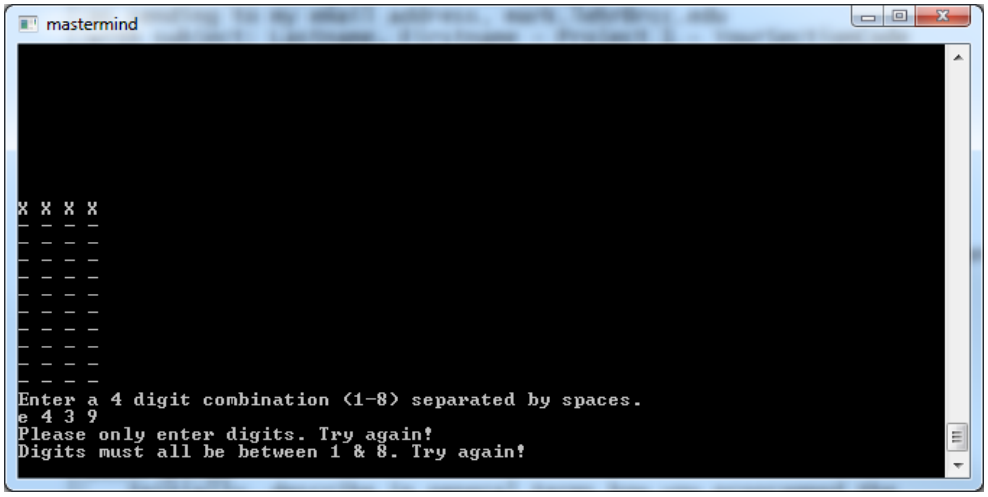
User is still prompted to save stats

If file name is reused, old stats are overwritten



Input will only accept digits 1 – 8.

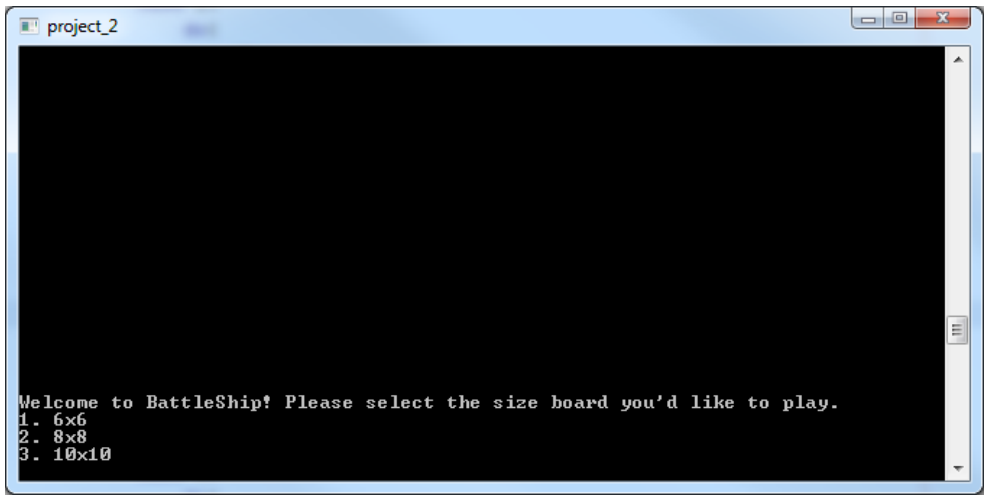
Non digits will give appropriate error, as will digits outside of range



Choosing 5 in the menu will launch Battleship

Battleship is implemented using abstract classes with polymorphism.

First the player is prompted to choose the dimension of the board. An exception is used to handle invalid input



A member function is called to place the player's ships one at time until there are ships equal to the dimension of the board.

In the case of the computer, a derived place function is used to randomly place its ships

```
project_2
  ---------------------------
4 !   !   !   !   !   !   !
  ---------------------------
5 !   !   !   !   !   !   !
  ---------------------------
Enter the row coordinate for where to begin ship 5
4
Enter the column coordinate for where to begin ship 5
5
    0   1   2   3   4   5
  ---------------------------
0 !   !   !   !   !   !   !
  ---------------------------
1 !   ! + ! + !   !   !   !
  ---------------------------
2 !   !   !   !   !   !   !
  ---------------------------
3 !   !   !   ! + ! + !   !
  ---------------------------
4 !   !   !   !   !   ! + !
  ---------------------------
5 !   !   !   !   !   !   !
  ---------------------------
Enter the row coordinate for where to begin ship 6
```

The BaseBS class contains a member function for displaying the board in a grid format. It shows the location of the ships as +, hits as X, misses as O.

The DerivBS class contains a member function radar() that displays a modified version of the instance's board that excludes ships from being displayed

```
project_2
Player board (5 ships remain)
    0   1   2   3   4   5
  ---------------------------
0 ! X !   ! O !   !   !   !
  ---------------------------
1 !   ! + ! O !   !   !   !
  ---------------------------
2 ! O ! O ! + ! O !   !   !
  ---------------------------
3 ! O ! O ! O ! + ! O !   !
  ---------------------------
4 !   !   ! O !   ! + ! O !
  ---------------------------
5 !   ! O !   !   !   ! + !
  ---------------------------

Radar (3 enemy ships remain)
    0   1   2   3   4   5
  ---------------------------
0 ! O !   !   !   !   !   !
  ---------------------------
1 !   !   ! O ! X !   ! O !
  ---------------------------
2 ! X ! O !   ! O !   !   !
  ---------------------------
3 !   !   !   !   !   ! O !
  ---------------------------
4 !   !   ! O !   !   !   !
  ---------------------------
5 ! X !   ! O ! O ! O !   !
  ---------------------------
Enter the row (vertical) component of the coordinate you wish to fire upon
```

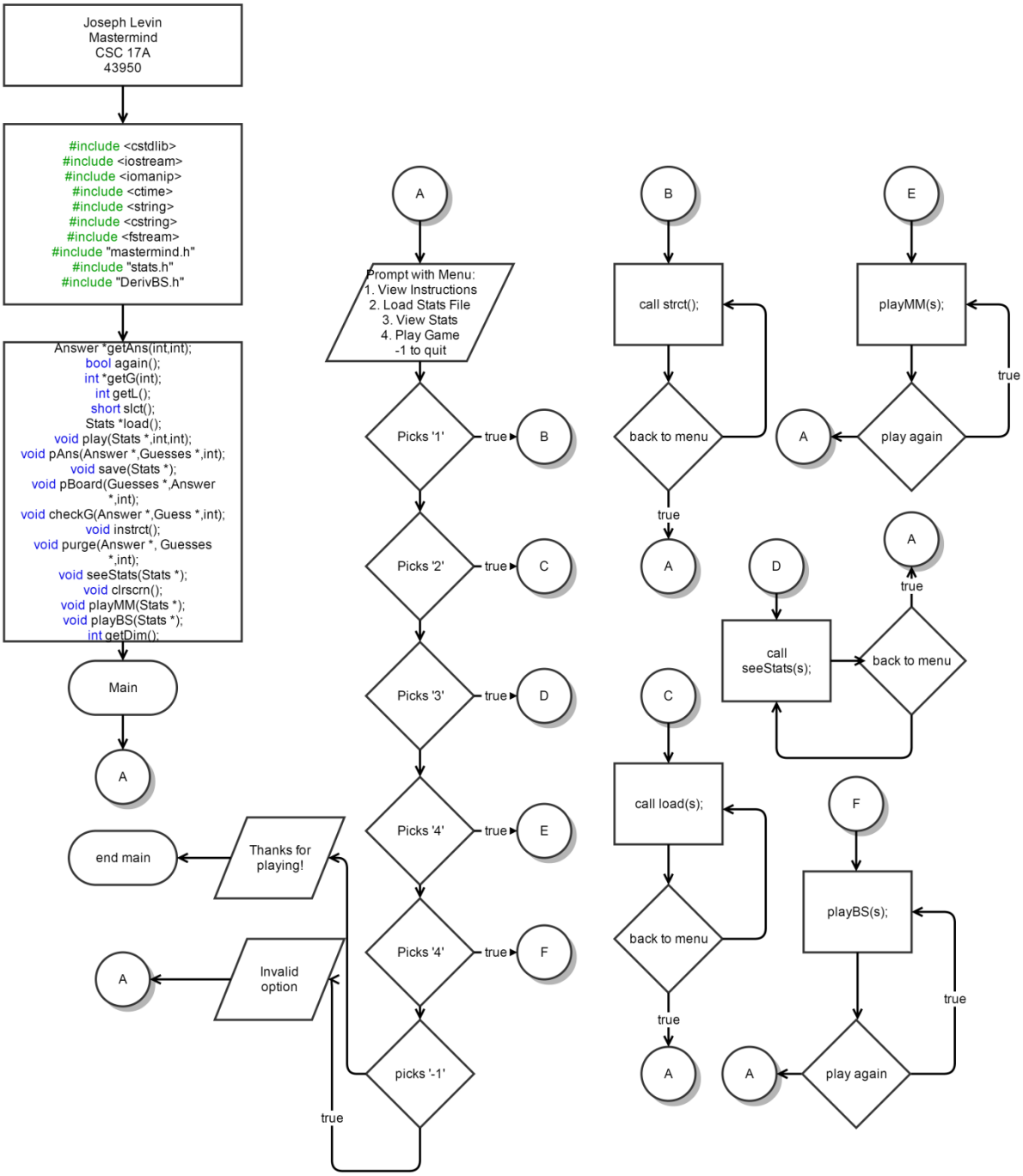
Gameplay continues until either the player or the computer runs out of ships.

Saving works identically to the implementation in Mastermind.

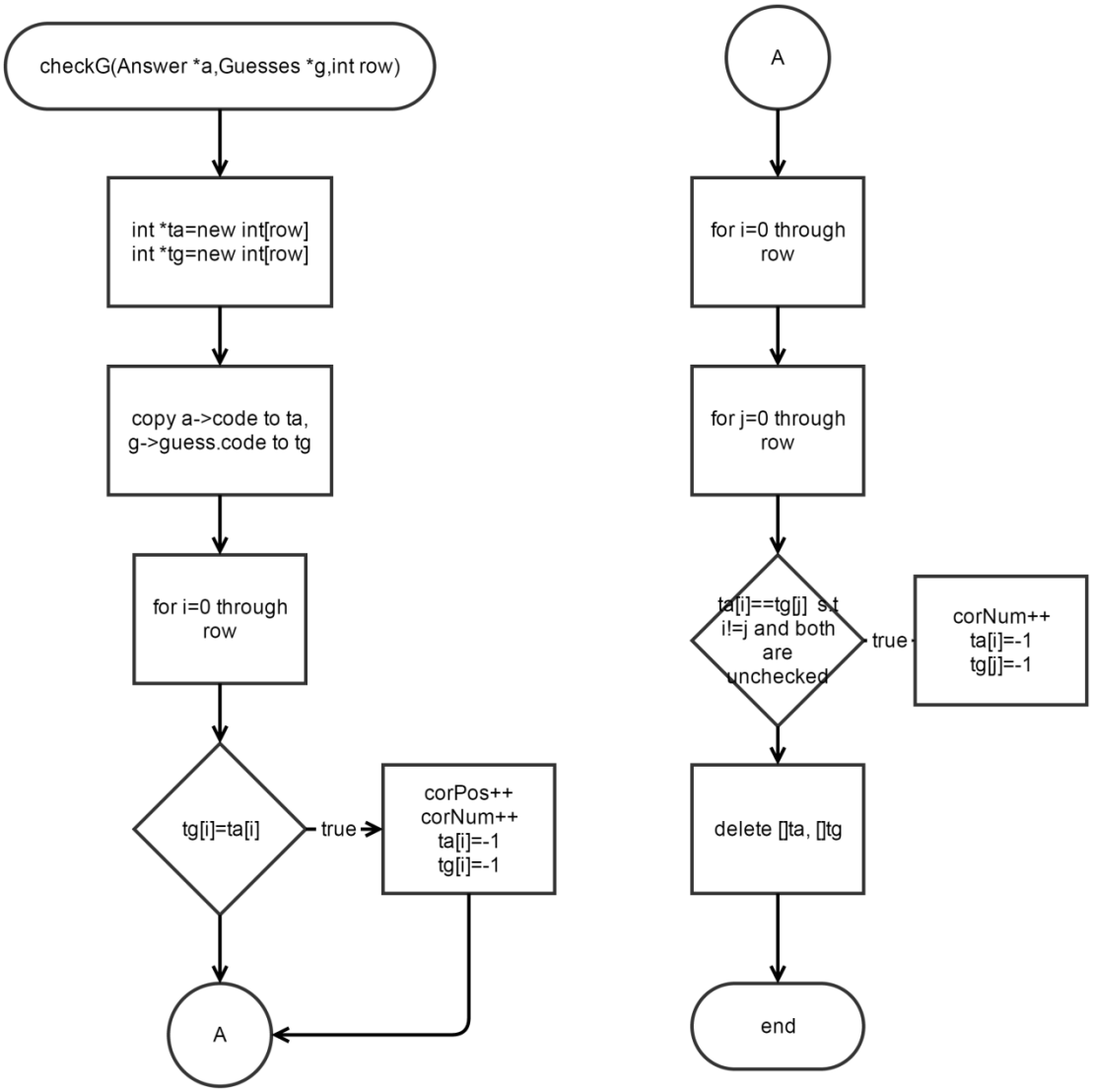
```
project_2
  ---------------------------
0 ! X !   ! O ! O ! O ! O !
  ---------------------------
1 ! O ! X ! O ! O !   ! O !
  ---------------------------
2 ! O ! O ! X ! O !   !   !
  ---------------------------
3 ! O ! O ! O ! + ! O ! O !
  ---------------------------
4 !   ! O ! O ! O ! + ! O !
  ---------------------------
5 ! O ! O !   !   ! O ! + !
  ---------------------------

Radar (0 enemy ships remain)
    0   1   2   3   4   5
  ---------------------------
0 ! O ! O ! O ! X ! O ! X !
  ---------------------------
1 ! O ! O ! O ! X !   ! O !
  ---------------------------
2 ! X ! O !   ! O !   !   !
  ---------------------------
3 ! X !   ! O !   !   ! O !
  ---------------------------
4 ! O ! O ! O ! O ! O !   !
  ---------------------------
5 ! X !   ! O ! O ! O !   !
  ---------------------------
You win! Congratulations!
Would you like to save your stats? y/n
y
Enter the name to store the stats under
Joseph
```

3.2 Flowcharts

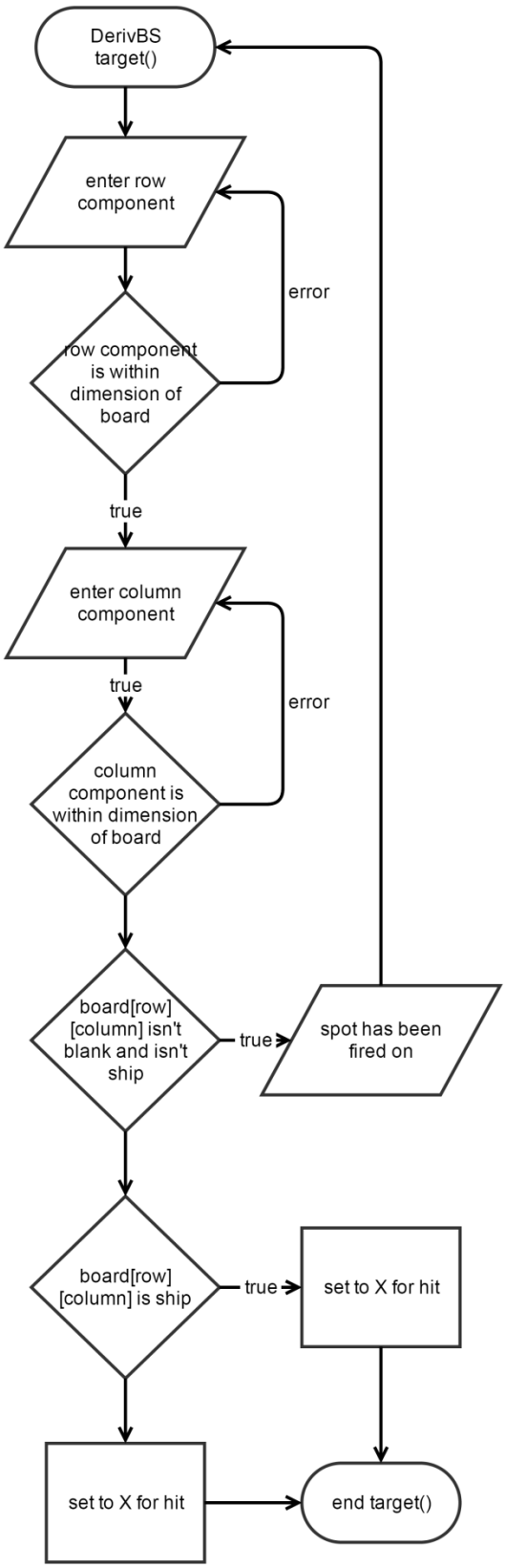
Main menu:

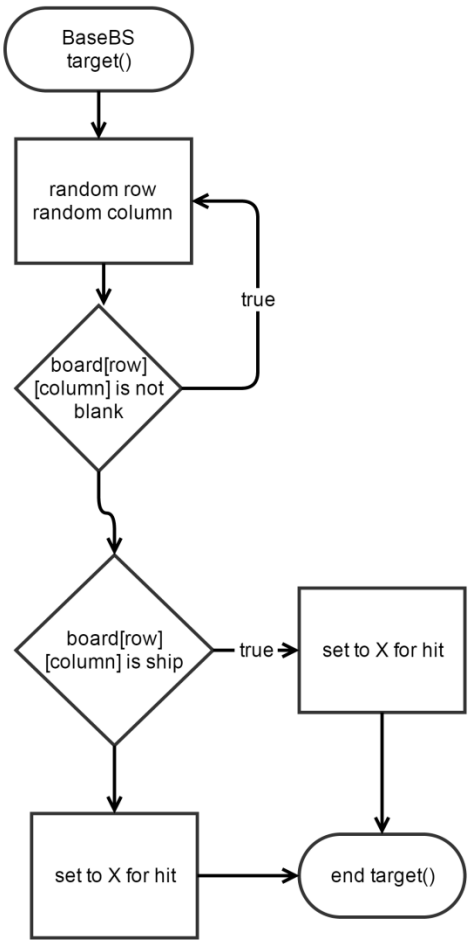


Mastermind Guess Checking Function (checkG):



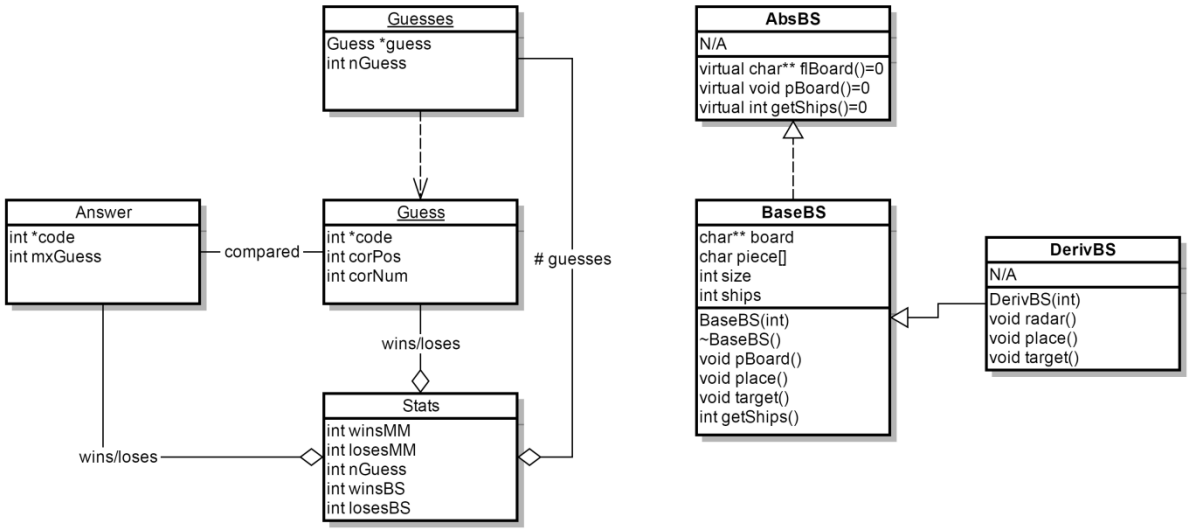
Battleship Target member functions





3.3 Structures, Classes and Variables

3.3.1 Structure and Classes UML:



3.3.2 Structures/Classes/Variable Listing:

Answer	int *code stores answer pattern int mxGuess stores maximum guesses allowed
Guesses	Guess *guess stores individual guess structures (for each turn) int nGuess stores number of guesses taken
Guess	int *code stores guess attempt int corPos # of positions guessed correctly during a turn (determines win) int corNum # of numbers guessed correctly during a turn
Stats	int winsMM stores number of games won in Mastermind int losesMM stores number of games lost in Mastermind int nGuess stores total guesses taken between all games in Mastermind

AbsBS	public: virtual char** flBoard() virtual void pBoard() virtual int getShips()
BaseBS	protected: char** board for storing the game boards char piece[4] stores the game pieces. Ships, hit marker, miss marker, and blank tile int size stores the dimension of the board. i.e, if board is 6x6 then size == 6 int ships stores the number of ships belonging to either the player or computer public: BaseBS(int) Constructor. Takes in int and sets size equal to it ~BaseBS() Destructor. Deletes all components to char** array void pBoard() outputs board with all pieces displayed including ships void place() prompts player to place all of his ships. Amount of ships is equal to size void target() randomly chooses a spot on the player's board. Serves as computer's turn int getShips() Returns remaining ships as int.
DerivBS	public: DerivBS(int) Inherits base class constructor without changes void radar() Displays board but with position of ships omitted. For player's targeting void place() Randomly places all of computers ships void target() Prompts player to target a spot on computer's board. Serves as player's turn

3.4 Concepts Implemented:

Chapter	
9	9.1 Getting Address of a Variable 9.2 Pointer Variables 9.3 Arrays/Pointers 9.5 Initializing Pointers 9.6 Comparing Pointers 9.7 Pointers as Function Parameters 9.8 Dynamic Memory Allocation 9.9 Returning pointers from functions
10	10.1 Character Testing 10.2 Character Case Conversion 10.4 Library Functions for Working With C-Strings 10.5 C-String/Numeric Conversion Functions 10.7 C++ string Class
11	11.1 Abstract Data Types 11.2 Combining Data into Structures 11.3 Accessing Structure Members 11.4 Initializing a Structure 11.5 Arrays of Structures 11.6 Nested Structures 11.7 Structures as Function Arguments 11.8 Returning a Structure From a Function 11.9 Pointers to Structures
12	12.4 More Detailed Error Testing 12.7 Binary Files 12.8 Creating Records with Structures
13	13.1 Procedural and Object-Oriented Programming 13.2 Introduction to Classes 13.3 Defining an Instance of a Class 13.4 Private Members 13.5 Separating Class Specification from Implementation 13.6 Inline Member Functions 13.7 Constructors 13.8 Passing Arguments to Constructors 13.9 Destructors 13.10 Overloading Constructors
15	15.1 Inheritance 15.2 Protected Members and Class Access 15.3 Constructors and Destructors in Base and Derived Classes 15.4 Redefining Base Class Functions 15.5 Class Hierarchies 15.6 Polymorphism and Virtual Member Functions 15.7 Abstract Base Classes and Pure Virtual Functions
16	16.1 Exceptions

If the area of implementation is not located in main, the location will be specified in **bold**

Concepts	Implementation	Line of Code
9.2 Pointer Variables	checkG(Answer *a,Guesses *g,int row)	143
9.3 Arrays/Pointers	checkG(Answer *a,Guesses *g,int row)	143
9.5 Initializing Pointers	checkG(Answer *a,Guesses *g,int row)	143
9.6 Comparing Pointers	checkG(Answer *a,Guesses *g,int row)	151
9.7 Pointers as Function Parameters	play(Stats *s,int m,int r)	310
9.8 Dynamic Memory Allocation	Stats *s=new Stats;	52
9.9 Returning pointers from functions	*getG(int row)	181
10.1 Character Testing	*getG(int row)	205
10.2 Character Case Conversion	bool again()	460
10.4 Lib. Funct. for Working w/ C-Strings	*getG(int row)	228
10.5 C-String/Numeric Conversion Functions	*getG(int row)	228
10.7 C++ string Class	save(Stats *s)	394
11.1 Abstract Data Types	**stats.h** struct Stats	13
11.2 Combining Data into Structures	**mastermind.h** struct Guess	25
11.3 Accessing Structure Members	*getAns(int max, int row)	126
11.4 Initializing a Structure	play(Stats *s,int m,int r)	315
11.5 Arrays of Structures	**mastermind.h** struct Guesses	40

11.6 Nested Structures	**mastermind.h** struct Guesses	40
11.7 Structures as Function Arguments	play(Stats *s,int m,int r)	310
11.8 Returning a Structure From a Function	*getAns(int max, int row)	132
11.9 Pointers to Structures	play(Stats *s,int m,int r)	317
12.7 Binary Files	save(Stats *s)	396
12.8 Creating Records with Structures	save(Stats *s)	397
13.1 Procedural and Object-Oriented Programming	**BaseBS.h** class BaseBS	15
13.2 Introduction to Classes	**BaseBS.h** class BaseBS	15
13.3 Defining an Instance of a Class	**BaseBS.cpp**	17-136
13.4 Private Members	**BaseBS.h**	15
13.5 Separating Class Specification from Implementation	**BaseBS.h/BaseBS.cpp** **DerivBS.h/DerivBS.cpp**	See files
13.6 Inline Member Functions	**BaseBS.h** int getShips()	44
13.7 Constructors	**BaseBS.h** BaseBS(int)	29
13.8 Passing Arguments to Constructors	**BaseBS.cpp** BaseBS(int)	17
13.9 Destructors	**BaseBS.cpp** ~BaseBS()	27
15.1 Inheritance	**BaseBS.h** Class BaseBS:AbsBS	15
15.2 Protected Members and Class Access	**BaseBS.h**	15
15.3 Constructors and Destructors in Base and Derived Classes	**DerivBS.cpp** DerivBS::DerivBS(int s):BaseBS(s){;}	15
15.4 Redefining Base Class Functions	**DerivBS.cpp** DerivBS::place()	43

15.5 Class Hierarchies	**AbsBS.h/BaseBS.h/DerivBS.h** **AbsBS -> BaseBS -> DerivBS**	See files
15.6 Polymorphism and Virtual Member Functions	**AbsBS.h** virtual char** flBoard()	14
15.7 Abstract Base Classes and Pure Virtual Functions	**AbsBS.h** class AbsBS	11
16.1 Exceptions	**main/BaseBS.cpp** playBS(Stats *s)/ place()	600/90

4 References

Textbook: C++ From Control Structures through Objects, 8th ed. Tony Gaddis. (2015)

5 Code Documentation

Main

```
/*
 * File:   main.cpp
 * Author: Joseph Levin
 * C++ Project 2 - Spring 2015 43950
 * 6/5/2015
 */

//System Libraries
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <ctime>
#include <string>
#include <cstring>
#include <fstream>

using namespace std;

//User Libraries
#include "mastermind.h"
#include "DerivBS.h"
#include "stats.h"

//Global Constants

//Function Prototypes
Answer *getAns(int,int);//generates an answer
bool again();//replay function
bool menu();//returns to menu
int *getG(int);//gets guess from user
int getL();//returns length for code combination
short slct();
Stats *load();//loads stats structure
void play(Stats *,int,int);//launches the game
void pAns(Answer *,Guesses *,int);//prints the answer
void save(Stats *);//saves the stats structure as binary file
void pBoard(Guesses *,Answer *,int);//prints the game board
void checkG(Answer *,Guess *,int);//checks guess against answer
void instrct();
void purge(Answer *, Guesses *,int);//deletes all structures related to a game
void seeStats(Stats *);//Displays stats
void clrscrn();//clears screen
void playMM(Stats *);//driver for mastermind game
void playBS(Stats *);//driver for battleship
int getDim();//gets dimension for BattleShip board

//Begin
int main(int argc, char** argv) {
    srand(time(0));
    short optn;
```

```
    bool optnChk;//for checking if menu option is valid
    Stats *s=new Stats;
    //initialize all Stats variables to 0
    s->gamesMM=0;
    s->winsMM=0;
    s->losesMM=0;
    s->gamesBS=0;
    s->winsBS=0;
    s->losesBS=0;
    s->nGuess=0;
    //display menu
    do{
        optnChk=false;
        clrscrn();
        //get menu selection
        optn=slct();
        switch(optn){
            //display instructions
            case 1:
                do{
                    clrscrn();
                    instrct();
                }while(!menu());
                break;
            //load a stats struct
            case 2:
                s=load();
                break;
            //view stats
            case 3:
                do{
                    clrscrn();
                    seeStats(s);
                }while(!menu());
                break;
            //play Mastermind
            case 4:
                do{
                    clrscrn();
                    playMM(s);
                }while(again());
                break;
            //play BattleShip
            case 5:
                      do{
                          clrscrn();
                          playBS(s);
                      }while(again());
                      break;
                  //exit case
                  case -1:
                      cout<<"Thanks for playing!"<<endl;
                      break;
                  //invalid option
                  default:
                      cout<<"Invalid option selected..."
                              "it's not that hard, really."<<endl;
```

```
                    break;
            }
        }while(optn!=-1);
        //clean up!
        delete s;

        //It's over!
        return 0;
    }

    //!Clear screen function outputs a ton of new lines in order to clear
    //!the command prompt to look nice
    void clrscrn(){
        for(int i=0; i<100; i++)
            cout<<endl;
    }//end
    //!getAns dynamically creates a Answer struct, fills the code array with
    //!random integers 0-9, sets nGuess to max (max # of guesse) and then returns
    Answer *getAns(int max, int row){
        Answer *answer=new Answer;
        answer->mxGuess=max;//max number of guesses allowed
        answer->code=new int[row];//the combination is 'row' digits long
        for(int i=0;i<row;i++)
            answer->code[i]=rand()%8+1;//fill code with 1-9
        return answer;
    }//end
    //!checkG takes in an Answer struct. It copies the contents of the code array
    //!into another int array and compares each element to a Guess array
    //!It changes the elments in the temp int array to -1 as it finds matches
    //!in order to ensure no duplicate matches are found
    //!if an element in the guess matches the position and number of the answer
    //!both the correct number and correct position counters are incremented
    //!otherwise if only the number is matched then the correct number indication
    //!is incremented. Man this is a long description.
    void checkG(Answer *a,Guesses *g,int row){
        int *ta=new int[row];//temp array to store answer
        int *tg=new int[row];//temp array to store guess
        for(int i=0;i<row;i++){
            ta[i]=a->code[i];//copy answer to temp answer
            tg[i]=g->guess[g->nGuess-1].code[i];//copy guess to temp guess
        }
        //loop through arrays to check for both correct position and correct number
        for(int i=0;i<row;i++){
            if(tg[i]==ta[i]){
                g->guess[g->nGuess-1].corPos++;
```

```
                    g->guess[g->nGuess-1].corNum++;
                    ta[i]=-1;//no duplicates
                    tg[i]=-1;
                }
            }
            //loop through temp answer again
            for(int i=0;i<row;i++){
                //loop through guess
                for(int j=0;j<row;j++){
                    //check for same number
                    if(ta[i]==tg[j]&&i!=j&&tg[j]!=-1&&ta[i]!=-1){//same num, diff pos
                        g->guess[g->nGuess-1].corNum++;
                        ta[i]=-1;
                        tg[j]=-1;
                    }
                }
            }
            delete []ta;
            delete []tg;
        }//end
        /*!
         * getG prompts the user to enter a 4 digit combination, separated by spaces
         * it stores it as a string, and checks if every odd element is a digit 1-8
         * and then checks if every even is a space
         * if this is true, it then converts each odd element into an int and stores
         * it in an int array and returns this array
         * takes an int to specify how big an int array to create
         */
        int *getG(int row){
            //declare variables
            int *temp=new int[row];//will be returned
            string guess;//for character checking
            bool check;//input validation
            cout<<"Enter a "<<row
                    <<" digit combination (1-8) separated by spaces."<<endl;
            do{
                check=true;
                getline(cin,guess);
                if(guess.size()!=2*row-1){//row digits plus spaces between
                    cout<<"You must enter "<<row<<" digits separated by spaces."
                            <<"Try again!"<<endl;
                    check=false;;
                }
                else{
                    //every odd must be 1-8, and every even must be space
                    for(int i=0;i<2*row-1;i++){
                        //check odds for all digits
                        if(i%2==0&&!isdigit(guess[i])){
                            check=false;
```

```
                        cout<<"Please only enter digits. Try again!"<<endl;
                    }
                    //check if all digits are 1-8
                    else if(i%2==0&&(atoi(&guess[i])>8||atoi(&guess[i])<1)){
                        check=false;
                        cout<<"Digits must all be between 1 & 8. Try again!"<<endl;
                        break;
                    }
                    //check if every even is space
                    else{
                        if(!isspace(guess[i])&&!isdigit(guess[i])){
                            check=false;
                            cout<<"Every digit must be "
                                    <<"separated by a space. Try again!"<<endl;
                        }
                    }
                }
            }
        }while(!check);
        //copy digits into int array to return
        for(int i=0;i<row;i++)
            temp[i]=atoi(&guess[2*i]);
        //int array ready to return
        return temp;
    }//end
    /*!
     * pBoard takes in pointers to an Answer and Guess, and an int
     * specifying the code length. It prints out dashed lines indicating
     * how many guesses remain and also prints out all previous guesses
     */
    void pBoard(Guesses *g, Answer *a, int r){
        //X's represent the mystery code
        for(int i=0;i<r;i++)
            cout<<"X ";
        cout<<endl;
        //-'s represent spaces left for remaining guesses
        for(int i=0;i<(a->mxGuess)-(g->nGuess)+1;i++){
            for(int j=0;j<r;j++)
                cout<<"- ";
            cout<<endl;
        }
        //all previous guesses
        if(g->nGuess-1!=0){
        for(int i=g->nGuess-2;i>=0;i--){
            for(int j=0;j<r;j++)
                cout<<g->guess[i].code[j]<<" ";
            cout<<" N:"<<g->guess[i].corNum
            <<" P:"<<g->guess[i].corPos<<endl;
        }
        }
     //finished
```

```
    }//end
    /*!
     * getL prompts the user to enter either 4, 6, or 8. It then returns
     * the value selected as an int. No parameters
     */
    int getL(){
        int l;//length of code
        bool check=false;//error checking flag
        cout <<"Please the code length!"<<endl<<"Options: 4, 6, 8"<<endl;
        do{
            cin>>l;
            if(cin.fail()||(l!=4&&l!=6&&l!=8)){
                cin.clear();
                cin.ignore(256,'\n');
                cout<<"Sorry! You have to choose "
                        "either 4, 6, or 8. Try again!"<<endl;
            }
            else
                check=true;
        }while(!check);
        return l;
    }//end
    /*!
     * pAns prints the answer code
     */
    void pAns(Answer *a,Guesses *g,int r){
        //Show Answer
        for(int i=0;i<r;i++)
            cout<<a->code[i]<<" ";

        cout<<" <- Answer!"<<endl;
        //-'s represent spaces left for remaining guesses
        for(int i=0;i<(a->mxGuess)-(g->nGuess);i++){
            for(int j=0;j<r;j++)
                cout<<"- ";
            cout<<endl;
        }
        //all previous guesses
        if(g->nGuess!=0){
        for(int i=g->nGuess-1;i>=0;i--){
            for(int j=0;j<r;j++)
                cout<<g->guess[i].code[j]<<" ";
            cout<<" N:"<<g->guess[i].corNum
            <<" P:"<<g->guess[i].corPos<<endl;
        }
        }
    }//end
    /*!
     * play is the main driver for Mastermind gameplay. It handles turn taking,
     * win/lose checks, as well as stat saving. Void function, takes in an integer
     * to determine max # of guesses and an integer to determine code length
     */
```

```
310     void play(Stats *s,int m,int r){
311         //generate answer
312         cin.clear();
313         cin.ignore(265,'\n');
314         char optn;
315         Answer *a=getAns(m,r);
316         //generate and prepare Guesses
317         Guesses *g=new Guesses;
318         g->nGuess=0;
319         g->guess=new Guess[m];
320         for(int i=0;i<m;i++){
321             g->guess[i].corNum=0;
322             g->guess[i].corPos=0;
323         }
324         //loop until win/lose
325         do{
326             clrscrn();
327             g->nGuess++;
328             pBoard(g,a,r);
329             g->guess[g->nGuess-1].code=getG(r);
330             checkG(a,g,r);
331         }while(g->nGuess<a->mxGuess&&g->guess[g->nGuess-
    1].corPos!=r);
332         //display answer and determine win or lose and increment
    stats counters
333         clrscrn();
334         pAns(a,g,r);
335         s->gamesMM++;
336         if(g->guess[g->nGuess-1].corPos==r){
337             cout<<"You cracked the code! You win!"<<endl;
338             s->winsMM++;
339         }
340         else{
341             cout<<"You didn't crack the code! You lose...
    Sorry!"<<endl;
342             s->losesMM++;
343         }
344         s->nGuess+=g->nGuess;
345         cout<<"Would you like to save your stats? y/n"<<endl;
346         do{
347             cin>>optn;
348             if(tolower(optn)!='y'&&tolower(optn)!='n')
349                 cout<<"Sorry, that's not a valid option. Please try
    again."<<endl;
350             else if(tolower(optn)=='y'){
351                 save(s);
352                 cout<<"Stats saved!"<<endl;
353             }
354             else
355                 cout<<"Stats not saved."<<endl;
356         }while(tolower(optn)!='y'&&tolower(optn)!='n');
357         //clean up
358         purge(a,g,m);
359     }//end
360     /*!
361      * purge takes in an Answers and Guesses pointer and deletes all
    dynamically
```

```
 * allocated elements of the struct, as well as the structs themselves
 * and then points them to nullptrs
 */
void purge(Answer *a,Guesses *g,int m){
    delete []a->code;
    a->code=NULL;
    delete a;
    for(int i=0;i<m;i++){
        delete []g->guess[i].code;
        g->guess[i].code=NULL;
    }
    delete g->guess;
    g->guess=NULL;
    delete g;
    g=NULL;
}
/*!
 * save takes in a Stats struct pointer. It prompts the user for a name
 * to store the stats struct under, and then writes the contents of the Stats
 * to a binary file. Returns void
 */
void save(Stats *s){
    cin.clear();
    cin.ignore(256,'\n');
    ofstream out;//file stream
    cout<<"Enter the name to store the stats under"<<endl;
    string name;
    getline(cin,name);
    out.open(name.c_str(),ios::binary);
    out.write(reinterpret_cast<char *>(s),sizeof(Stats));
    out.close();
}//end
/*!
 * load prompts the user for a name, and attempts to open a file
 * with that name. If found, it reads the contents into a Stats structure
 * and returns it.
 */
Stats *load(){
    cin.clear();
    cin.ignore(256,'\n');
    string name;
    ifstream in;
    cout<<"Enter the name of the person whose stats to load"<<endl;
    do{
        getline(cin,name);
        in.open(name.c_str(),ios::binary);
        if(in.fail()){
            cout<<"Name not found. Try again!"<<endl;
        }
        else{
            Stats *s=new Stats;
            in.read(reinterpret_cast<char *>(s),sizeof(Stats));
```

```
                in.close();
                return s;
            }
        }while(in.fail());
    }
    //!slct serves to take in input for menu selection, performs error checks
    //!and then returns the value if it passes checks
    short slct(){
        short pick; //for menu selection
        bool check=false;
        cout<<"Welcome to Joseph Levin's Project 2!"<<endl;
        cout<<"Choose an option from the menu: "<<endl
                <<"1. View Instructions"<<endl
                <<"2. Load Stats File"<<endl
                <<"3. View Stats"<<endl
                <<"4. Play Mastermind!"<<endl
                <<"5. Play BattleShip!"<<endl
                <<"-1 to quit"<<endl;
        do{
            cin>>pick;
            if(cin.fail()||pick<=0&&pick!=-1||pick>5){//error checking
                cin.clear();
                cin.ignore(256,'\n');
                cout<<"Error. Invalid selection. Try again."<<endl;
            }
            else
                check=true;//valid input
        }while(!check);
        return pick;
    }
    /*!
     * again asks the user if they would like to play another game
     * it returns true if the user does, false if they do not
     */
    bool again(){
        bool check=false;
        char pick;
        cout<<"Would you like to play again? y/n"<<endl;
        do{
            cin>>pick;
            if(cin.fail()||tolower(pick)!='y'&&tolower(pick)!='n'){//only accepts //y or n as input
                cin.clear();
                cin.ignore(256,'\n');
                cout<<"Error. Invalid selection. Try again."<<endl;
            }
            else if(tolower(pick)=='y'){//user wants to repeat
                check=true;
                cin.clear();
                cin.ignore(256,'\n');
                return true;
            }
            else{ //user does not want to repeat
                cin.clear();
```

```
                cin.ignore(256,'\n');
                check=true;
            return false;
            }
        }while(!check);

    }//end
    /*!
     * menu prompts the user if they want to return to the main menu.
     * if the user types 'y', it returns true
     * if the user types 'n', it returns false
     */
    bool menu(){
        bool check=false;
        char pick;
        cout<<"Would you like to return to the menu? y/n"<<endl;
        do{
            cin>>pick;
if(cin.fail()||tolower(pick)!='y'&&tolower(pick)!='n'){//only accepts
                cin.clear();
//y or n as input
                cin.ignore(256,'\n');
                cout<<"Error. Invalid selection. Try again."<<endl;
            }
            else if(tolower(pick)=='y'){//user wants to repeat
                check=true;
                cin.clear();
                cin.ignore(256,'\n');
                return true;
            }
            else{ //user does not want to repeat
                cin.clear();
                cin.ignore(256,'\n');
                check=true;
            return false;
            }
        }while(!check);
    }
    /*!
     * seeStats takes in Stats pointer. It displays the elements within,
     * (wins/loses/total guesses) and also calculates win percentage, and
     * correct guess percentage
     */
    void seeStats(Stats *s){
        if(s->gamesBS!=0||s->gamesMM!=0){
            cout<<"Mastermind"<<endl;
            cout<<"Games played: "<<s->gamesMM<<endl;
            cout<<"Wins: "<<s->winsMM<<endl;
            cout<<"Losses: "<<s->losesMM<<endl;
            cout<<"Total guesses: "<<s->nGuess<<endl;
            cout<<endl;
            cout<<"BattleShip"<<endl;
            cout<<"Games played: "<<s->gamesBS<<endl;
            cout<<"Wins: "<<s->winsBS<<endl;
```

```
            cout<<"Losses: "<<s->losesBS<<endl;
        }
        else
            cout<<"Stats File is empty!"<<endl;
    }
    /*!
     * instrct displays the rules for Code Breaker (based on Mastermind)
     */
    void instrct(){
        cout<<"       **************"<<endl;
        cout<<"       *            *"<<endl;
        cout<<"       * MASTERMIND *"<<endl;
        cout<<"       *            *"<<endl;
        cout<<"       **************"<<endl;
        cout<<"Mastermind is a game of decryption!"<<endl
                <<"In it, the player (that's you!)"<<endl
                <<"attempts to decipher a secret code"<<endl
                <<"generated at random. The code consists"<<endl
                <<"of integers between 1 and 8."<<endl
                <<"The player chooses from 3 code lengths (4, 6, 8),"<<endl
                <<"and guesses at each of the individual digits"<<endl
                <<"that make up the secret code. After each"<<endl
                <<"attempt the player is told how many correct "<<endl
                <<"numbers (shown as 'N') and correct position "<<endl
                <<"(shown as 'P') the attempt had. The player"<<endl
                <<"wins by correctly determining all 4 correct"<<endl
                <<"numbers and positions. Remember, a guess can"<<endl
                <<"have 4 correct numbers but 0 correct positions,"<<endl
                <<"but you can't have a correct position without"<<endl
                <<"it also being the correct number."<<endl;
        cout<<endl;
        cout<<"       **************"<<endl;
        cout<<"       *            *"<<endl;
        cout<<"       * BATTLESHIP *"<<endl;
        cout<<"       *            *"<<endl;
        cout<<"       **************"<<endl;
        cout<<"Battleship is a guessing game!"<<endl
                <<"In it, the player (that's you again!)"<<endl
                <<"attempts to find and destroy the enemy's ships,"<<endl
                <<"which are placed randomly. However, the enemy"<<endl
                <<"also fires on the player! The player starts"<<endl
                <<"by choosing the starting location of his ships"<<endl
                <<"and then takes turns guessing at the location"<<endl
                <<"of the enemy's ships by entering the row and"<<endl
```

```
                    <<"column coordinates of the spot on the board"<<endl
                    <<"at which to 'fire' upon. Hits are designated"<<endl
                    <<"with an 'X', and misses are 'O'. The game"<<endl
                    <<"until either the player or the computer runs"<<endl
                    <<"out of ships."<<endl;
            cout<<endl<<"Good luck!"<<endl;
        }//end
        /*!
         * playMMis the main driver for Mastermind.
         * Parameters: a stats struct for storing stats
         */
        void playMM(Stats *s){
            clrscrn();
            int l=getL();
            play(s,10,l);
        }
        /*!
         * playBS is the main driver for BattleShip.
         * Parameters: a stats struct for storing stats
         */
        void playBS(Stats *s){
            int dim;//for storing dimension of board
            char optn;//for stat saving selection
            bool sConf=false;//confirming size is valid
            cout<<"Welcome to BattleShip!"
                    " Please select the size board you'd like to play."<<endl;
            cout<<"1. 6x6"<<endl;
            cout<<"2. 8x8"<<endl;
            cout<<"3. 10x10"<<endl;
            do{
                try{
                    dim=getDim();
                    sConf=true;
                }
                catch(string invalid){
                    cout<<invalid<<endl;
                }
            }while(!sConf);
            //initialize BaseBS (for player) and DerivBS (for computer) of dim
            BaseBS player(dim);
            DerivBS comp(dim);
            //randomly place computer's ships
            comp.place();
            //place players ships
            player.place();
            //begin rounds
            do{
                clrscrn();
                //display board and radar
                cout<<"Player board ("<<player.getShips()<<" ships remain)"<<endl;
                player.pBoard();
                cout<<endl;
```

```
            cout<<"Radar ("<<comp.getShips()<<" enemy ships remain)"<<endl;
            comp.radar();
            //targeting round
            player.target();
            comp.target();
        }while(comp.getShips()!=0&&player.getShips()!=0);
        clrscrn();
        //Display boards final time
        cout<<"Player board ("<<player.getShips()<<" ships remain)"<<endl;
        player.pBoard();
        cout<<endl;
        cout<<"Radar ("<<comp.getShips()<<" enemy ships remain)"<<endl;
        comp.radar();
        //Determine victor and increment stats counters
        s->gamesBS++;
        if(comp.getShips()==0){
            cout<<"You win! Congratulations!"<<endl;
            s->winsBS++;
        }
        else{
            cout<<"You lose. Sorry!"<<endl;
            s->losesBS++;
        }
        cout<<"Would you like to save your stats? y/n"<<endl;
        do{
            cin>>optn;
            if(tolower(optn)!='y'&&tolower(optn)!='n')
                cout<<"Sorry, that's not a valid option. Please try again."<<endl;
            else if(tolower(optn)=='y'){
                save(s);
                cout<<"Stats saved!"<<endl;
            }
            else
                cout<<"Stats not saved."<<endl;
        }while(tolower(optn)!='y'&&tolower(optn)!='n');
    }//end
    /*!
     * getDim prompts user for dimension for BattleShip board. It will throw
     * an exception if invalid, otherwise returns the value
     */
    int getDim(){
        int dim;
        cin>>dim;
        if(dim<1||dim>3||cin.fail()){
            string invalid="Invalid selection for dimension. Try again.";
            throw invalid;
        }
        //Reassign dim to the corresponding dimension
        if(dim==1){
            dim=6;
        }
```

```
        else if(dim==2){
            dim=8;
        }
        else{
            dim=10;
        }
        return dim;
}
```

Mastermind.h

```
    /*
     * File:   mastermind.h
     * Author: Joseph
     *
     * Created on April 26, 2015, 7:47 PM
     */
    #ifndef MASTERMIND_H
    #define MASTERMIND_H

    /*!
     * Answer stores the correct code in an int array and the max guesses allowed
     */
    struct Answer{
        //!stores the integer combination
        int *code;
        //!keeps track of maximum guesses allowed
        int mxGuess;
    };

    /*!
     * Guess stores a guess in an int array, the number of correct positions
     * associated with the guess and the number of correct numbers
     */
    struct Guess{
        //!stores the guess
        int *code;
        //!tallies the number of correct positions in a guess
        int corPos;
        //!tallies the number of correct numbers in a guess
        int corNum;
    };

    /*!
     * Guesses stores a pointer a Guess (one for each turn) and
     * the total number of guesses taken
     */
    struct Guesses{
        //!array of guess attempts
        Guess *guess;
        //!how many guesses have been taken
        int nGuess;
    };
#endif  /* MASTERMIND_H *
```

Stats.h

```
/*
 * File:   stats.h
 * Author: Joseph Levin
 * C++ Project 2 - Spring 2015 43950
 * 6/5/2015
 */

#ifndef STATS_H
#define STATS_H
/*!
 * Stats stores the number of wins, losses, and number of guesses
 */
struct Stats{
    //!number of wins for Mastermind
    int winsMM;
    //!number of loses for Mastermind
    int losesMM;
    //!number of guesses for Mastermind
    int nGuess;
    //!number of wins for BattleShip
    int winsBS;
    //!number of loses for BattleShip
    int losesBS;
    //!tracks games played
    int gamesBS;
    int gamesMM;
};
#endif  /* STATS_H */
```

AbsBS.h

```

/*
 * File:   AbsBS
 * Author: Joseph Levin
 * C++ Project 2 - Spring 2015 43950
 * 6/5/2015
 */

#ifndef ABSBS_H
#define ABSBS_H
//!Abstract class for the Battleship board
class AbsBS{
    public:
        //!2D char array to store the board
        virtual char** flBoard()=0;
        //!outputs the board
        virtual void pBoard()=0;
        //!returns ships remaining
        virtual int getShips()=0;
};
#endif  /*  ABSBS_H */
```

BaseBS.h

```
/*
 * File:   BaseBS.h
 * Author: Joseph Levin
 * C++ Project 2 - Spring 2015 43950
 * 6/5/2015
 */

#ifndef BASEBS_H
#define BASEBS_H
#include "AbsBS.h"

/*!
 * BaseBS is the base class for the battleship board.
 */
class BaseBS:AbsBS{
    protected:
        //!for storing the board
        char** board;
        //!X is hit, O is miss, + is ship, <space> is empty
        char piece[4]={'X','O','+', ' '};
        //!dimension of board
        int size;
        //!ships on board
        int ships;
    public:
        //!Constructor for base battleship board. Takes in an integer for
        //!the dimension of the board (area is nxn))
        BaseBS(int);
        //!Destructor for the board
        ~BaseBS();
        //!For initially filling board
        char** flBoard();
        //!For outputting the board formatted to show ships
        void pBoard();
        //!place handles the ship placing procedure for the player.
        //!it checks to make sure the given coordinates are within
        //!the acceptable range of size, and then sets the given tile
        //!to the + char to represent a ship has been placed
        void place();
        //!target handles the process of targeting a spot to fire on it
        void target();
        //!getShips returns the value of the ships member variable
        int getShips();
};
#endif  /* BASEBS_H */
```

BaseBS.cpp

```
/*
 * File:   BaseBS.cpp
 * Author: Joseph Levin
 * C++ Project 2 - Spring 2015 43950
 * 6/5/2015
 */

//Libraries
#include <iostream>
#include <iomanip>//board formatting
#include <cstdlib>
#include <vector>//for rand
#include "BaseBS.h"

using namespace std;

BaseBS::BaseBS(int s){
    size=s;
    board=flBoard();
}
BaseBS::~BaseBS(){
    for(int i=0;i<size;i++){
        delete []board[i];
    }
    delete []board;
}
/*!
 * flBoard() initializes the board with all empty spaces
 */
char** BaseBS::flBoard(){
    char** board = new char*[size];
    for (int i=0;i<size; i++){
        board[i]=new char[size];
    }
    for (int i=0;i<size;i++) {
        for (int j=0;j<size;j++)
            board[i][j]=piece[3];
    }

    return board;
}
void BaseBS::pBoard(){
    cout<<"  ";
    for (int i=0;i<size;i++) {
        cout<<setw(3)<<setfill(' ')<<i<<" ";
    }
    cout<<endl;
    for (int i=0;i<size; i++) {
        cout<<"  "<<setw(size*4+1)<<setfill('-')<<'-'<<endl;
        cout<<i<<" ";
        for (int j=0;j<size;j++) {
            cout<<"|"<<setw(2)<<setfill(' ')<<board[i][j]<<" ";
        }
        cout<<"| "<<endl;
    }
```

```
        cout<<"  "<<setw(size*4+1)<<setfill('-') << '-' << endl;
    }
    void BaseBS::place(){
        bool conf1,conf2,conf3;//error buffers
        ships=0;//begin with no ships placed
        int row, col;//for checking the given coordinate
        for(int i=0;i<size;i++){//will place size # of ships
            do{
                pBoard();
                cout<<"Enter the row coordinate for where to begin ship "
                        <<ships+1<<endl;
                //check confirms to false
                conf1=false;
                conf2=false;
                conf3=false;
                do{//gets the row coordinate
                    cin>>row;
                    if (cin.fail()||(row<0||row>size-1)){//size-1 denotes edge
                        cin.clear();
                        cin.ignore(256,'\n');
                        cout<<"Error. Invalid input."<<endl;
                    } else
                        conf1=true;//row coordinate is acceptable
                } while(cin.fail()||row<0||row>size-1||!conf1);
                cout<<"Enter the column coordinate for where to begin ship "
                        <<ships+1<<endl;
                do{
                    cin>>col;
                    if (cin.fail()||(col<0||col>size-1)){
                        cin.clear();
                        cin.ignore(256,'\n');
                        cout<<"Error. Invalid input."<<endl;
                    } else
                        conf2=true;//column coordinate is acceptable
                } while(cin.fail()||col<0||row>size-1||!conf1);
                if(board[row][col]!=piece[3]){//piece[3] == blank space
                    cout << "This spot is already occupied" << endl;
                }
                else {
                    board[row][col]=piece[2];//piece[2]== '+'
                    ships++;//added ship to board
                    conf3=true;//process is completed successfully
                }
            }while(!conf3);
        }
    }
    //!target for BaseBS handles the computer firing on the player's board
    //!it is a glorified random number generator
    void BaseBS::target(){
        bool confirm=false;
        int row,col;
        //randomly fires at spots until it
```

```
        do{
            row=(rand()%size);
            col=(rand()%size);
            //piece[3]==blank, piece[2]==ship
            if(board[row][col]==piece[3]||board[row][col]==piece[2])
                confirm=true;
        } while(!confirm);
        //piece[2]==ship,piece[0]==X
        if (board[row][col]==piece[2]){
            board[row][col]=piece[0];
            ships--;
            //piece[1]==O
        } else
            board[row][col]=piece[1];
    }
    //!getShips returns the value of the ships member variable
int BaseBS::getShips(){return ships;}
```

DerivBS.h

```
    /*
     * File:   DerivBS.h
     * Author: Joseph Levin
     * C++ Project 2 - Spring 2015 43950
     * 6/5/2015
     */
    #ifndef DERIVBS_H
    #define DERIVBS_H

    #include "BaseBS.h"

    /*!
     * DerivBS is a derived class of BaseBS. It contains addition functions
     * for handling the computer logic
     */
    class DerivBS:public BaseBS{
        public:
            //!constructor for DerivBS calls BaseBS constructor
            DerivBS(int);
            //!displays a version of the board with ships masked
            void radar();
            //!place for DerivBS randomly places ships for computer
            void place();
            //!target for DerivBS randomly fires on spots for computer
            void target();
    };
#endif  /* DERIVBS_H */
```

DerivBS.cpp

```cpp
/*
 * File:   DerivBS.cpp
 * Author: Joseph Levin
 * C++ Project 2 - Spring 2015 43950
 * 6/5/2015
 */

#include "DerivBS.h"
#include<iostream>
#include<iomanip>
#include <cstdlib>

using namespace std;


//!constructor for DerivBS calls BaseBS constructor
DerivBS::DerivBS(int s):BaseBS(s){;}
//!displays a version of the board with ships masked
void DerivBS::radar(){
    cout<<"  ";
    for (int i=0;i<size;i++) {
        cout<<setw(3)<<setfill(' ')<<i<<" ";
    }
    cout<<endl;
    for (int i=0;i<size; i++) {
        cout<<"  "<<setw(size*4+1)<<setfill('-')<<'-'<<endl;
        cout<<i<<" ";
        for (int j=0;j<size;j++) {
            //if ship isn't at that spot, place piece at i,j
            if(board[i][j]!=piece[2]){
                cout<<"|"<<setw(2)<<setfill(' ')<<board[i][j]<<" ";
            }
            //if ship is at that spot, mask ship by placing blank spot instead
            else{
                cout<<"|"<<setw(2)<<setfill(' ')<<piece[3]<<" ";
            }
        }
        cout<<"| "<<endl;
    }
    cout<<"  "<<setw(size*4+1)<<setfill('-') << '-' << endl;
}
//!place for DerivBS randomly places ships for computer
void DerivBS::place(){
    int row, col;
    ships=0;
    for(int i=0;i<size;i++){
        //randomly picks spots until it finds one that isn't occupied already
        do{
            row=(rand()%size);
            col=(rand()%size);
        }while(board[row][col]!=piece[3]);//piece[3]==blank space
        board[row][col]=piece[2];//piece[2] == '+', ship
```

```
            ships++;
        }
    }
    //!target for DerivBS handles the player firing on the computer's board
    //!it prompts for the coordinates one component a time, checks for any errors
    //!and decides if it was a hit or miss and announces such
    void DerivBS::target(){
        int row, col;
        bool conf1,conf2,conf3;//error buffers
        do{
            conf1=false;
            conf2=false;
            conf3=false;
            cout<<"Enter the row (vertical) component of the coordinate "
                    <<"you wish to fire upon"<<endl;
            do{
                cin>>row;
                if (cin.fail()||row<0||row>size-1) {
                    cin.clear();
                    cin.ignore(256,'\n');
                    cout<<"Error. Invalid selection." << endl;
                } else
                    conf1=true;
            }while(cin.fail()||row<0||row>size-1||!conf1);
            cout<<"Enter the column (horizontal) component "
                    <<"of the coordinate you wish to fire upon"<<endl;
            do{
                cin>>col;
                if(cin.fail()||col<0||col>size-1){
                    cin.clear();
                    cin.ignore(256,'\n');
                    cout<<"Error. Invalid selection."<<endl;
                }else
                    conf2=true;
            }while(cin.fail()||col<0||col>size-1||!conf2);
            if(board[row][col]!=piece[3]&&board[row][col]!=piece[2])
                cout<<"This spot has been fired upon already."<<endl;
            else
                conf3=true;
        } while (!conf3);
        //if a ship was hit, replace with X and decrease remaining ships
        //and announce hit was successful
        if (board[row][col]==piece[2]){
            board[row][col]=piece[0];
            ships--;
        }
        //if a ship wasn't hit, replace with O and announce miss
        else{
            board[row][col]=piece[1];
        }
    }
```