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.

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
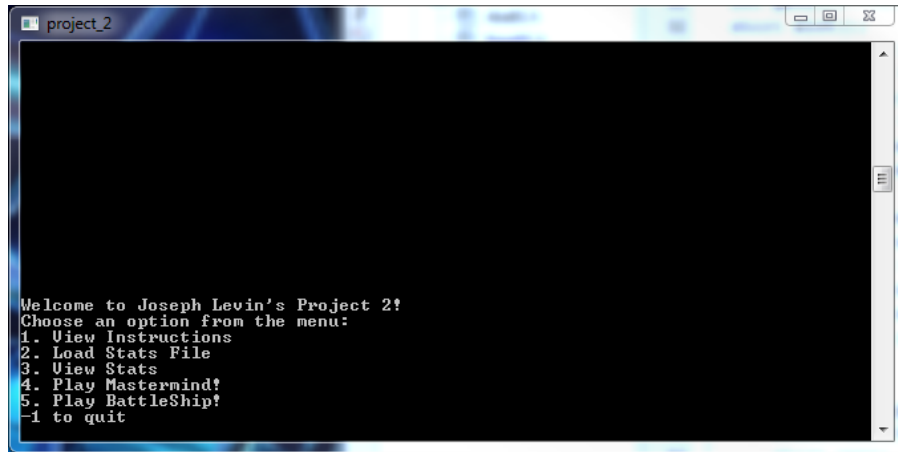
The main menu offer s5 options to the user.

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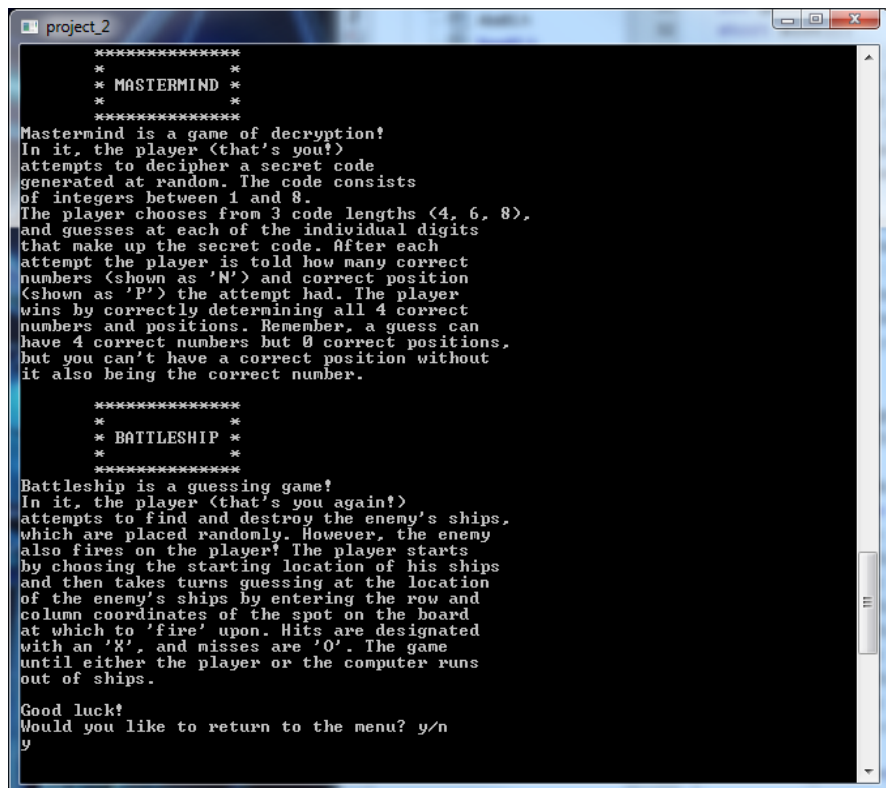
Choosing 2 prompts the user to enter the name of a stats file to load.

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


```
Mastermind
Games played: 0
Wins: 0
Losses: 0
Total guesses: 0

BattleShip
Games played: 1
Wins: 1
Losses: 0
Would you like to return to the menu? y/n
```
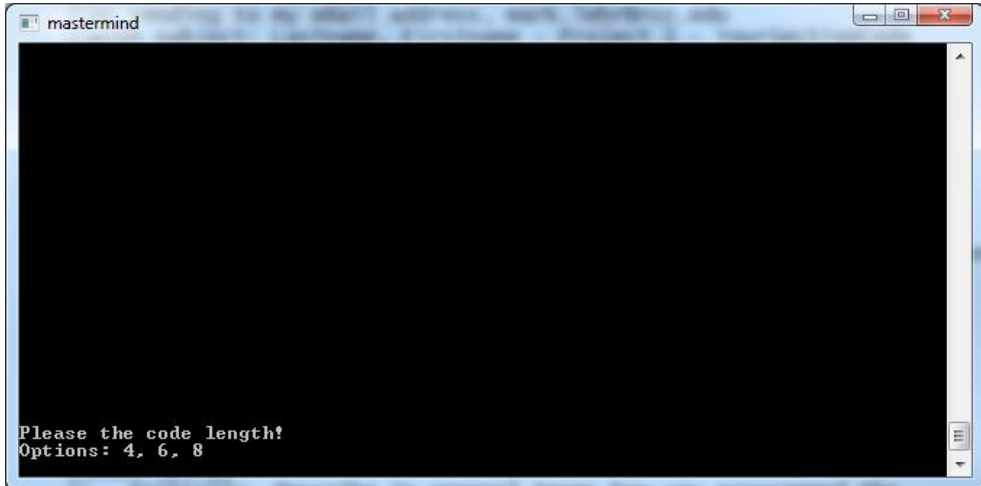
Choosing 4 launches Mastermind

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
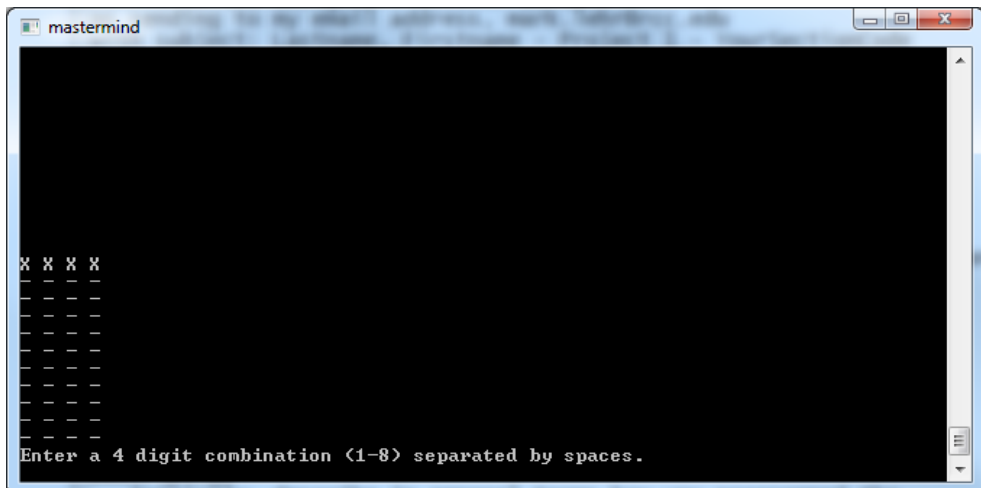

```
Please the code length!
Options: 4, 6, 8
```

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=#)
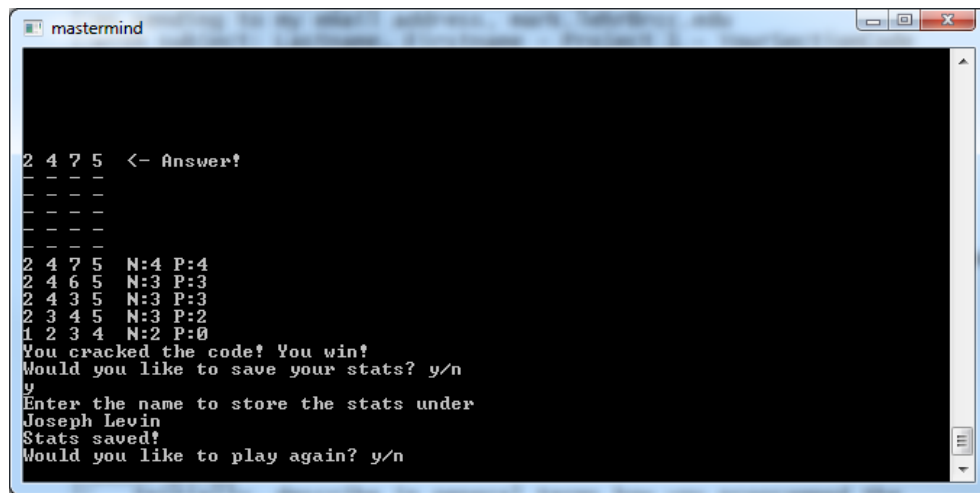

```
X X X X
- - - -
- - - -
- - - -
- - - -
- - - -
- - - -
- - - -
Enter a 4 digit combination (1-8) separated by spaces.
```

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>
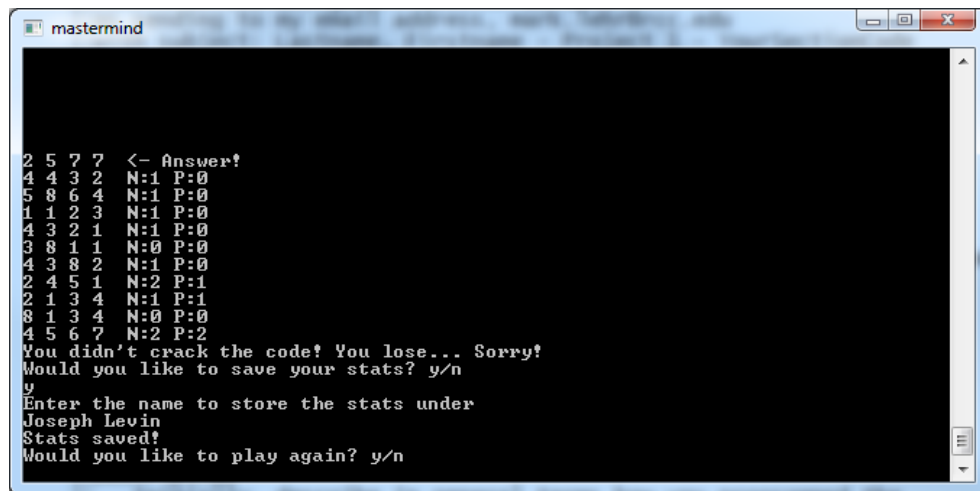
```
mastermind

2 4 7 5   <- Answer!
- - - -

- - - -

- - - -

- - - -
2 4 7 5   N:4 P:4
2 4 6 5   N:3 P:3
2 4 3 5   N:3 P:3
2 3 4 5   N:3 P:2
1 2 3 4   N:2 P:0
You cracked the code! You win!
Would you like to save your stats? y/n
y
Enter the name to store the stats under
Joseph Levin
Stats saved!
Would you like to play again? y/n
```

Similarly, losing reveals answer and a defeat message is displayed

User is still prompted to save stats
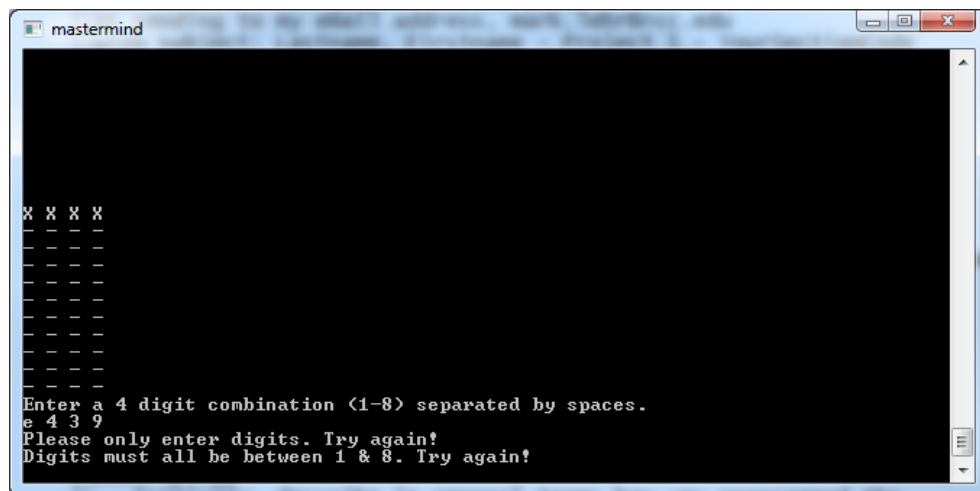
If file name is reused, old stats are overwritten

```
mastermind

2 5 7 7   <- Answer!
4 4 3 2   N:1 P:0
5 8 6 4   N:1 P:0
1 1 2 3   N:1 P:0
4 3 2 1   N:1 P:0
3 8 1 1   N:0 P:0
4 3 8 2   N:1 P:0
2 4 5 1   N:2 P:1
2 1 3 4   N:1 P:1
8 1 3 4   N:0 P:0
4 5 6 7   N:2 P:2
You didn't crack the code! You lose... Sorry!
Would you like to save your stats? y/n
y
Enter the name to store the stats under
Joseph Levin
Stats saved!
Would you like to play again? y/n
```

Input will only accept digits 1 – 8.

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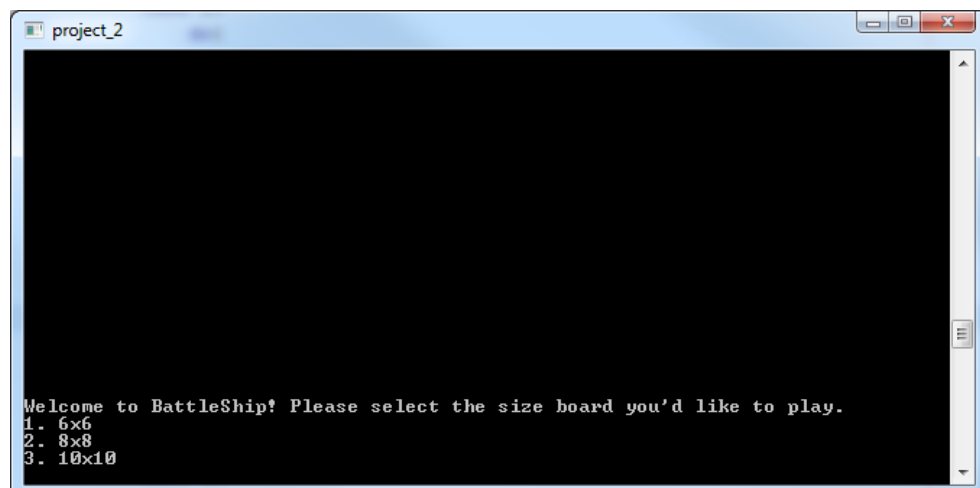
```
mastermind

X X X X
- - - -

- - - -

- - - -

- - - -

- - - -

- - - -
Enter a 4 digit combination (1-8) separated by spaces.
e 4 3 9
Please only enter digits. Try again!
Digits must all be between 1 & 8. Try again!
```

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

```
project_2

Welcome to BattleShip! Please select the size board you'd like to play.
1. 6x6
2. 8x8
3. 10x10
```

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

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

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

Saving works identically to the implementation in Mastermind.

3.2 Flowcharts

Main menu:

```
Joseph Levin
Mastermind
CSC 17A
43950
```

```
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <ctime>
#include <string>
#include <cstring>
#include <fstream>
#include "mastermind.h"
#include "stats.h"
#include "DerivBS.h"
```

```
Answer *getAns(int,int);
bool again();
int *getG(int);
int getL();
short slct();
Stats *load();
void play(Stats *,int,int);
void pAns(Answer *,Guesses *,int);
void save(Stats *);
void pBoard(Guesses *,Answer
*,int);
void checkG(Answer *,Guess *,int);
void instrct();
void purge(Answer *, Guesses
*,int);
void seeStats(Stats *);
void clrscrn();
void playMM(Stats *);
void playBS(Stats *);
int getDim();
```

Main

(A)

end main ← **Thanks for playing!**

(A) ← **Invalid option**

A

Prompt with Menu:
1. View Instructions
2. Load Stats File
3. View Stats
4. Play Game
-1 to quit

Picks '1' — true → (B)

Picks '2' — true → (C)

Picks '3' — true → (D)

Picks '4' — true → (E)

Picks '4' — true → (F)

picks '-1' — true

B

call strct();

back to menu — true → (A)

C

call load(s);

back to menu — true → (A)

D

call seeStats(s); — back to menu — true → (A)

E

playMM(s); — true

play again — true → (A)

F

playBS(s); — true

play again — true → (A)

Mastermind Guess Checking Function (checkG):

```
checkG(Answer *a,Guesses *g,int row)
```

```
int *ta=new int[row]
int *tg=new int[row]
```

```
copy a->code to ta,
g->guess.code to tg
```

```
for i=0 through
row
```

tg[i]=ta[i] —true→
```
corPos++
corNum++
ta[i]=-1
tg[i]=-1
```

(A)

(A)

```
for i=0 through
row
```

```
for j=0 through
row
```

ta[i]==tg[j] s.t
i!=j and both
are
unchecked —true→
```
corNum++
ta[i]=-1
tg[j]=-1
```

```
delete []ta, []tg
```

(end)
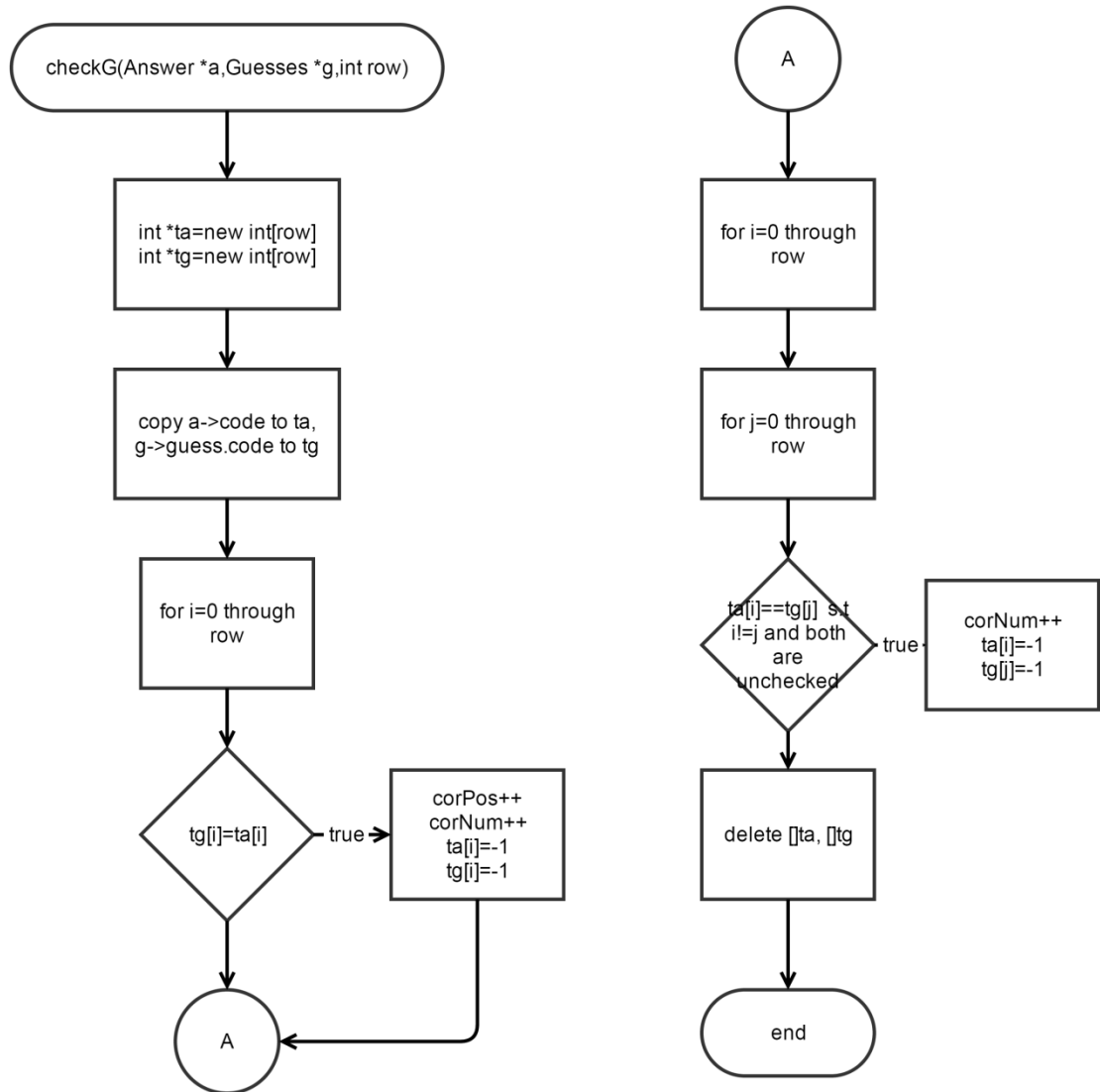
Battleship Target member functions

```
DerivBS target()
  │
  ▼
enter row component  ◄───── error
  │
  ▼
row component is within dimension of board ──── error
  │ true
  ▼
enter column component  ◄───── error
  │ true
  ▼
column component is within dimension of board ──── error
  │
  ▼
board[row][column] isn't blank and isn't ship ──── true ──► spot has been fired on ──► (back to DerivBS target())
  │
  ▼
board[row][column] is ship ──── true ──► set to X for hit
  │                                          │
  ▼                                          ▼
set to X for hit ──────────────────────► end target()
```

```
BaseBS target()
  │
  ▼
random row random column  ◄───── true
  │
  ▼
board[row][column] is not blank ──── true
  │
  ▼
board[row][column] is ship ──── true ──► set to X for hit
  │                                          │
  ▼                                          ▼
set to X for hit ──────────────────────► end target()
```

3.3　　Structures, Classes and Variables

3.3.1　Structure and Classes UML:
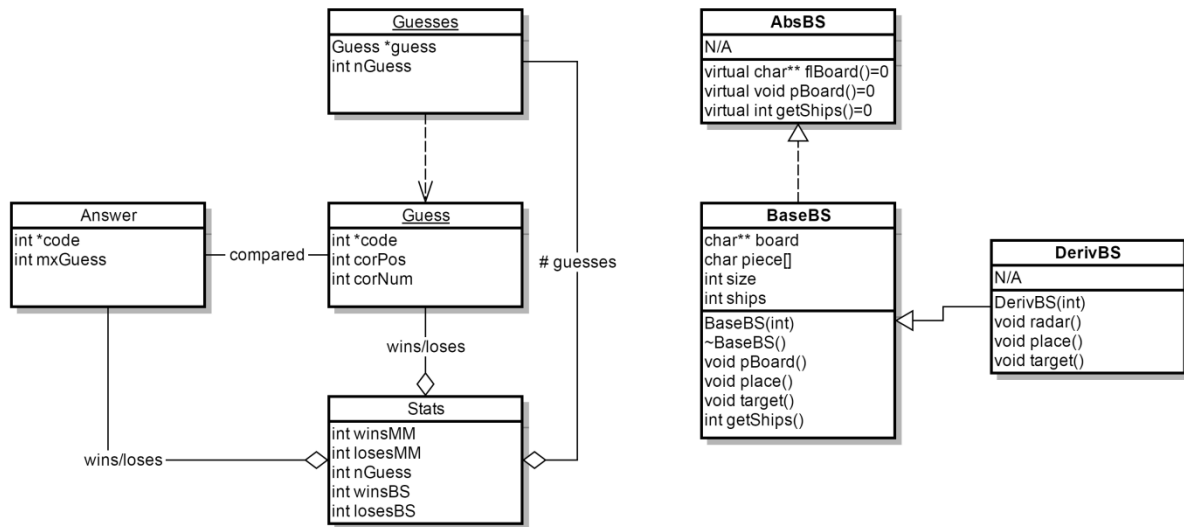
3.3.2　Structures/Classes/Variable Listing:

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

4 References

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

5 Code Documentation

Main

```cpp
6  /*
7   * File:   main.cpp
8   * Author: Joseph Levin
9   * C++ Project 2 - Spring 2015 43950
10  * 6/5/2015
11  */
12
13 //System Libraries
14 #include <cstdlib>
15 #include <iostream>
16 #include <iomanip>
17 #include <ctime>
18 #include <string>
19 #include <cstring>
20 #include <fstream>
21
22 using namespace std;
23
24 //User Libraries
25 #include "mastermind.h"
26 #include "DerivBS.h"
27 #include "stats.h"
28
29 //Global Constants
30
31 //Function Prototypes
32 Answer *getAns(int,int);//generates an answer
33 bool again();//replay function
34 bool menu();//returns to menu
35 int *getG(int);//gets guess from user
36 int getL();//returns length for code combination
37 short slct();
38 Stats *load();//loads stats structure
39 void play(Stats *,int,int);//launches the game
40 void pAns(Answer *,Guesses *,int);//prints the answer
41 void save(Stats *);//saves the stats structure as binary file
42 void pBoard(Guesses *,Answer *,int);//prints the game board
43 void checkG(Answer *,Guess *,int);//checks guess against answer
44 void instrct();
45 void purge(Answer *, Guesses *,int);//deletes all structures related to
   a game
46 void seeStats(Stats *);//Displays stats
47 void clrscrn();//clears screen
48 void playMM(Stats *);//driver for mastermind game
49 void playBS(Stats *);//driver for battleship
50 int getDim();//gets dimension for BattleShip board
51
52 //Begin
53 int main(int argc, char** argv) {
54     srand(time(0));
55     short optn;
```

```cpp
56      bool optnChk;//for checking if menu option is valid
57      Stats *s=new Stats;
58      //initialize all Stats variables to 0
59      s->gamesMM=0;
60      s->winsMM=0;
61      s->losesMM=0;
62      s->gamesBS=0;
63      s->winsBS=0;
64      s->losesBS=0;
65      s->nGuess=0;
66      //display menu
67      do{
68          optnChk=false;
69          clrscrn();
70          //get menu selection
71          optn=slct();
72          switch(optn){
73              //display instructions
74              case 1:
75                  do{
76                      clrscrn();
77                      instrct();
78                  }while(!menu());
79                  break;
80              //load a stats struct
81              case 2:
82                  s=load();
83                  break;
84              //view stats
85              case 3:
86                  do{
87                      clrscrn();
88                      seeStats(s);
89                  }while(!menu());
90                  break;
91              //play Mastermind
92              case 4:
93                  do{
94                      clrscrn();
95                      playMM(s);
96                  }while(again());
97                  break;
98              //play BattleShip
99              case 5:
100                     do{
101                         clrscrn();
102                         playBS(s);
103                     }while(again());
104                     break;
105                 //exit case
106                 case -1:
107                     cout<<"Thanks for playing!"<<endl;
108                     break;
109                 //invalid option
110                 default:
111                     cout<<"Invalid option selected..."
112                             "it's not that hard, really."<<endl;
```

```
113                        break;
114                    }
115              }while(optn!=-1);
116              //clean up!
117              delete s;
118
119              //It's over!
120              return 0;
121          }
122
123          //!Clear screen function outputs a ton of new lines in order to
    clear
124          //!the command prompt to look nice
125          void clrscrn(){
126              for(int i=0; i<100; i++)
127                  cout<<endl;
128          }//end
129          //!getAns dynamically creates a Answer struct, fills the code
    array with
130          //!random integers 0-9, sets nGuess to max (max # of guesse) and
    then returns
131          Answer *getAns(int max, int row){
132              Answer *answer=new Answer;
133              answer->mxGuess=max;//max number of guesses allowed
134              answer->code=new int[row];//the combination is 'row' digits
    long
135              for(int i=0;i<row;i++)
136                  answer->code[i]=rand()%8+1;//fill code with 1-9
137              return answer;
138          }//end
139          //!checkG takes in an Answer struct. It copies the contents of
    the code array
140          //!into another int array and compares each element to a Guess
    array
141          //!It changes the elments in the temp int array to -1 as it finds
    matches
142          //!in order to ensure no duplicate matches are found
143          //!if an element in the guess matches the position and number of
    the answer
144          //!both the correct number and correct position counters are
    incremented
145          //!otherwise if only the number is matched then the correct
    number indication
146          //!is incremented. Man this is a long description.
147          void checkG(Answer *a,Guesses *g,int row){
148              int *ta=new int[row];//temp array to store answer
149              int *tg=new int[row];//temp array to store guess
150              for(int i=0;i<row;i++){
151                  ta[i]=a->code[i];//copy answer to temp answer
152                  tg[i]=g->guess[g->nGuess-1].code[i];//copy guess to temp
    guess
153              }
154              //loop through arrays to check for both correct position and
    correct number
155              for(int i=0;i<row;i++){
156                  if(tg[i]==ta[i]){
157                      g->guess[g->nGuess-1].corPos++;
```

```cpp
158                    g->guess[g->nGuess-1].corNum++;
159                    ta[i]=-1;//no duplicates
160                    tg[i]=-1;
161                }
162            }
163            //loop through temp answer again
164            for(int i=0;i<row;i++){
165                //loop through guess
166                for(int j=0;j<row;j++){
167                    //check for same number
168                    if(ta[i]==tg[j]&&i!=j&&tg[j]!=-1&&ta[i]!=-1){//same
    num, diff pos
169                        g->guess[g->nGuess-1].corNum++;
170                        ta[i]=-1;
171                        tg[j]=-1;
172                    }
173                }
174            }
175            delete []ta;
176            delete []tg;
177        }//end
178        /*!
179         * getG prompts the user to enter a 4 digit combination,
    separated by spaces
180         * it stores it as a string, and checks if every odd element is a
    digit 1-8
181         * and then checks if every even is a space
182         * if this is true, it then converts each odd element into an int
    and stores
183         * it in an int array and returns this array
184         * takes an int to specify how big an int array to create
185         */
186        int *getG(int row){
187            //declare variables
188            int *temp=new int[row];//will be returned
189            string guess;//for character checking
190            bool check;//input validation
191            cout<<"Enter a "<<row
192                    <<" digit combination (1-8) separated by
    spaces."<<endl;
193            do{
194                check=true;
195                getline(cin,guess);
196                if(guess.size()!=2*row-1){//row digits plus spaces
    between
197                    cout<<"You must enter "<<row<<" digits separated by
    spaces."
198                            <<"Try again!"<<endl;
199                    check=false;;
200                }
201                else{
202                    //every odd must be 1-8, and every even must be space
203                    for(int i=0;i<2*row-1;i++){
204                        //check odds for all digits
205                        if(i%2==0&&!isdigit(guess[i])){
206                            check=false;
```

```cpp
207                        cout<<"Please only enter digits. Try
      again!"<<endl;
208                    }
209                    //check if all digits are 1-8
210                    else
      if(i%2==0&&(atoi(&guess[i])>8||atoi(&guess[i])<1)){
211                        check=false;
212                        cout<<"Digits must all be between 1 & 8. Try
      again!"<<endl;
213                        break;
214                    }
215                    //check if every even is space
216                    else{
217                        if(!isspace(guess[i])&&!isdigit(guess[i])){
218                            check=false;
219                            cout<<"Every digit must be "
220                                <<"separated by a space. Try
      again!"<<endl;
221                        }
222                    }
223                }
224            }
225        }while(!check);
226        //copy digits into int array to return
227        for(int i=0;i<row;i++)
228            temp[i]=atoi(&guess[2*i]);
229        //int array ready to return
230        return temp;
231    }//end
232    /*!
233     * pBoard takes in pointers to an Answer and Guess, and an int
234     * specifying the code length. It prints out dashed lines
      indicating
235     * how many guesses remain and also prints out all previous
      guesses
236     */
237    void pBoard(Guesses *g, Answer *a, int r){
238        //X's represent the mystery code
239        for(int i=0;i<r;i++)
240            cout<<"X ";
241        cout<<endl;
242        //-'s represent spaces left for remaining guesses
243        for(int i=0;i<(a->mxGuess)-(g->nGuess)+1;i++){
244            for(int j=0;j<r;j++)
245                cout<<"- ";
246            cout<<endl;
247        }
248        //all previous guesses
249        if(g->nGuess-1!=0){
250        for(int i=g->nGuess-2;i>=0;i--){
251            for(int j=0;j<r;j++)
252                cout<<g->guess[i].code[j]<<" ";
253            cout<<" N:"<<g->guess[i].corNum
254            <<" P:"<<g->guess[i].corPos<<endl;
255        }
256        }
257    //finished
```

```cpp
258        }//end
259        /*!
260         * getL prompts the user to enter either 4, 6, or 8. It then
   returns
261         * the value selected as an int. No parameters
262         */
263        int getL(){
264            int l;//length of code
265            bool check=false;//error checking flag
266            cout <<"Please the code length!"<<endl<<"Options: 4, 6,
   8"<<endl;
267            do{
268                cin>>l;
269                if(cin.fail()||(l!=4&&l!=6&&l!=8)){
270                    cin.clear();
271                    cin.ignore(256,'\n');
272                    cout<<"Sorry! You have to choose "
273                            "either 4, 6, or 8. Try again!"<<endl;
274                }
275                else
276                    check=true;
277            }while(!check);
278            return l;
279        }//end
280        /*!
281         * pAns prints the answer code
282         */
283        void pAns(Answer *a,Guesses *g,int r){
284            //Show Answer
285            for(int i=0;i<r;i++)
286                cout<<a->code[i]<<" ";
287
288            cout<<" <- Answer!"<<endl;
289            //-'s represent spaces left for remaining guesses
290            for(int i=0;i<(a->mxGuess)-(g->nGuess);i++){
291                for(int j=0;j<r;j++)
292                    cout<<"- ";
293                cout<<endl;
294            }
295            //all previous guesses
296            if(g->nGuess!=0){
297            for(int i=g->nGuess-1;i>=0;i--){
298                for(int j=0;j<r;j++)
299                    cout<<g->guess[i].code[j]<<" ";
300                cout<<" N:"<<g->guess[i].corNum
301                <<" P:"<<g->guess[i].corPos<<endl;
302            }
303            }
304        }//end
305        /*!
306         * play is the main driver for Mastermind gameplay. It handles
   turn taking,
307         * win/lose checks, as well as stat saving. Void function, takes
   in an integer
308         * to determine max # of guesses and an integer to determine code
   length
309         */
```

```cpp
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
```

```
362          * allocated elements of the struct, as well as the structs
     themselves
363          * and then points them to nullptrs
364          */
365         void purge(Answer *a,Guesses *g,int m){
366              delete []a->code;
367              a->code=NULL;
368              delete a;
369              for(int i=0;i<m;i++){
370                   delete []g->guess[i].code;
371                   g->guess[i].code=NULL;
372              }
373              delete g->guess;
374              g->guess=NULL;
375              delete g;
376              g=NULL;
377         }
378         /*!
379          * save takes in a Stats struct pointer. It prompts the user for
     a name
380          * to store the stats struct under, and then writes the contents
     of the Stats
381          * to a binary file. Returns void
382          */
383         void save(Stats *s){
384              cin.clear();
385              cin.ignore(256,'\n');
386              ofstream out;//file stream
387              cout<<"Enter the name to store the stats under"<<endl;
388              string name;
389              getline(cin,name);
390              out.open(name.c_str(),ios::binary);
391              out.write(reinterpret_cast<char *>(s),sizeof(Stats));
392              out.close();
393         }//end
394         /*!
395          * load prompts the user for a name, and attempts to open a file
396          * with that name. If found, it reads the contents into a Stats
     structure
397          * and returns it.
398          */
399         Stats *load(){
400              cin.clear();
401              cin.ignore(256,'\n');
402              string name;
403              ifstream in;
404              cout<<"Enter the name of the person whose stats to
     load"<<endl;
405              do{
406                   getline(cin,name);
407                   in.open(name.c_str(),ios::binary);
408                   if(in.fail()){
409                        cout<<"Name not found. Try again!"<<endl;
410                   }
411                   else{
412                        Stats *s=new Stats;
413                        in.read(reinterpret_cast<char *>(s),sizeof(Stats));
```

```cpp
414                    in.close();
415                    return s;
416                }
417            }while(in.fail());
418        }
419        //!slct serves to take in input for menu selection, performs
   error checks
420        //!and then returns the value if it passes checks
421        short slct(){
422            short pick; //for menu selection
423            bool check=false;
424            cout<<"Welcome to Joseph Levin's Project 2!"<<endl;
425            cout<<"Choose an option from the menu: "<<endl
426                    <<"1. View Instructions"<<endl
427                    <<"2. Load Stats File"<<endl
428                    <<"3. View Stats"<<endl
429                    <<"4. Play Mastermind!"<<endl
430                    <<"5. Play BattleShip!"<<endl
431                    <<"-1 to quit"<<endl;
432            do{
433                cin>>pick;
434                if(cin.fail()||pick<=0&&pick!=-1||pick>5){//error
   checking
435                    cin.clear();
436                    cin.ignore(256,'\n');
437                    cout<<"Error. Invalid selection. Try again."<<endl;
438                }
439                else
440                    check=true;//valid input
441            }while(!check);
442            return pick;
443        }
444        /*!
445         * again asks the user if they would like to play another game
446         * it returns true if the user does, false if they do not
447         */
448        bool again(){
449            bool check=false;
450            char pick;
451            cout<<"Would you like to play again? y/n"<<endl;
452            do{
453                cin>>pick;
454
   if(cin.fail()||tolower(pick)!='y'&&tolower(pick)!='n'){//only accepts
455                    cin.clear();
   //y or n as input
456                    cin.ignore(256,'\n');
457                    cout<<"Error. Invalid selection. Try again."<<endl;
458                }
459                else if(tolower(pick)=='y'){//user wants to repeat
460                    check=true;
461                    cin.clear();
462                    cin.ignore(256,'\n');
463                    return true;
464                }
465                else{ //user does not want to repeat
466                    cin.clear();
```

```cpp
467                    cin.ignore(256,'\n');
468                    check=true;
469                return false;
470                }
471            }while(!check);
472
473        }//end
474        /*!
475         * menu prompts the user if they want to return to the main menu.
476         * if the user types 'y', it returns true
477         * if the user types 'n', it returns false
478         */
479        bool menu(){
480            bool check=false;
481            char pick;
482            cout<<"Would you like to return to the menu? y/n"<<endl;
483            do{
484                cin>>pick;
485
    if(cin.fail()||tolower(pick)!='y'&&tolower(pick)!='n'){//only accepts
   //y or n as input
486                    cin.clear();
487                    cin.ignore(256,'\n');
488                    cout<<"Error. Invalid selection. Try again."<<endl;
489                }
490                else if(tolower(pick)=='y'){//user wants to repeat
491                    check=true;
492                    cin.clear();
493                    cin.ignore(256,'\n');
494                    return true;
495                }
496                else{ //user does not want to repeat
497                    cin.clear();
498                    cin.ignore(256,'\n');
499                    check=true;
500                return false;
501                }
502            }while(!check);
503        }
504        /*!
505         * seeStats takes in Stats pointer. It displays the elements
   within,
506         * (wins/loses/total guesses) and also calculates win percentage,
   and
507         * correct guess percentage
508         */
509        void seeStats(Stats *s){
510            if(s->gamesBS!=0||s->gamesMM!=0){
511                cout<<"Mastermind"<<endl;
512                cout<<"Games played: "<<s->gamesMM<<endl;
513                cout<<"Wins: "<<s->winsMM<<endl;
514                cout<<"Losses: "<<s->losesMM<<endl;
515                cout<<"Total guesses: "<<s->nGuess<<endl;
516                cout<<endl;
517                cout<<"BattleShip"<<endl;
518                cout<<"Games played: "<<s->gamesBS<<endl;
519                cout<<"Wins: "<<s->winsBS<<endl;
```

```cpp
520                cout<<"Losses: "<<s->losesBS<<endl;
521            }
522            else
523                cout<<"Stats File is empty!"<<endl;
524        }
525        /*!
526         * instrct displays the rules for Code Breaker (based on
   Mastermind)
527         */
528        void instrct(){
529            cout<<"     **************"<<endl;
530            cout<<"     *            *"<<endl;
531            cout<<"     * MASTERMIND *"<<endl;
532            cout<<"     *            *"<<endl;
533            cout<<"     **************"<<endl;
534            cout<<"Mastermind is a game of decryption!"<<endl
535                    <<"In it, the player (that's you!)"<<endl
536                    <<"attempts to decipher a secret code"<<endl
537                    <<"generated at random. The code consists"<<endl
538                    <<"of integers between 1 and 8."<<endl
539                    <<"The player chooses from 3 code lengths (4, 6,
   8),"<<endl
540                    <<"and guesses at each of the individual
   digits"<<endl
541                    <<"that make up the secret code. After each"<<endl
542                    <<"attempt the player is told how many correct
   "<<endl
543                    <<"numbers (shown as 'N') and correct position
   "<<endl
544                    <<"(shown as 'P') the attempt had. The player"<<endl
545                    <<"wins by correctly determining all 4 correct"<<endl
546                    <<"numbers and positions. Remember, a guess
   can"<<endl
547                    <<"have 4 correct numbers but 0 correct
   positions,"<<endl
548                    <<"but you can't have a correct position
   without"<<endl
549                    <<"it also being the correct number."<<endl;
550            cout<<endl;
551            cout<<"     **************"<<endl;
552            cout<<"     *            *"<<endl;
553            cout<<"     * BATTLESHIP *"<<endl;
554            cout<<"     *            *"<<endl;
555            cout<<"     **************"<<endl;
556            cout<<"Battleship is a guessing game!"<<endl
557                    <<"In it, the player (that's you again!)"<<endl
558                    <<"attempts to find and destroy the enemy's
   ships,"<<endl
559                    <<"which are placed randomly. However, the
   enemy"<<endl
560                    <<"also fires on the player! The player starts"<<endl
561                    <<"by choosing the starting location of his
   ships"<<endl
562                    <<"and then takes turns guessing at the
   location"<<endl
563                    <<"of the enemy's ships by entering the row
   and"<<endl
```

```cpp
564                     <<"column coordinates of the spot on the board"<<endl
565                     <<"at which to 'fire' upon. Hits are
   designated"<<endl
566                     <<"with an 'X', and misses are 'O'. The game"<<endl
567                     <<"until either the player or the computer
   runs"<<endl
568                     <<"out of ships."<<endl;
569             cout<<endl<<"Good luck!"<<endl;
570         }//end
571         /*!
572          * playMMis the main driver for Mastermind.
573          * Parameters: a stats struct for storing stats
574          */
575         void playMM(Stats *s){
576             clrscrn();
577             int l=getL();
578             play(s,10,l);
579         }
580         /*!
581          * playBS is the main driver for BattleShip.
582          * Parameters: a stats struct for storing stats
583          */
584         void playBS(Stats *s){
585             int dim;//for storing dimension of board
586             char optn;//for stat saving selection
587             bool sConf=false;//confirming size is valid
588             cout<<"Welcome to BattleShip!"
589                     " Please select the size board you'd like to
   play."<<endl;
590             cout<<"1. 6x6"<<endl;
591             cout<<"2. 8x8"<<endl;
592             cout<<"3. 10x10"<<endl;
593             do{
594                 try{
595                     dim=getDim();
596                     sConf=true;
597                 }
598                 catch(string invalid){
599                     cout<<invalid<<endl;
600                 }
601             }while(!sConf);
602             //initialize BaseBS (for player) and DerivBS (for computer)
   of dim
603             BaseBS player(dim);
604             DerivBS comp(dim);
605             //randomly place computer's ships
606             comp.place();
607             //place players ships
608             player.place();
609             //begin rounds
610             do{
611                 clrscrn();
612                 //display board and radar
613                 cout<<"Player board ("<<player.getShips()<<" ships
   remain)"<<endl;
614                 player.pBoard();
615                 cout<<endl;
```

```cpp
616                 cout<<"Radar ("<<comp.getShips()<<" enemy ships
    remain)"<<endl;
617                 comp.radar();
618                 //targeting round
619                 player.target();
620                 comp.target();
621             }while(comp.getShips()!=0&&player.getShips()!=0);
622             clrscrn();
623             //Display boards final time
624             cout<<"Player board ("<<player.getShips()<<" ships
    remain)"<<endl;
625             player.pBoard();
626             cout<<endl;
627             cout<<"Radar ("<<comp.getShips()<<" enemy ships
    remain)"<<endl;
628             comp.radar();
629             //Determine victor and increment stats counters
630             s->gamesBS++;
631             if(comp.getShips()==0){
632                 cout<<"You win! Congratulations!"<<endl;
633                 s->winsBS++;
634             }
635             else{
636                 cout<<"You lose. Sorry!"<<endl;
637                 s->losesBS++;
638             }
639             cout<<"Would you like to save your stats? y/n"<<endl;
640             do{
641                 cin>>optn;
642                 if(tolower(optn)!='y'&&tolower(optn)!='n')
643                     cout<<"Sorry, that's not a valid option. Please try
    again."<<endl;
644                 else if(tolower(optn)=='y'){
645                     save(s);
646                     cout<<"Stats saved!"<<endl;
647                 }
648                 else
649                     cout<<"Stats not saved."<<endl;
650             }while(tolower(optn)!='y'&&tolower(optn)!='n');
651         }//end
652         /*!
653          * getDim prompts user for dimension for BattleShip board. It
    will throw
654          * an exception if invalid, otherwise returns the value
655          */
656         int getDim(){
657             int dim;
658             cin>>dim;
659             if(dim<1||dim>3||cin.fail()){
660                 string invalid="Invalid selection for dimension. Try
    again.";
661                 throw invalid;
662             }
663             //Reassign dim to the corresponding dimension
664             if(dim==1){
665                 dim=6;
666             }
```

```
667            else if(dim==2){
668                dim=8;
669            }
670            else{
671                dim=10;
672            }
673            return dim;
674 }
```

Mastermind.h

```
675      /*
676       * File:   mastermind.h
677       * Author: Joseph
678       *
679       * Created on April 26, 2015, 7:47 PM
680       */
681      #ifndef MASTERMIND_H
682      #define MASTERMIND_H
683
684      /*!
685       * Answer stores the correct code in an int array and the max
    guesses allowed
686       */
687      struct Answer{
688          //!stores the integer combination
689          int *code;
690          //!keeps track of maximum guesses allowed
691          int mxGuess;
692      };
693
694      /*!
695       * Guess stores a guess in an int array, the number of correct
    positions
696       * associated with the guess and the number of correct numbers
697       */
698      struct Guess{
699          //!stores the guess
700          int *code;
701          //!tallies the number of correct positions in a guess
702          int corPos;
703          //!tallies the number of correct numbers in a guess
704          int corNum;
705      };
706
707      /*!
708       * Guesses stores a pointer a Guess (one for each turn) and
709       * the total number of guesses taken
710       */
711      struct Guesses{
712          //!array of guess attempts
713          Guess *guess;
714          //!how many guesses have been taken
715          int nGuess;
716      };
717 #endif  /* MASTERMIND_H *
```

Stats.h

```
718     /*
719      * File:    stats.h
720      * Author: Joseph Levin
721      * C++ Project 2 - Spring 2015 43950
722      * 6/5/2015
723      */
724
725     #ifndef STATS_H
726     #define STATS_H
727     /*!
728      * Stats stores the number of wins, losses, and number of guesses
729      */
730     struct Stats{
731         //!number of wins for Mastermind
732         int winsMM;
733         //!number of loses for Mastermind
734         int losesMM;
735         //!number of guesses for Mastermind
736         int nGuess;
737         //!number of wins for BattleShip
738         int winsBS;
739         //!number of loses for BattleShip
740         int losesBS;
741         //!tracks games played
742         int gamesBS;
743         int gamesMM;
744     };
745 #endif  /* STATS_H */
```

AbsBS.h

```
746
747     /*
748      * File:    AbsBS
749      * Author: Joseph Levin
750      * C++ Project 2 - Spring 2015 43950
751      * 6/5/2015
752      */
753
754     #ifndef ABSBS_H
755     #define ABSBS_H
756     //!Abstract class for the Battleship board
757     class AbsBS{
758         public:
759             //!2D char array to store the board
760             virtual char** flBoard()=0;
761             //!outputs the board
762             virtual void pBoard()=0;
763             //!returns ships remaining
764             virtual int getShips()=0;
765     };
766     #endif  /*  ABSBS_H */
```

BaseBS.h

```cpp
767      /*
768       * File:   BaseBS.h
769       * Author: Joseph Levin
770       * C++ Project 2 - Spring 2015 43950
771       * 6/5/2015
772       */
773
774      #ifndef BASEBS_H
775      #define BASEBS_H
776      #include "AbsBS.h"
777
778      /*!
779       * BaseBS is the base class for the battleship board.
780       */
781      class BaseBS:AbsBS{
782          protected:
783              //!for storing the board
784              char** board;
785              //!X is hit, O is miss, + is ship, <space> is empty
786              char piece[4]={'X','O','+', ' '};
787              //!dimension of board
788              int size;
789              //!ships on board
790              int ships;
791          public:
792              //!Constructor for base battleship board. Takes in an
   integer for
793              //!the dimension of the board (area is nxn))
794              BaseBS(int);
795              //!Destructor for the board
796              ~BaseBS();
797              //!For initially filling board
798              char** flBoard();
799              //!For outputting the board formatted to show ships
800              void pBoard();
801              //!place handles the ship placing procedure for the
   player.
802              //!it checks to make sure the given coordinates are
   within
803              //!the acceptable range of size, and then sets the given
   tile
804              //!to the + char to represent a ship has been placed
805              void place();
806              //!target handles the process of targeting a spot to fire
   on it
807              void target();
808              //!getShips returns the value of the ships member
   variable
809              int getShips();
810      };
811 #endif  /* BASEBS_H */
```

BaseBS.cpp

```cpp
812     /*
813      * File:   BaseBS.cpp
814      * Author: Joseph Levin
815      * C++ Project 2 - Spring 2015 43950
816      * 6/5/2015
817      */
818
819     //Libraries
820     #include <iostream>
821     #include <iomanip>//board formatting
822     #include <cstdlib>
823     #include <vector>//for rand
824     #include "BaseBS.h"
825
826     using namespace std;
827
828     BaseBS::BaseBS(int s){
829         size=s;
830         board=flBoard();
831     }
832     BaseBS::~BaseBS(){
833         for(int i=0;i<size;i++){
834             delete []board[i];
835         }
836         delete []board;
837     }
838     /*!
839      * flBoard() initializes the board with all empty spaces
840      */
841     char** BaseBS::flBoard(){
842         char** board = new char*[size];
843         for (int i=0;i<size; i++){
844             board[i]=new char[size];
845         }
846         for (int i=0;i<size;i++) {
847             for (int j=0;j<size;j++)
848                 board[i][j]=piece[3];
849         }
850
851         return board;
852     }
853     void BaseBS::pBoard(){
854         cout<<"   ";
855         for (int i=0;i<size;i++) {
856             cout<<setw(3)<<setfill(' ')<<i<<" ";
857         }
858         cout<<endl;
859         for (int i=0;i<size; i++) {
860             cout<<"  "<<setw(size*4+1)<<setfill('-')<<'-'<<endl;
861             cout<<i<<" ";
862             for (int j=0;j<size;j++) {
863                 cout<<"|"<<setw(2)<<setfill(' ')<<board[i][j]<<" ";
864             }
865             cout<<"| "<<endl;
866         }
```

```cpp
867              cout<<"  "<<setw(size*4+1)<<setfill('-') << '-' << endl;
868         }
869        void BaseBS::place(){
870             bool conf1,conf2,conf3;//error buffers
871             ships=0;//begin with no ships placed
872             int row, col;//for checking the given coordinate
873             for(int i=0;i<size;i++){//will place size # of ships
874                 do{
875                     pBoard();
876                     cout<<"Enter the row coordinate for where to begin
   ship "
877                             <<ships+1<<endl;
878                     //check confirms to false
879                     conf1=false;
880                     conf2=false;
881                     conf3=false;
882                     do{//gets the row coordinate
883                         cin>>row;
884                         if (cin.fail()||(row<0||row>size-1)){//size-1
   denotes edge
885                             cin.clear();
886                             cin.ignore(256,'\n');
887                             cout<<"Error. Invalid input."<<endl;
888                         } else
889                             conf1=true;//row coordinate is acceptable
890                     } while(cin.fail()||row<0||row>size-1||!conf1);
891                     cout<<"Enter the column coordinate for where to begin
   ship "
892                             <<ships+1<<endl;
893                     do{
894                         cin>>col;
895                         if (cin.fail()||(col<0||col>size-1)){
896                             cin.clear();
897                             cin.ignore(256,'\n');
898                             cout<<"Error. Invalid input."<<endl;
899                         } else
900                             conf2=true;//column coordinate is acceptable
901                     } while(cin.fail()||col<0||row>size-1||!conf1);
902                     if(board[row][col]!=piece[3]){//piece[3] == blank
   space
903                         cout << "This spot is already occupied" << endl;
904                     }
905                     else {
906                         board[row][col]=piece[2];//piece[2]== '+'
907                         ships++;//added ship to board
908                         conf3=true;//process is completed successfully
909                     }
910                 }while(!conf3);
911             }
912         }
913        //!target for BaseBS handles the computer firing on the player's
   board
914        //!it is a glorified random number generator
915        void BaseBS::target(){
916             bool confirm=false;
917             int row,col;
918             //randomly fires at spots until it
```

```cpp
919             do{
920                 row=(rand()%size);
921                 col=(rand()%size);
922                 //piece[3]==blank, piece[2]==ship
923                 if(board[row][col]==piece[3]||board[row][col]==piece[2])
924                     confirm=true;
925             } while(!confirm);
926             //piece[2]==ship,piece[0]==X
927             if (board[row][col]==piece[2]){
928                 board[row][col]=piece[0];
929                 ships--;
930                 //piece[1]==O
931             } else
932                 board[row][col]=piece[1];
933         }
934     //!getShips returns the value of the ships member variable
935 int BaseBS::getShips(){return ships;}
```

DerivBS.h

```cpp
936     /*
937      * File:   DerivBS.h
938      * Author: Joseph Levin
939      * C++ Project 2 - Spring 2015 43950
940      * 6/5/2015
941      */
942     #ifndef DERIVBS_H
943     #define DERIVBS_H
944
945     #include "BaseBS.h"
946
947     /*!
948      * DerivBS is a derived class of BaseBS. It contains addition
    functions
949      * for handling the computer logic
950      */
951     class DerivBS:public BaseBS{
952         public:
953             //!constructor for DerivBS calls BaseBS constructor
954             DerivBS(int);
955             //!displays a version of the board with ships masked
956             void radar();
957             //!place for DerivBS randomly places ships for computer
958             void place();
959             //!target for DerivBS randomly fires on spots for
    computer
960             void target();
961     };
962 #endif  /* DERIVBS_H */
```

DerivBS.cpp

```cpp
963     /*
964      * File:   DerivBS.cpp
965      * Author: Joseph Levin
966      * C++ Project 2 - Spring 2015 43950
967      * 6/5/2015
968      */
969
970     #include "DerivBS.h"
971     #include<iostream>
972     #include<iomanip>
973     #include <cstdlib>
974
975     using namespace std;
976
977
978     //!constructor for DerivBS calls BaseBS constructor
979     DerivBS::DerivBS(int s):BaseBS(s){;}
980     //!displays a version of the board with ships masked
981     void DerivBS::radar(){
982         cout<<"  ";
983         for (int i=0;i<size;i++) {
984             cout<<setw(3)<<setfill(' ')<<i<<" ";
985         }
986         cout<<endl;
987         for (int i=0;i<size; i++) {
988             cout<<"  "<<setw(size*4+1)<<setfill('-')<<'-'<<endl;
989             cout<<i<<" ";
990             for (int j=0;j<size;j++) {
991                 //if ship isn't at that spot, place piece at i,j
992                 if(board[i][j]!=piece[2]){
993                     cout<<"|"<<setw(2)<<setfill(' ')<<board[i][j]<<" ";
994                 }
995                 //if ship is at that spot, mask ship by placing blank
     spot instead
996                 else{
997                     cout<<"|"<<setw(2)<<setfill(' ')<<piece[3]<<" ";
998                 }
999             }
1000            cout<<"| "<<endl;
1001        }
1002        cout<<"  "<<setw(size*4+1)<<setfill('-') << '-' << endl;
1003    }
1004    //!place for DerivBS randomly places ships for computer
1005    void DerivBS::place(){
1006        int row, col;
1007        ships=0;
1008        for(int i=0;i<size;i++){
1009            //randomly picks spots until it finds one that isn't
     occupied already
1010            do{
1011                row=(rand()%size);
1012                col=(rand()%size);
1013            }while(board[row][col]!=piece[3]);//piece[3]==blank space
1014            board[row][col]=piece[2];//piece[2] == '+', ship
```

```cpp
1015                ships++;
1016            }
1017        }
1018        //!target for DerivBS handles the player firing on the computer's
    board
1019        //!it prompts for the coordinates one component a time, checks
    for any errors
1020        //!and decides if it was a hit or miss and announces such
1021        void DerivBS::target(){
1022            int row, col;
1023            bool conf1,conf2,conf3;//error buffers
1024            do{
1025                conf1=false;
1026                conf2=false;
1027                conf3=false;
1028                cout<<"Enter the row (vertical) component of the
    coordinate "
1029                        <<"you wish to fire upon"<<endl;
1030                do{
1031                    cin>>row;
1032                    if (cin.fail()||row<0||row>size-1) {
1033                        cin.clear();
1034                        cin.ignore(256,'\n');
1035                        cout<<"Error. Invalid selection." << endl;
1036                    } else
1037                        conf1=true;
1038                }while(cin.fail()||row<0||row>size-1||!conf1);
1039                cout<<"Enter the column (horizontal) component "
1040                        <<"of the coordinate you wish to fire
    upon"<<endl;
1041                do{
1042                    cin>>col;
1043                    if(cin.fail()||col<0||col>size-1){
1044                        cin.clear();
1045                        cin.ignore(256,'\n');
1046                        cout<<"Error. Invalid selection."<<endl;
1047                    }else
1048                        conf2=true;
1049                }while(cin.fail()||col<0||col>size-1||!conf2);
1050                if(board[row][col]!=piece[3]&&board[row][col]!=piece[2])
1051                    cout<<"This spot has been fired upon already."<<endl;
1052                else
1053                    conf3=true;
1054            } while (!conf3);
1055            //if a ship was hit, replace with X and decrease remaining
    ships
1056            //and announce hit was successful
1057            if (board[row][col]==piece[2]){
1058                board[row][col]=piece[0];
1059                ships--;
1060            }
1061            //if a ship wasn't hit, replace with O and announce miss
1062            else{
1063                board[row][col]=piece[1];
1064            }
1065        }
```