|  |
| --- |
|  |
| 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).

1. 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.

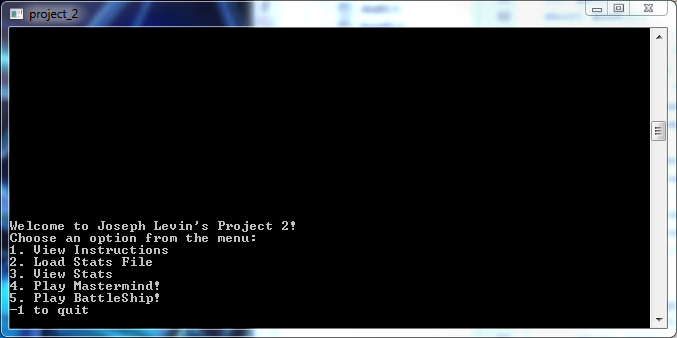
As for Battleship, I enjoyed working with classes in terms of implementation. Having less experience with objects the start was disorienting but as the member functions began to take shape it became increasingly easy to test my code as I wrote it as opposed to constantly having to pass test parameters 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.

1. Program Description

3.1 Program Walkthrough

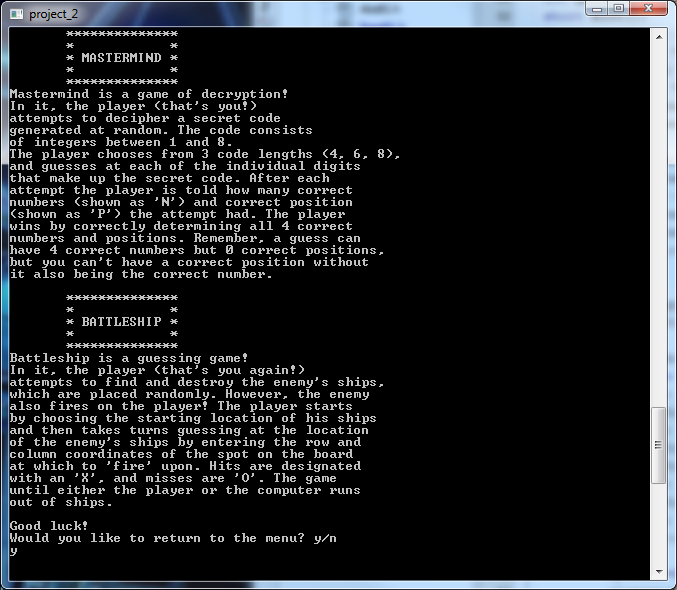


The main menu offer s5 options to the user.

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

-1 to quit

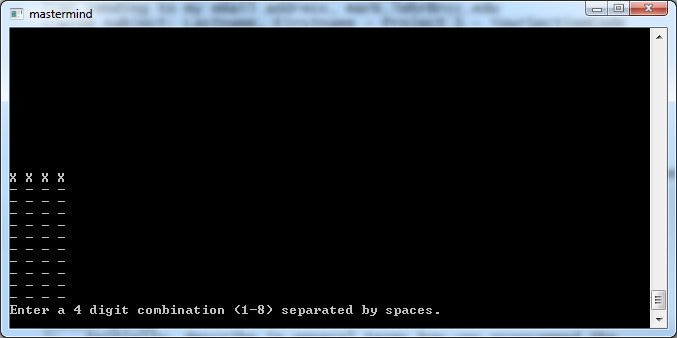
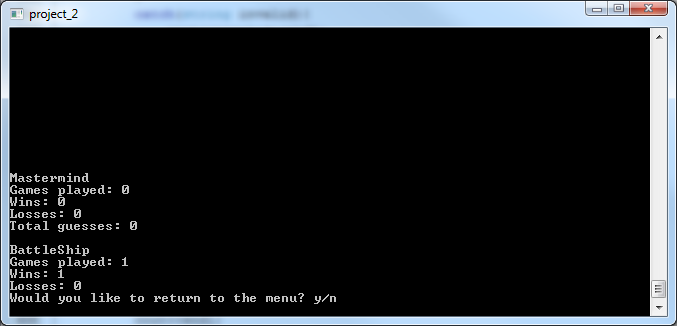
If the user inputs 1, they are shown a brief explanation of how Mastermind is played. It should be noted that the original game allows 6 different colors, but I forgot this and chose 8 instead.



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).

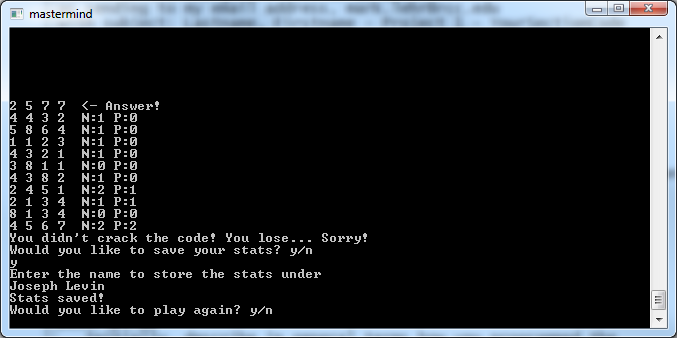
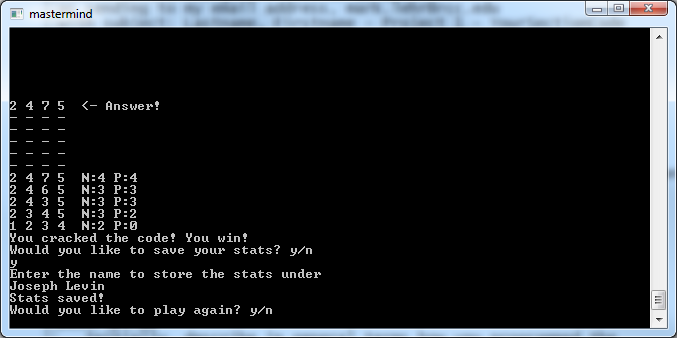
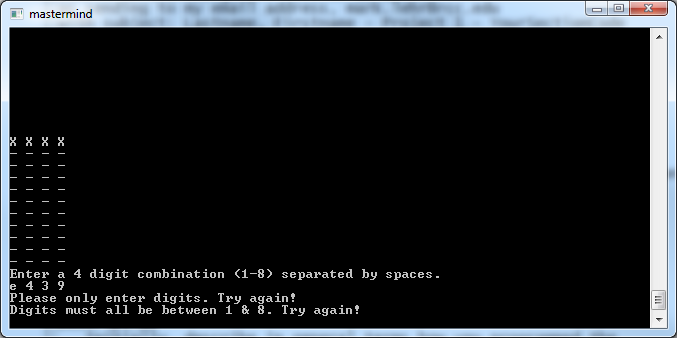
i.e, l=4, odds=1/(8^4)=0.0002

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.

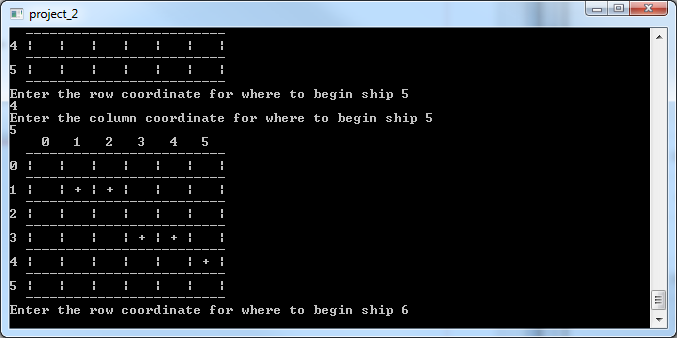
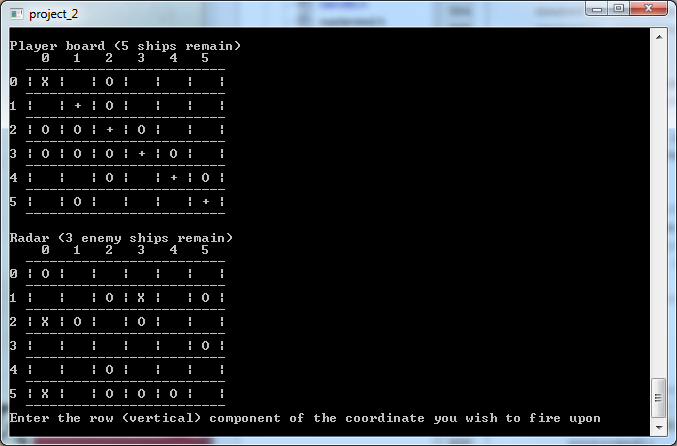
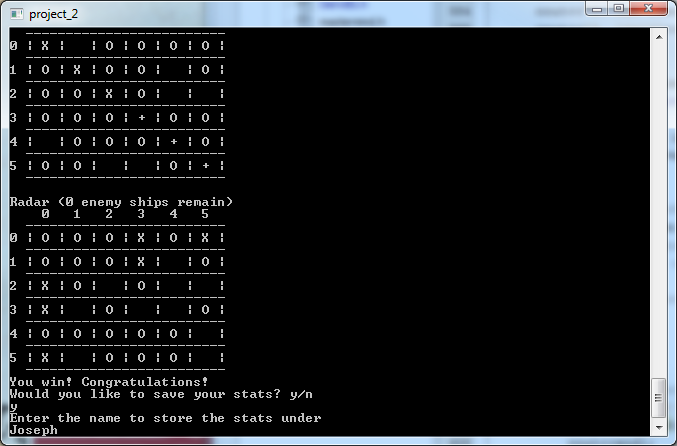
Non digits will give appropriate error, as will digits outside of range



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



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

Saving works identically to the implementation in Mastermind.

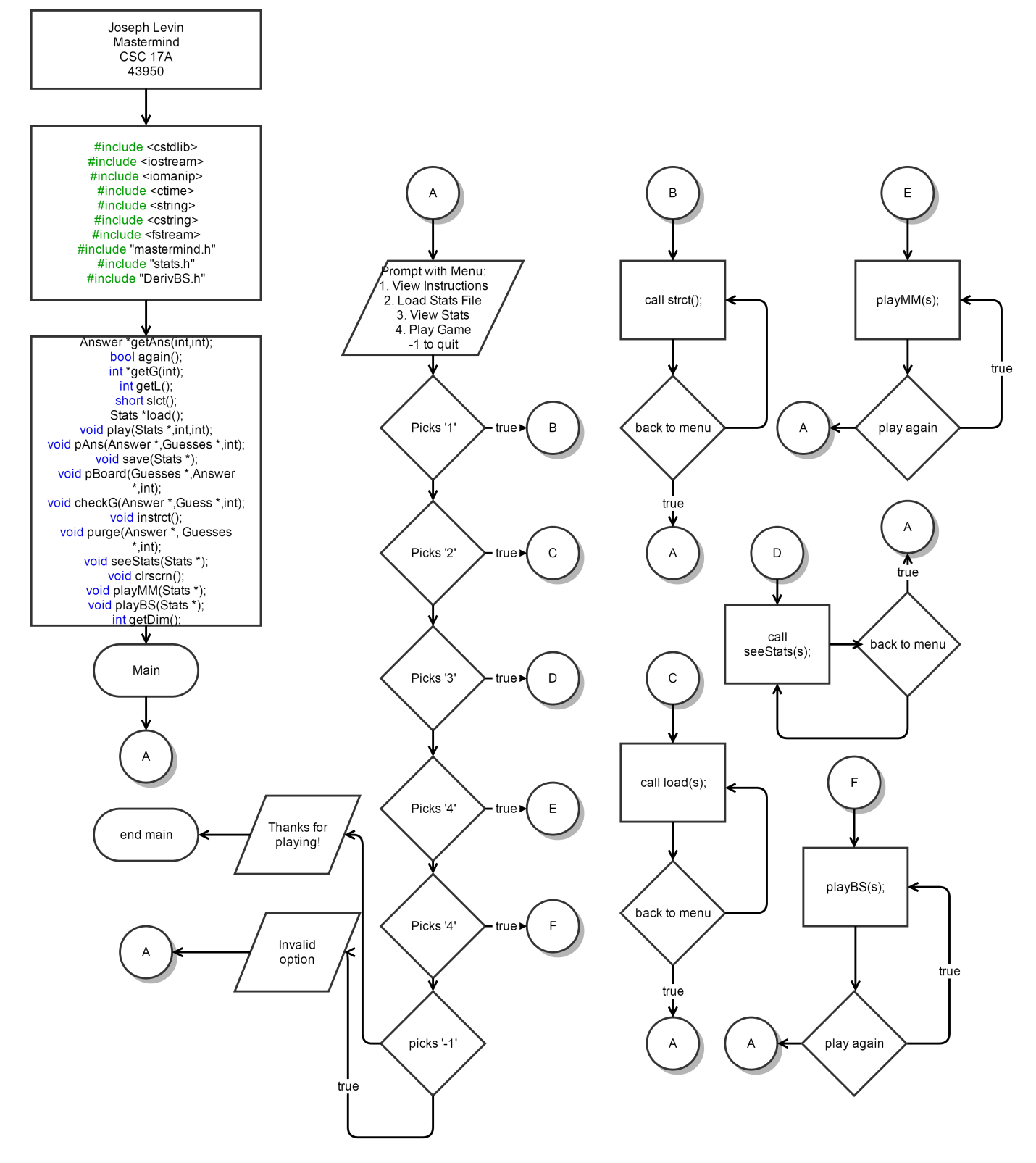
The BaseBS class contains a member function for displaying the board in a grid format. It shows the location of the ships as +, hits as X, misses as O.

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

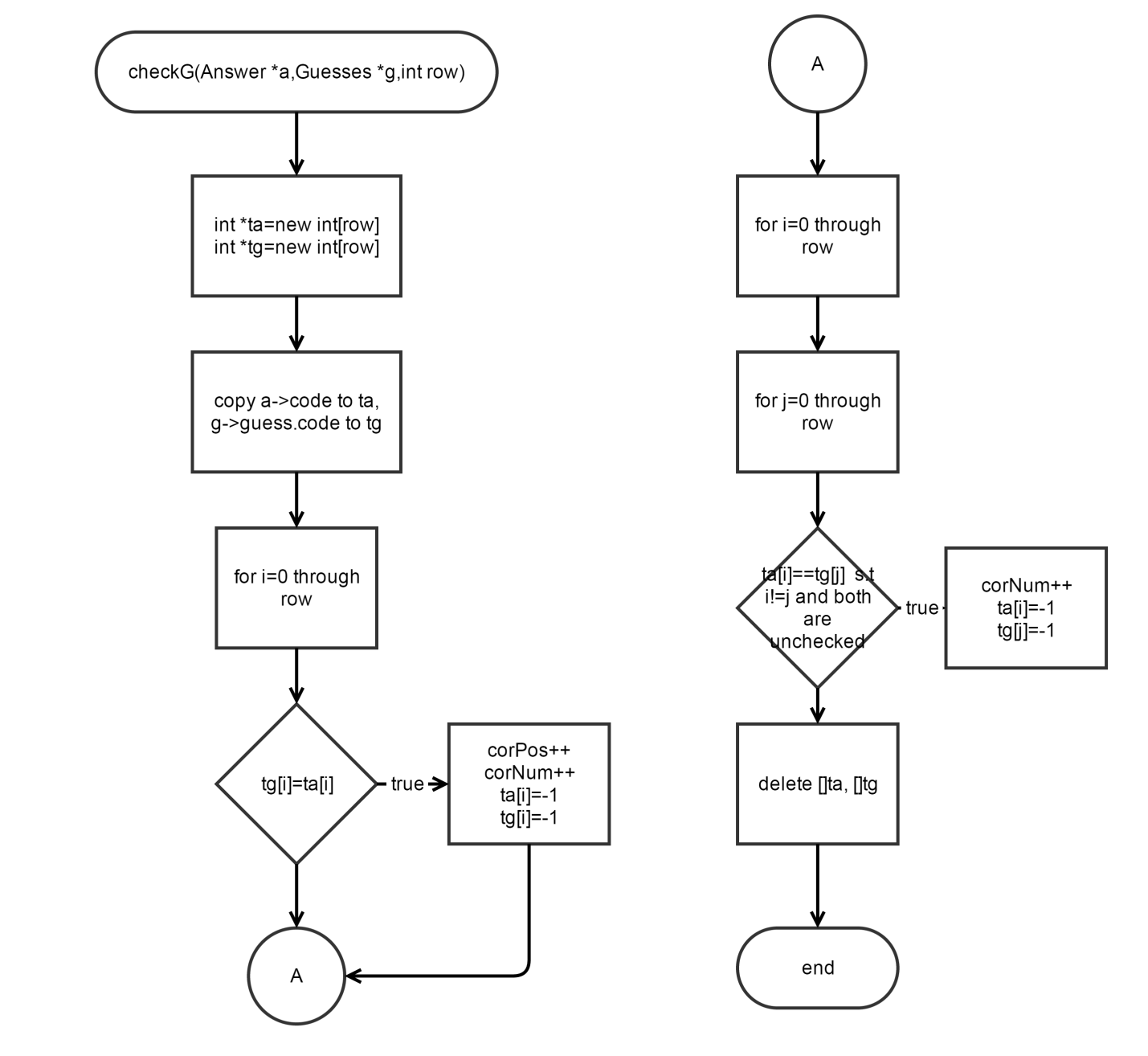
A member function is called to place the player’s ships one at time until there are ships equal to the dimension of the board.

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

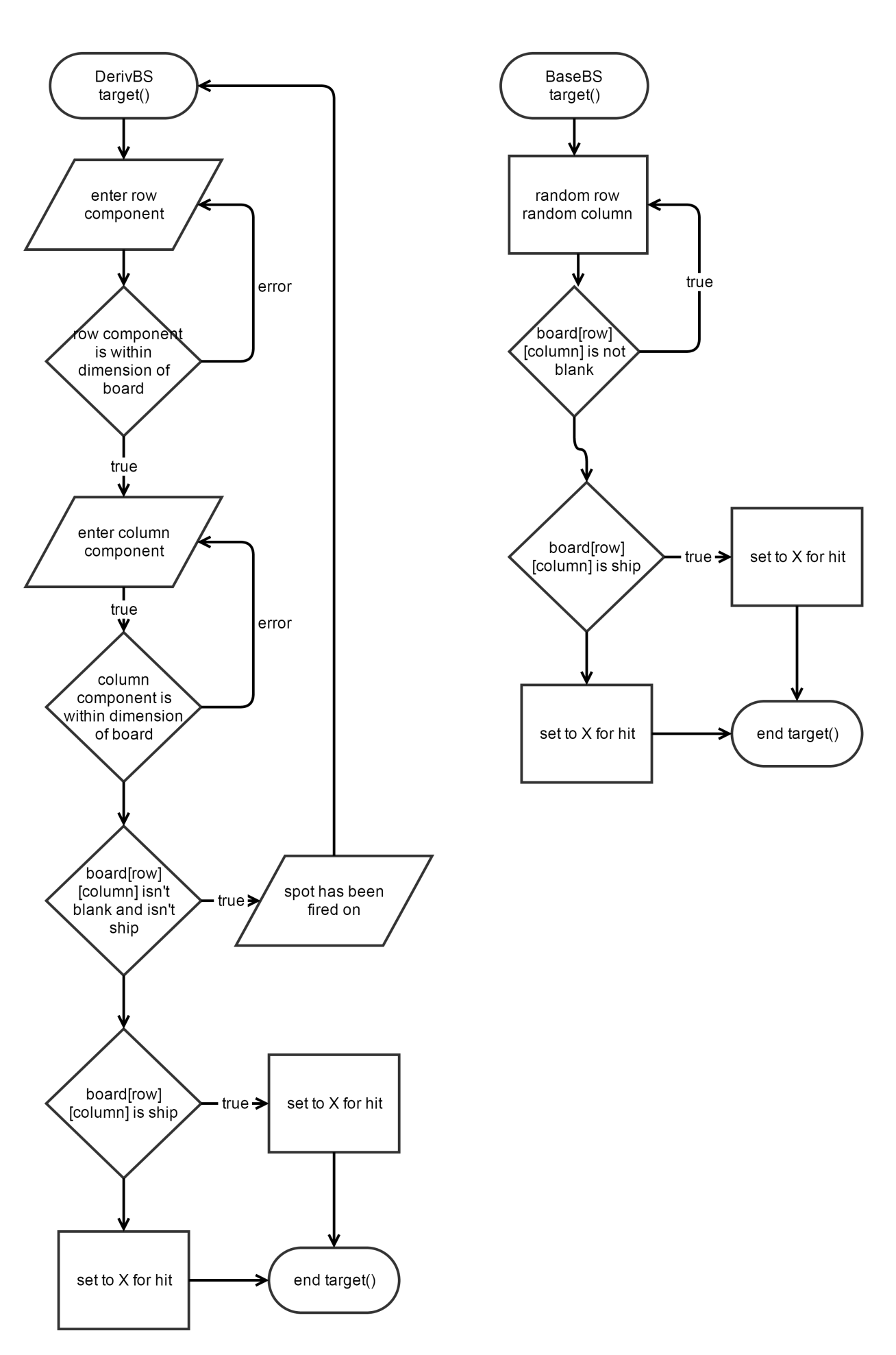
* 1. Flowcharts

Main menu:

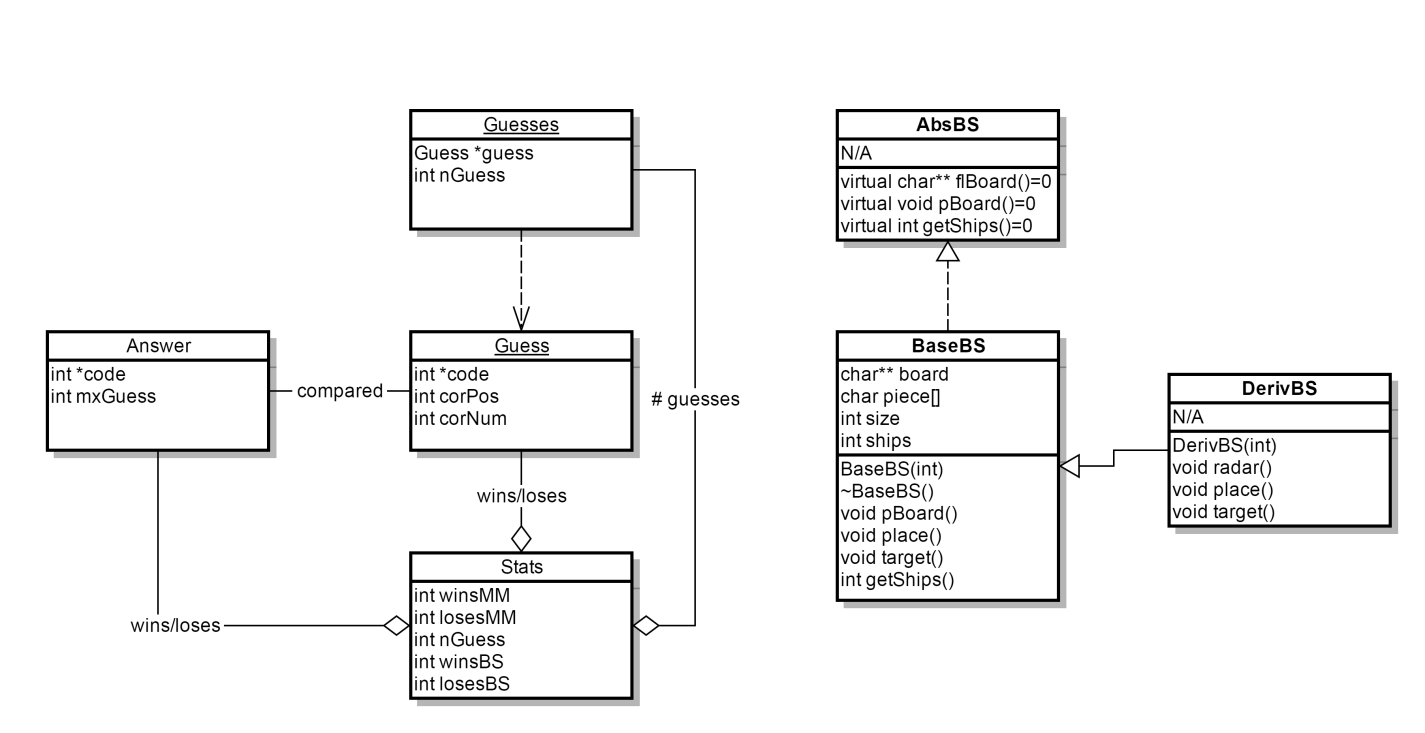
Mastermind Guess Checking Function (checkG):



Battleship Target member functions



* 1. Structures, Classes and Variables
     1. Structure and Classes UML:



* + 1. 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 |

* 1. 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 |

1. References

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

1. Code Documentation

Main

1. /\*
2. \* File: main.cpp
3. \* Author: Joseph Levin
4. \* C++ Project 2 - Spring 2015 43950
5. \* 6/5/2015
6. \*/
7. //System Libraries
8. #include <cstdlib>
9. #include <iostream>
10. #include <iomanip>
11. #include <ctime>
12. #include <string>
13. #include <cstring>
14. #include <fstream>
15. **using** **namespace** std**;**
16. //User Libraries
17. #include "mastermind.h"
18. #include "DerivBS.h"
19. #include "stats.h"
20. //Global Constants
21. //Function Prototypes
22. Answer **\***getAns**(**int**,**int**);**//generates an answer
23. bool again**();**//replay function
24. bool menu**();**//returns to menu
25. int **\***getG**(**int**);**//gets guess from user
26. int getL**();**//returns length for code combination
27. short slct**();**
28. Stats **\***load**();**//loads stats structure
29. void play**(**Stats **\*,**int**,**int**);**//launches the game
30. void pAns**(**Answer **\*,**Guesses **\*,**int**);**//prints the answer
31. void save**(**Stats **\*);**//saves the stats structure as binary file
32. void pBoard**(**Guesses **\*,**Answer **\*,**int**);**//prints the game board
33. void checkG**(**Answer **\*,**Guess **\*,**int**);**//checks guess against answer
34. void instrct**();**
35. void purge**(**Answer **\*,** Guesses **\*,**int**);**//deletes all structures related to a game
36. void seeStats**(**Stats **\*);**//Displays stats
37. void clrscrn**();**//clears screen
38. void playMM**(**Stats **\*);**//driver for mastermind game
39. void playBS**(**Stats **\*);**//driver for battleship
40. int getDim**();**//gets dimension for BattleShip board
41. //Begin
42. int main**(**int argc**,** char**\*\*** argv**)** **{**
43. srand**(**time**(**0**));**
44. short optn**;**
45. bool optnChk**;**//for checking if menu option is valid
46. Stats **\***s**=new** Stats**;**
47. //initialize all Stats variables to 0
48. s**->**gamesMM**=**0**;**
49. s**->**winsMM**=**0**;**
50. s**->**losesMM**=**0**;**
51. s**->**gamesBS**=**0**;**
52. s**->**winsBS**=**0**;**
53. s**->**losesBS**=**0**;**
54. s**->**nGuess**=**0**;**
55. //display menu
56. **do{**
57. optnChk**=false;**
58. clrscrn**();**
59. //get menu selection
60. optn**=**slct**();**
61. **switch(**optn**){**
62. //display instructions
63. **case** 1**:**
64. **do{**
65. clrscrn**();**
66. instrct**();**
67. **}while(!**menu**());**
68. **break;**
69. //load a stats struct
70. **case** 2**:**
71. s**=**load**();**
72. **break;**
73. //view stats
74. **case** 3**:**
75. **do{**
76. clrscrn**();**
77. seeStats**(**s**);**
78. **}while(!**menu**());**
79. **break;**
80. //play Mastermind
81. **case** 4**:**
82. **do{**
83. clrscrn**();**
84. playMM**(**s**);**
85. **}while(**again**());**
86. **break;**
87. //play BattleShip
88. **case** 5**:**
89. **do{**
90. clrscrn**();**
91. playBS**(**s**);**
92. **}while(**again**());**
93. **break;**
94. //exit case
95. **case** **-**1**:**
96. cout**<<**"Thanks for playing!"**<<**endl**;**
97. **break;**
98. //invalid option
99. **default:**
100. cout**<<**"Invalid option selected..."
101. "it's not that hard, really."**<<**endl**;**
102. **break;**
103. **}**
104. **}while(**optn**!=-**1**);**
105. //clean up!
106. **delete** s**;**
108. //It's over!
109. **return** 0**;**
110. **}**
111. //!Clear screen function outputs a ton of new lines in order to clear
112. //!the command prompt to look nice
113. void clrscrn**(){**
114. **for(**int i**=**0**;** i**<**100**;** i**++)**
115. cout**<<**endl**;**
116. **}**//end
117. //!getAns dynamically creates a Answer struct, fills the code array with
118. //!random integers 0-9, sets nGuess to max (max # of guesse) and then returns
119. Answer **\***getAns**(**int max**,** int row**){**
120. Answer **\***answer**=new** Answer**;**
121. answer**->**mxGuess**=**max**;**//max number of guesses allowed
122. answer**->**code**=new** int**[**row**];**//the combination is 'row' digits long
123. **for(**int i**=**0**;**i**<**row**;**i**++)**
124. answer**->**code**[**i**]=**rand**()%**8**+**1**;**//fill code with 1-9
125. **return** answer**;**
126. **}**//end
127. //!checkG takes in an Answer struct. It copies the contents of the code array
128. //!into another int array and compares each element to a Guess array
129. //!It changes the elments in the temp int array to -1 as it finds matches
130. //!in order to ensure no duplicate matches are found
131. //!if an element in the guess matches the position and number of the answer
132. //!both the correct number and correct position counters are incremented
133. //!otherwise if only the number is matched then the correct number indication
134. //!is incremented. Man this is a long description.
135. void checkG**(**Answer **\***a**,**Guesses **\***g**,**int row**){**
136. int **\***ta**=new** int**[**row**];**//temp array to store answer
137. int **\***tg**=new** int**[**row**];**//temp array to store guess
138. **for(**int i**=**0**;**i**<**row**;**i**++){**
139. ta**[**i**]=**a**->**code**[**i**];**//copy answer to temp answer
140. tg**[**i**]=**g**->**guess**[**g**->**nGuess**-**1**].**code**[**i**];**//copy guess to temp guess
141. **}**
142. //loop through arrays to check for both correct position and correct number
143. **for(**int i**=**0**;**i**<**row**;**i**++){**
144. **if(**tg**[**i**]==**ta**[**i**]){**
145. g**->**guess**[**g**->**nGuess**-**1**].**corPos**++;**
146. g**->**guess**[**g**->**nGuess**-**1**].**corNum**++;**
147. ta**[**i**]=-**1**;**//no duplicates
148. tg**[**i**]=-**1**;**
149. **}**
150. **}**
151. //loop through temp answer again
152. **for(**int i**=**0**;**i**<**row**;**i**++){**
153. //loop through guess
154. **for(**int j**=**0**;**j**<**row**;**j**++){**
155. //check for same number
156. **if(**ta**[**i**]==**tg**[**j**]&&**i**!=**j**&&**tg**[**j**]!=-**1**&&**ta**[**i**]!=-**1**){**//same num, diff pos
157. g**->**guess**[**g**->**nGuess**-**1**].**corNum**++;**
158. ta**[**i**]=-**1**;**
159. tg**[**j**]=-**1**;**
160. **}**
161. **}**
162. **}**
163. **delete** **[]**ta**;**
164. **delete** **[]**tg**;**
165. **}**//end
166. /\*!
167. \* getG prompts the user to enter a 4 digit combination, separated by spaces
168. \* it stores it as a string, and checks if every odd element is a digit 1-8
169. \* and then checks if every even is a space
170. \* if this is true, it then converts each odd element into an int and stores
171. \* it in an int array and returns this array
172. \* takes an int to specify how big an int array to create
173. \*/
174. int **\***getG**(**int row**){**
175. //declare variables
176. int **\***temp**=new** int**[**row**];**//will be returned
177. string guess**;**//for character checking
178. bool check**;**//input validation
179. cout**<<**"Enter a "**<<**row
180. **<<**" digit combination (1-8) separated by spaces."**<<**endl**;**
181. **do{**
182. check**=true;**
183. getline**(**cin**,**guess**);**
184. **if(**guess**.**size**()!=**2**\***row**-**1**){**//row digits plus spaces between
185. cout**<<**"You must enter "**<<**row**<<**" digits separated by spaces."
186. **<<**"Try again!"**<<**endl**;**
187. check**=false;;**
188. **}**
189. **else{**
190. //every odd must be 1-8, and every even must be space
191. **for(**int i**=**0**;**i**<**2**\***row**-**1**;**i**++){**
192. //check odds for all digits
193. **if(**i**%**2**==**0**&&!**isdigit**(**guess**[**i**])){**
194. check**=false;**
195. cout**<<**"Please only enter digits. Try again!"**<<**endl**;**
196. **}**
197. //check if all digits are 1-8
198. **else** **if(**i**%**2**==**0**&&(**atoi**(&**guess**[**i**])>**8**||**atoi**(&**guess**[**i**])<**1**)){**
199. check**=false;**
200. cout**<<**"Digits must all be between 1 & 8. Try again!"**<<**endl**;**
201. **break;**
202. **}**
203. //check if every even is space
204. **else{**
205. **if(!**isspace**(**guess**[**i**])&&!**isdigit**(**guess**[**i**])){**
206. check**=false;**
207. cout**<<**"Every digit must be "
208. **<<**"separated by a space. Try again!"**<<**endl**;**
209. **}**
210. **}**
211. **}**
212. **}**
213. **}while(!**check**);**
214. //copy digits into int array to return
215. **for(**int i**=**0**;**i**<**row**;**i**++)**
216. temp**[**i**]=**atoi**(&**guess**[**2**\***i**]);**
217. //int array ready to return
218. **return** temp**;**
219. **}**//end
220. /\*!
221. \* pBoard takes in pointers to an Answer and Guess, and an int
222. \* specifying the code length. It prints out dashed lines indicating
223. \* how many guesses remain and also prints out all previous guesses
224. \*/
225. void pBoard**(**Guesses **\***g**,** Answer **\***a**,** int r**){**
226. //X's represent the mystery code
227. **for(**int i**=**0**;**i**<**r**;**i**++)**
228. cout**<<**"X "**;**
229. cout**<<**endl**;**
230. //-'s represent spaces left for remaining guesses
231. **for(**int i**=**0**;**i**<(**a**->**mxGuess**)-(**g**->**nGuess**)+**1**;**i**++){**
232. **for(**int j**=**0**;**j**<**r**;**j**++)**
233. cout**<<**"- "**;**
234. cout**<<**endl**;**
235. **}**
236. //all previous guesses
237. **if(**g**->**nGuess**-**1**!=**0**){**
238. **for(**int i**=**g**->**nGuess**-**2**;**i**>=**0**;**i**--){**
239. **for(**int j**=**0**;**j**<**r**;**j**++)**
240. cout**<<**g**->**guess**[**i**].**code**[**j**]<<**" "**;**
241. cout**<<**" N:"**<<**g**->**guess**[**i**].**corNum
242. **<<**" P:"**<<**g**->**guess**[**i**].**corPos**<<**endl**;**
243. **}**
244. **}**
245. //finished
246. **}**//end
247. /\*!
248. \* getL prompts the user to enter either 4, 6, or 8. It then returns
249. \* the value selected as an int. No parameters
250. \*/
251. int getL**(){**
252. int l**;**//length of code
253. bool check**=false;**//error checking flag
254. cout **<<**"Please the code length!"**<<**endl**<<**"Options: 4, 6, 8"**<<**endl**;**
255. **do{**
256. cin**>>**l**;**
257. **if(**cin**.**fail**()||(**l**!=**4**&&**l**!=**6**&&**l**!=**8**)){**
258. cin**.**clear**();**
259. cin**.**ignore**(**256**,**'\n'**);**
260. cout**<<**"Sorry! You have to choose "
261. "either 4, 6, or 8. Try again!"**<<**endl**;**
262. **}**
263. **else**
264. check**=true;**
265. **}while(!**check**);**
266. **return** l**;**
267. **}**//end
268. /\*!
269. \* pAns prints the answer code
270. \*/
271. void pAns**(**Answer **\***a**,**Guesses **\***g**,**int r**){**
272. //Show Answer
273. **for(**int i**=**0**;**i**<**r**;**i**++)**
274. cout**<<**a**->**code**[**i**]<<**" "**;**
276. cout**<<**" <- Answer!"**<<**endl**;**
277. //-'s represent spaces left for remaining guesses
278. **for(**int i**=**0**;**i**<(**a**->**mxGuess**)-(**g**->**nGuess**);**i**++){**
279. **for(**int j**=**0**;**j**<**r**;**j**++)**
280. cout**<<**"- "**;**
281. cout**<<**endl**;**
282. **}**
283. //all previous guesses
284. **if(**g**->**nGuess**!=**0**){**
285. **for(**int i**=**g**->**nGuess**-**1**;**i**>=**0**;**i**--){**
286. **for(**int j**=**0**;**j**<**r**;**j**++)**
287. cout**<<**g**->**guess**[**i**].**code**[**j**]<<**" "**;**
288. cout**<<**" N:"**<<**g**->**guess**[**i**].**corNum
289. **<<**" P:"**<<**g**->**guess**[**i**].**corPos**<<**endl**;**
290. **}**
291. **}**
292. **}**//end
293. /\*!
294. \* play is the main driver for Mastermind gameplay. It handles turn taking,
295. \* win/lose checks, as well as stat saving. Void function, takes in an integer
296. \* to determine max # of guesses and an integer to determine code length
297. \*/
298. void play**(**Stats **\***s**,**int m**,**int r**){**
299. //generate answer
300. cin**.**clear**();**
301. cin**.**ignore**(**265**,**'\n'**);**
302. char optn**;**
303. Answer **\***a**=**getAns**(**m**,**r**);**
304. //generate and prepare Guesses
305. Guesses **\***g**=new** Guesses**;**
306. g**->**nGuess**=**0**;**
307. g**->**guess**=new** Guess**[**m**];**
308. **for(**int i**=**0**;**i**<**m**;**i**++){**
309. g**->**guess**[**i**].**corNum**=**0**;**
310. g**->**guess**[**i**].**corPos**=**0**;**
311. **}**
312. //loop until win/lose
313. **do{**
314. clrscrn**();**
315. g**->**nGuess**++;**
316. pBoard**(**g**,**a**,**r**);**
317. g**->**guess**[**g**->**nGuess**-**1**].**code**=**getG**(**r**);**
318. checkG**(**a**,**g**,**r**);**
319. **}while(**g**->**nGuess**<**a**->**mxGuess**&&**g**->**guess**[**g**->**nGuess**-**1**].**corPos**!=**r**);**
320. //display answer and determine win or lose and increment stats counters
321. clrscrn**();**
322. pAns**(**a**,**g**,**r**);**
323. s**->**gamesMM**++;**
324. **if(**g**->**guess**[**g**->**nGuess**-**1**].**corPos**==**r**){**
325. cout**<<**"You cracked the code! You win!"**<<**endl**;**
326. s**->**winsMM**++;**
327. **}**
328. **else{**
329. cout**<<**"You didn't crack the code! You lose... Sorry!"**<<**endl**;**
330. s**->**losesMM**++;**
331. **}**
332. s**->**nGuess**+=**g**->**nGuess**;**
333. cout**<<**"Would you like to save your stats? y/n"**<<**endl**;**
334. **do{**
335. cin**>>**optn**;**
336. **if(**tolower**(**optn**)!=**'y'**&&**tolower**(**optn**)!=**'n'**)**
337. cout**<<**"Sorry, that's not a valid option. Please try again."**<<**endl**;**
338. **else** **if(**tolower**(**optn**)==**'y'**){**
339. save**(**s**);**
340. cout**<<**"Stats saved!"**<<**endl**;**
341. **}**
342. **else**
343. cout**<<**"Stats not saved."**<<**endl**;**
344. **}while(**tolower**(**optn**)!=**'y'**&&**tolower**(**optn**)!=**'n'**);**
345. //clean up
346. purge**(**a**,**g**,**m**);**
347. **}**//end
348. /\*!
349. \* purge takes in an Answers and Guesses pointer and deletes all dynamically
350. \* allocated elements of the struct, as well as the structs themselves
351. \* and then points them to nullptrs
352. \*/
353. void purge**(**Answer **\***a**,**Guesses **\***g**,**int m**){**
354. **delete** **[]**a**->**code**;**
355. a**->**code**=NULL;**
356. **delete** a**;**
357. **for(**int i**=**0**;**i**<**m**;**i**++){**
358. **delete** **[]**g**->**guess**[**i**].**code**;**
359. g**->**guess**[**i**].**code**=NULL;**
360. **}**
361. **delete** g**->**guess**;**
362. g**->**guess**=NULL;**
363. **delete** g**;**
364. g**=NULL;**
365. **}**
366. /\*!
367. \* save takes in a Stats struct pointer. It prompts the user for a name
368. \* to store the stats struct under, and then writes the contents of the Stats
369. \* to a binary file. Returns void
370. \*/
371. void save**(**Stats **\***s**){**
372. cin**.**clear**();**
373. cin**.**ignore**(**256**,**'\n'**);**
374. ofstream out**;**//file stream
375. cout**<<**"Enter the name to store the stats under"**<<**endl**;**
376. string name**;**
377. getline**(**cin**,**name**);**
378. out**.**open**(**name**.**c\_str**(),**ios**::**binary**);**
379. out**.**write**(reinterpret\_cast<**char **\*>(**s**),sizeof(**Stats**));**
380. out**.**close**();**
381. **}**//end
382. /\*!
383. \* load prompts the user for a name, and attempts to open a file
384. \* with that name. If found, it reads the contents into a Stats structure
385. \* and returns it.
386. \*/
387. Stats **\***load**(){**
388. cin**.**clear**();**
389. cin**.**ignore**(**256**,**'\n'**);**
390. string name**;**
391. ifstream in**;**
392. cout**<<**"Enter the name of the person whose stats to load"**<<**endl**;**
393. **do{**
394. getline**(**cin**,**name**);**
395. in**.**open**(**name**.**c\_str**(),**ios**::**binary**);**
396. **if(**in**.**fail**()){**
397. cout**<<**"Name not found. Try again!"**<<**endl**;**
398. **}**
399. **else{**
400. Stats **\***s**=new** Stats**;**
401. in**.**read**(reinterpret\_cast<**char **\*>(**s**),sizeof(**Stats**));**
402. in**.**close**();**
403. **return** s**;**
404. **}**
405. **}while(**in**.**fail**());**
406. **}**
407. //!slct serves to take in input for menu selection, performs error checks
408. //!and then returns the value if it passes checks
409. short slct**(){**
410. short pick**;** //for menu selection
411. bool check**=false;**
412. cout**<<**"Welcome to Joseph Levin's Project 2!"**<<**endl**;**
413. cout**<<**"Choose an option from the menu: "**<<**endl
414. **<<**"1. View Instructions"**<<**endl
415. **<<**"2. Load Stats File"**<<**endl
416. **<<**"3. View Stats"**<<**endl
417. **<<**"4. Play Mastermind!"**<<**endl
418. **<<**"5. Play BattleShip!"**<<**endl
419. **<<**"-1 to quit"**<<**endl**;**
420. **do{**
421. cin**>>**pick**;**
422. **if(**cin**.**fail**()||**pick**<=**0**&&**pick**!=-**1**||**pick**>**5**){**//error checking
423. cin**.**clear**();**
424. cin**.**ignore**(**256**,**'\n'**);**
425. cout**<<**"Error. Invalid selection. Try again."**<<**endl**;**
426. **}**
427. **else**
428. check**=true;**//valid input
429. **}while(!**check**);**
430. **return** pick**;**
431. **}**
432. /\*!
433. \* again asks the user if they would like to play another game
434. \* it returns true if the user does, false if they do not
435. \*/
436. bool again**(){**
437. bool check**=false;**
438. char pick**;**
439. cout**<<**"Would you like to play again? y/n"**<<**endl**;**
440. **do{**
441. cin**>>**pick**;**
442. **if(**cin**.**fail**()||**tolower**(**pick**)!=**'y'**&&**tolower**(**pick**)!=**'n'**){**//only accepts
443. cin**.**clear**();** //y or n as input
444. cin**.**ignore**(**256**,**'\n'**);**
445. cout**<<**"Error. Invalid selection. Try again."**<<**endl**;**
446. **}**
447. **else** **if(**tolower**(**pick**)==**'y'**){**//user wants to repeat
448. check**=true;**
449. cin**.**clear**();**
450. cin**.**ignore**(**256**,**'\n'**);**
451. **return** **true;**
452. **}**
453. **else{** //user does not want to repeat
454. cin**.**clear**();**
455. cin**.**ignore**(**256**,**'\n'**);**
456. check**=true;**
457. **return** **false;**
458. **}**
459. **}while(!**check**);**
461. **}**//end
462. /\*!
463. \* menu prompts the user if they want to return to the main menu.
464. \* if the user types 'y', it returns true
465. \* if the user types 'n', it returns false
466. \*/
467. bool menu**(){**
468. bool check**=false;**
469. char pick**;**
470. cout**<<**"Would you like to return to the menu? y/n"**<<**endl**;**
471. **do{**
472. cin**>>**pick**;**
473. **if(**cin**.**fail**()||**tolower**(**pick**)!=**'y'**&&**tolower**(**pick**)!=**'n'**){**//only accepts
474. cin**.**clear**();** //y or n as input
475. cin**.**ignore**(**256**,**'\n'**);**
476. cout**<<**"Error. Invalid selection. Try again."**<<**endl**;**
477. **}**
478. **else** **if(**tolower**(**pick**)==**'y'**){**//user wants to repeat
479. check**=true;**
480. cin**.**clear**();**
481. cin**.**ignore**(**256**,**'\n'**);**
482. **return** **true;**
483. **}**
484. **else{** //user does not want to repeat
485. cin**.**clear**();**
486. cin**.**ignore**(**256**,**'\n'**);**
487. check**=true;**
488. **return** **false;**
489. **}**
490. **}while(!**check**);**
491. **}**
492. /\*!
493. \* seeStats takes in Stats pointer. It displays the elements within,
494. \* (wins/loses/total guesses) and also calculates win percentage, and
495. \* correct guess percentage
496. \*/
497. void seeStats**(**Stats **\***s**){**
498. **if(**s**->**gamesBS**!=**0**||**s**->**gamesMM**!=**0**){**
499. cout**<<**"Mastermind"**<<**endl**;**
500. cout**<<**"Games played: "**<<**s**->**gamesMM**<<**endl**;**
501. cout**<<**"Wins: "**<<**s**->**winsMM**<<**endl**;**
502. cout**<<**"Losses: "**<<**s**->**losesMM**<<**endl**;**
503. cout**<<**"Total guesses: "**<<**s**->**nGuess**<<**endl**;**
504. cout**<<**endl**;**
505. cout**<<**"BattleShip"**<<**endl**;**
506. cout**<<**"Games played: "**<<**s**->**gamesBS**<<**endl**;**
507. cout**<<**"Wins: "**<<**s**->**winsBS**<<**endl**;**
508. cout**<<**"Losses: "**<<**s**->**losesBS**<<**endl**;**
509. **}**
510. **else**
511. cout**<<**"Stats File is empty!"**<<**endl**;**
512. **}**
513. /\*!
514. \* instrct displays the rules for Code Breaker (based on Mastermind)
515. \*/
516. void instrct**(){**
517. cout**<<**" \*\*\*\*\*\*\*\*\*\*\*\*\*\*"**<<**endl**;**
518. cout**<<**" \* \*"**<<**endl**;**
519. cout**<<**" \* MASTERMIND \*"**<<**endl**;**
520. cout**<<**" \* \*"**<<**endl**;**
521. cout**<<**" \*\*\*\*\*\*\*\*\*\*\*\*\*\*"**<<**endl**;**
522. cout**<<**"Mastermind is a game of decryption!"**<<**endl
523. **<<**"In it, the player (that's you!)"**<<**endl
524. **<<**"attempts to decipher a secret code"**<<**endl
525. **<<**"generated at random. The code consists"**<<**endl
526. **<<**"of integers between 1 and 8."**<<**endl
527. **<<**"The player chooses from 3 code lengths (4, 6, 8),"**<<**endl
528. **<<**"and guesses at each of the individual digits"**<<**endl
529. **<<**"that make up the secret code. After each"**<<**endl
530. **<<**"attempt the player is told how many correct "**<<**endl
531. **<<**"numbers (shown as 'N') and correct position "**<<**endl
532. **<<**"(shown as 'P') the attempt had. The player"**<<**endl
533. **<<**"wins by correctly determining all 4 correct"**<<**endl
534. **<<**"numbers and positions. Remember, a guess can"**<<**endl
535. **<<**"have 4 correct numbers but 0 correct positions,"**<<**endl
536. **<<**"but you can't have a correct position without"**<<**endl
537. **<<**"it also being the correct number."**<<**endl**;**
538. cout**<<**endl**;**
539. cout**<<**" \*\*\*\*\*\*\*\*\*\*\*\*\*\*"**<<**endl**;**
540. cout**<<**" \* \*"**<<**endl**;**
541. cout**<<**" \* BATTLESHIP \*"**<<**endl**;**
542. cout**<<**" \* \*"**<<**endl**;**
543. cout**<<**" \*\*\*\*\*\*\*\*\*\*\*\*\*\*"**<<**endl**;**
544. cout**<<**"Battleship is a guessing game!"**<<**endl
545. **<<**"In it, the player (that's you again!)"**<<**endl
546. **<<**"attempts to find and destroy the enemy's ships,"**<<**endl
547. **<<**"which are placed randomly. However, the enemy"**<<**endl
548. **<<**"also fires on the player! The player starts"**<<**endl
549. **<<**"by choosing the starting location of his ships"**<<**endl
550. **<<**"and then takes turns guessing at the location"**<<**endl
551. **<<**"of the enemy's ships by entering the row and"**<<**endl
552. **<<**"column coordinates of the spot on the board"**<<**endl
553. **<<**"at which to 'fire' upon. Hits are designated"**<<**endl
554. **<<**"with an 'X', and misses are 'O'. The game"**<<**endl
555. **<<**"until either the player or the computer runs"**<<**endl
556. **<<**"out of ships."**<<**endl**;**
557. cout**<<**endl**<<**"Good luck!"**<<**endl**;**
558. **}**//end
559. /\*!
560. \* playMMis the main driver for Mastermind.
561. \* Parameters: a stats struct for storing stats
562. \*/
563. void playMM**(**Stats **\***s**){**
564. clrscrn**();**
565. int l**=**getL**();**
566. play**(**s**,**10**,**l**);**
567. **}**
568. /\*!
569. \* playBS is the main driver for BattleShip.
570. \* Parameters: a stats struct for storing stats
571. \*/
572. void playBS**(**Stats **\***s**){**
573. int dim**;**//for storing dimension of board
574. char optn**;**//for stat saving selection
575. bool sConf**=false;**//confirming size is valid
576. cout**<<**"Welcome to BattleShip!"
577. " Please select the size board you'd like to play."**<<**endl**;**
578. cout**<<**"1. 6x6"**<<**endl**;**
579. cout**<<**"2. 8x8"**<<**endl**;**
580. cout**<<**"3. 10x10"**<<**endl**;**
581. **do{**
582. **try{**
583. dim**=**getDim**();**
584. sConf**=true;**
585. **}**
586. **catch(**string invalid**){**
587. cout**<<**invalid**<<**endl**;**
588. **}**
589. **}while(!**sConf**);**
590. //initialize BaseBS (for player) and DerivBS (for computer) of dim
591. BaseBS player**(**dim**);**
592. DerivBS comp**(**dim**);**
593. //randomly place computer's ships
594. comp**.**place**();**
595. //place players ships
596. player**.**place**();**
597. //begin rounds
598. **do{**
599. clrscrn**();**
600. //display board and radar
601. cout**<<**"Player board ("**<<**player**.**getShips**()<<**" ships remain)"**<<**endl**;**
602. player**.**pBoard**();**
603. cout**<<**endl**;**
604. cout**<<**"Radar ("**<<**comp**.**getShips**()<<**" enemy ships remain)"**<<**endl**;**
605. comp**.**radar**();**
606. //targeting round
607. player**.**target**();**
608. comp**.**target**();**
609. **}while(**comp**.**getShips**()!=**0**&&**player**.**getShips**()!=**0**);**
610. clrscrn**();**
611. //Display boards final time
612. cout**<<**"Player board ("**<<**player**.**getShips**()<<**" ships remain)"**<<**endl**;**
613. player**.**pBoard**();**
614. cout**<<**endl**;**
615. cout**<<**"Radar ("**<<**comp**.**getShips**()<<**" enemy ships remain)"**<<**endl**;**
616. comp**.**radar**();**
617. //Determine victor and increment stats counters
618. s**->**gamesBS**++;**
619. **if(**comp**.**getShips**()==**0**){**
620. cout**<<**"You win! Congratulations!"**<<**endl**;**
621. s**->**winsBS**++;**
622. **}**
623. **else{**
624. cout**<<**"You lose. Sorry!"**<<**endl**;**
625. s**->**losesBS**++;**
626. **}**
627. cout**<<**"Would you like to save your stats? y/n"**<<**endl**;**
628. **do{**
629. cin**>>**optn**;**
630. **if(**tolower**(**optn**)!=**'y'**&&**tolower**(**optn**)!=**'n'**)**
631. cout**<<**"Sorry, that's not a valid option. Please try again."**<<**endl**;**
632. **else** **if(**tolower**(**optn**)==**'y'**){**
633. save**(**s**);**
634. cout**<<**"Stats saved!"**<<**endl**;**
635. **}**
636. **else**
637. cout**<<**"Stats not saved."**<<**endl**;**
638. **}while(**tolower**(**optn**)!=**'y'**&&**tolower**(**optn**)!=**'n'**);**
639. **}**//end
640. /\*!
641. \* getDim prompts user for dimension for BattleShip board. It will throw
642. \* an exception if invalid, otherwise returns the value
643. \*/
644. int getDim**(){**
645. int dim**;**
646. cin**>>**dim**;**
647. **if(**dim**<**1**||**dim**>**3**||**cin**.**fail**()){**
648. string invalid**=**"Invalid selection for dimension. Try again."**;**
649. **throw** invalid**;**
650. **}**
651. //Reassign dim to the corresponding dimension
652. **if(**dim**==**1**){**
653. dim**=**6**;**
654. **}**
655. **else** **if(**dim**==**2**){**
656. dim**=**8**;**
657. **}**
658. **else{**
659. dim**=**10**;**
660. **}**
661. **return** dim**;**
662. **}**

Mastermind.h

1. /\*
2. \* File: mastermind.h
3. \* Author: Joseph
4. \*
5. \* Created on April 26, 2015, 7:47 PM
6. \*/
7. #ifndef MASTERMIND\_H
8. #define MASTERMIND\_H
9. /\*!
10. \* Answer stores the correct code in an int array and the max guesses allowed
11. \*/
12. struct Answer**{**
13. //!stores the integer combination
14. int **\***code**;**
15. //!keeps track of maximum guesses allowed
16. int mxGuess**;**
17. **};**
18. /\*!
19. \* Guess stores a guess in an int array, the number of correct positions
20. \* associated with the guess and the number of correct numbers
21. \*/
22. struct Guess**{**
23. //!stores the guess
24. int **\***code**;**
25. //!tallies the number of correct positions in a guess
26. int corPos**;**
27. //!tallies the number of correct numbers in a guess
28. int corNum**;**
29. **};**
30. /\*!
31. \* Guesses stores a pointer a Guess (one for each turn) and
32. \* the total number of guesses taken
33. \*/
34. struct Guesses**{**
35. //!array of guess attempts
36. Guess **\***guess**;**
37. //!how many guesses have been taken
38. int nGuess**;**
39. **};**
40. #endif /\* MASTERMIND\_H \*

Stats.h

1. /\*
2. \* File: stats.h
3. \* Author: Joseph Levin
4. \* C++ Project 2 - Spring 2015 43950
5. \* 6/5/2015
6. \*/
7. #ifndef STATS\_H
8. #define STATS\_H
9. /\*!
10. \* Stats stores the number of wins, losses, and number of guesses
11. \*/
12. struct Stats**{**
13. //!number of wins for Mastermind
14. int winsMM**;**
15. //!number of loses for Mastermind
16. int losesMM**;**
17. //!number of guesses for Mastermind
18. int nGuess**;**
19. //!number of wins for BattleShip
20. int winsBS**;**
21. //!number of loses for BattleShip
22. int losesBS**;**
23. //!tracks games played
24. int gamesBS**;**
25. int gamesMM**;**
26. **};**
27. #endif /\* STATS\_H \*/

AbsBS.h

1. /\*
2. \* File: AbsBS
3. \* Author: Joseph Levin
4. \* C++ Project 2 - Spring 2015 43950
5. \* 6/5/2015
6. \*/
7. #ifndef ABSBS\_H
8. #define ABSBS\_H
9. //!Abstract class for the Battleship board
10. class AbsBS**{**
11. public**:**
12. //!2D char array to store the board
13. virtual char**\*\*** flBoard**()=**0**;**
14. //!outputs the board
15. virtual void pBoard**()=**0**;**
16. //!returns ships remaining
17. virtual int getShips**()=**0**;**
18. **};**
19. #endif /\* ABSBS\_H \*/

BaseBS.h

1. /\*
2. \* File: BaseBS.h
3. \* Author: Joseph Levin
4. \* C++ Project 2 - Spring 2015 43950
5. \* 6/5/2015
6. \*/
7. #ifndef BASEBS\_H
8. #define BASEBS\_H
9. #include "AbsBS.h"
10. /\*!
11. \* BaseBS is the base class for the battleship board.
12. \*/
13. class BaseBS**:**AbsBS**{**
14. protected**:**
15. //!for storing the board
16. char**\*\*** board**;**
17. //!X is hit, O is miss, + is ship, <space> is empty
18. char piece**[**4**]={**'X'**,**'O'**,**'+'**,** ' '**};**
19. //!dimension of board
20. int size**;**
21. //!ships on board
22. int ships**;**
23. public**:**
24. //!Constructor for base battleship board. Takes in an integer for
25. //!the dimension of the board (area is nxn))
26. BaseBS**(**int**);**
27. //!Destructor for the board
28. **~**BaseBS**();**
29. //!For initially filling board
30. char**\*\*** flBoard**();**
31. //!For outputting the board formatted to show ships
32. void pBoard**();**
33. //!place handles the ship placing procedure for the player.
34. //!it checks to make sure the given coordinates are within
35. //!the acceptable range of size, and then sets the given tile
36. //!to the + char to represent a ship has been placed
37. void place**();**
38. //!target handles the process of targeting a spot to fire on it
39. void target**();**
40. //!getShips returns the value of the ships member variable
41. int getShips**();**
42. **};**
43. #endif /\* BASEBS\_H \*/

BaseBS.cpp

1. /\*
2. \* File: BaseBS.cpp
3. \* Author: Joseph Levin
4. \* C++ Project 2 - Spring 2015 43950
5. \* 6/5/2015
6. \*/
7. //Libraries
8. #include <iostream>
9. #include <iomanip>//board formatting
10. #include <cstdlib>
11. #include <vector>//for rand
12. #include "BaseBS.h"
13. **using** **namespace** std**;**
14. BaseBS**::**BaseBS**(**int s**){**
15. size**=**s**;**
16. board**=**flBoard**();**
17. **}**
18. BaseBS**::~**BaseBS**(){**
19. **for(**int i**=**0**;**i**<**size**;**i**++){**
20. **delete** **[]**board**[**i**];**
21. **}**
22. **delete** **[]**board**;**
23. **}**
24. /\*!
25. \* flBoard() initializes the board with all empty spaces
26. \*/
27. char**\*\*** BaseBS**::**flBoard**(){**
28. char**\*\*** board **=** **new** char**\*[**size**];**
29. **for** **(**int i**=**0**;**i**<**size**;** i**++){**
30. board**[**i**]=new** char**[**size**];**
31. **}**
32. **for** **(**int i**=**0**;**i**<**size**;**i**++)** **{**
33. **for** **(**int j**=**0**;**j**<**size**;**j**++)**
34. board**[**i**][**j**]=**piece**[**3**];**
35. **}**
36. **return** board**;**
37. **}**
38. void BaseBS**::**pBoard**(){**
39. cout**<<**" "**;**
40. **for** **(**int i**=**0**;**i**<**size**;**i**++)** **{**
41. cout**<<**setw**(**3**)<<**setfill**(**' '**)<<**i**<<**" "**;**
42. **}**
43. cout**<<**endl**;**
44. **for** **(**int i**=**0**;**i**<**size**;** i**++)** **{**
45. cout**<<**" "**<<**setw**(**size**\***4**+**1**)<<**setfill**(**'-'**)<<**'-'**<<**endl**;**
46. cout**<<**i**<<**" "**;**
47. **for** **(**int j**=**0**;**j**<**size**;**j**++)** **{**
48. cout**<<**"|"**<<**setw**(**2**)<<**setfill**(**' '**)<<**board**[**i**][**j**]<<**" "**;**
49. **}**
50. cout**<<**"| "**<<**endl**;**
51. **}**
52. cout**<<**" "**<<**setw**(**size**\***4**+**1**)<<**setfill**(**'-'**)** **<<** '-' **<<** endl**;**
53. **}**
54. void BaseBS**::**place**(){**
55. bool conf1**,**conf2**,**conf3**;**//error buffers
56. ships**=**0**;**//begin with no ships placed
57. int row**,** col**;**//for checking the given coordinate
58. **for(**int i**=**0**;**i**<**size**;**i**++){**//will place size # of ships
59. **do{**
60. pBoard**();**
61. cout**<<**"Enter the row coordinate for where to begin ship "
62. **<<**ships**+**1**<<**endl**;**
63. //check confirms to false
64. conf1**=false;**
65. conf2**=false;**
66. conf3**=false;**
67. **do{**//gets the row coordinate
68. cin**>>**row**;**
69. **if** **(**cin**.**fail**()||(**row**<**0**||**row**>**size**-**1**)){**//size-1 denotes edge
70. cin**.**clear**();**
71. cin**.**ignore**(**256**,**'\n'**);**
72. cout**<<**"Error. Invalid input."**<<**endl**;**
73. **}** **else**
74. conf1**=true;**//row coordinate is acceptable
75. **}** **while(**cin**.**fail**()||**row**<**0**||**row**>**size**-**1**||!**conf1**);**
76. cout**<<**"Enter the column coordinate for where to begin ship "
77. **<<**ships**+**1**<<**endl**;**
78. **do{**
79. cin**>>**col**;**
80. **if** **(**cin**.**fail**()||(**col**<**0**||**col**>**size**-**1**)){**
81. cin**.**clear**();**
82. cin**.**ignore**(**256**,**'\n'**);**
83. cout**<<**"Error. Invalid input."**<<**endl**;**
84. **}** **else**
85. conf2**=true;**//column coordinate is acceptable
86. **}** **while(**cin**.**fail**()||**col**<**0**||**row**>**size**-**1**||!**conf1**);**
87. **if(**board**[**row**][**col**]!=**piece**[**3**]){**//piece[3] == blank space
88. cout **<<** "This spot is already occupied" **<<** endl**;**
89. **}**
90. **else** **{**
91. board**[**row**][**col**]=**piece**[**2**];**//piece[2]== '+'
92. ships**++;**//added ship to board
93. conf3**=true;**//process is completed successfully
94. **}**
95. **}while(!**conf3**);**
96. **}**
97. **}**
98. //!target for BaseBS handles the computer firing on the player's board
99. //!it is a glorified random number generator
100. void BaseBS**::**target**(){**
101. bool confirm**=false;**
102. int row**,**col**;**
103. //randomly fires at spots until it
104. **do{**
105. row**=(**rand**()%**size**);**
106. col**=(**rand**()%**size**);**
107. //piece[3]==blank, piece[2]==ship
108. **if(**board**[**row**][**col**]==**piece**[**3**]||**board**[**row**][**col**]==**piece**[**2**])**
109. confirm**=true;**
110. **}** **while(!**confirm**);**
111. //piece[2]==ship,piece[0]==X
112. **if** **(**board**[**row**][**col**]==**piece**[**2**]){**
113. board**[**row**][**col**]=**piece**[**0**];**
114. ships**--;**
115. //piece[1]==O
116. **}** **else**
117. board**[**row**][**col**]=**piece**[**1**];**
118. **}**
119. //!getShips returns the value of the ships member variable
120. int BaseBS**::**getShips**(){return** ships**;}**

DerivBS.h

1. /\*
2. \* File: DerivBS.h
3. \* Author: Joseph Levin
4. \* C++ Project 2 - Spring 2015 43950
5. \* 6/5/2015
6. \*/
7. #ifndef DERIVBS\_H
8. #define DERIVBS\_H
9. #include "BaseBS.h"
10. /\*!
11. \* DerivBS is a derived class of BaseBS. It contains addition functions
12. \* for handling the computer logic
13. \*/
14. class DerivBS**:**public BaseBS**{**
15. public**:**
16. //!constructor for DerivBS calls BaseBS constructor
17. DerivBS**(**int**);**
18. //!displays a version of the board with ships masked
19. void radar**();**
20. //!place for DerivBS randomly places ships for computer
21. void place**();**
22. //!target for DerivBS randomly fires on spots for computer
23. void target**();**
24. **};**
25. #endif /\* DERIVBS\_H \*/

DerivBS.cpp

1. /\*
2. \* File: DerivBS.cpp
3. \* Author: Joseph Levin
4. \* C++ Project 2 - Spring 2015 43950
5. \* 6/5/2015
6. \*/
7. #include "DerivBS.h"
8. #include<iostream>
9. #include<iomanip>
10. #include <cstdlib>
11. **using** **namespace** std**;**
12. //!constructor for DerivBS calls BaseBS constructor
13. DerivBS**::**DerivBS**(**int s**):**BaseBS**(**s**){;}**
14. //!displays a version of the board with ships masked
15. void DerivBS**::**radar**(){**
16. cout**<<**" "**;**
17. **for** **(**int i**=**0**;**i**<**size**;**i**++)** **{**
18. cout**<<**setw**(**3**)<<**setfill**(**' '**)<<**i**<<**" "**;**
19. **}**
20. cout**<<**endl**;**
21. **for** **(**int i**=**0**;**i**<**size**;** i**++)** **{**
22. cout**<<**" "**<<**setw**(**size**\***4**+**1**)<<**setfill**(**'-'**)<<**'-'**<<**endl**;**
23. cout**<<**i**<<**" "**;**
24. **for** **(**int j**=**0**;**j**<**size**;**j**++)** **{**
25. //if ship isn't at that spot, place piece at i,j
26. **if(**board**[**i**][**j**]!=**piece**[**2**]){**
27. cout**<<**"|"**<<**setw**(**2**)<<**setfill**(**' '**)<<**board**[**i**][**j**]<<**" "**;**
28. **}**
29. //if ship is at that spot, mask ship by placing blank spot instead
30. **else{**
31. cout**<<**"|"**<<**setw**(**2**)<<**setfill**(**' '**)<<**piece**[**3**]<<**" "**;**
32. **}**
33. **}**
34. cout**<<**"| "**<<**endl**;**
35. **}**
36. cout**<<**" "**<<**setw**(**size**\***4**+**1**)<<**setfill**(**'-'**)** **<<** '-' **<<** endl**;**
37. **}**
38. //!place for DerivBS randomly places ships for computer
39. void DerivBS**::**place**(){**
40. int row**,** col**;**
41. ships**=**0**;**
42. **for(**int i**=**0**;**i**<**size**;**i**++){**
43. //randomly picks spots until it finds one that isn't occupied already
44. **do{**
45. row**=(**rand**()%**size**);**
46. col**=(**rand**()%**size**);**
47. **}while(**board**[**row**][**col**]!=**piece**[**3**]);**//piece[3]==blank space
48. board**[**row**][**col**]=**piece**[**2**];**//piece[2] == '+', ship
49. ships**++;**
50. **}**
51. **}**
52. //!target for DerivBS handles the player firing on the computer's board
53. //!it prompts for the coordinates one component a time, checks for any errors
54. //!and decides if it was a hit or miss and announces such
55. void DerivBS**::**target**(){**
56. int row**,** col**;**
57. bool conf1**,**conf2**,**conf3**;**//error buffers
58. **do{**
59. conf1**=false;**
60. conf2**=false;**
61. conf3**=false;**
62. cout**<<**"Enter the row (vertical) component of the coordinate "
63. **<<**"you wish to fire upon"**<<**endl**;**
64. **do{**
65. cin**>>**row**;**
66. **if** **(**cin**.**fail**()||**row**<**0**||**row**>**size**-**1**)** **{**
67. cin**.**clear**();**
68. cin**.**ignore**(**256**,**'\n'**);**
69. cout**<<**"Error. Invalid selection." **<<** endl**;**
70. **}** **else**
71. conf1**=true;**
72. **}while(**cin**.**fail**()||**row**<**0**||**row**>**size**-**1**||!**conf1**);**
73. cout**<<**"Enter the column (horizontal) component "
74. **<<**"of the coordinate you wish to fire upon"**<<**endl**;**
75. **do{**
76. cin**>>**col**;**
77. **if(**cin**.**fail**()||**col**<**0**||**col**>**size**-**1**){**
78. cin**.**clear**();**
79. cin**.**ignore**(**256**,**'\n'**);**
80. cout**<<**"Error. Invalid selection."**<<**endl**;**
81. **}else**
82. conf2**=true;**
83. **}while(**cin**.**fail**()||**col**<**0**||**col**>**size**-**1**||!**conf2**);**
84. **if(**board**[**row**][**col**]!=**piece**[**3**]&&**board**[**row**][**col**]!=**piece**[**2**])**
85. cout**<<**"This spot has been fired upon already."**<<**endl**;**
86. **else**
87. conf3**=true;**
88. **}** **while** **(!**conf3**);**
89. //if a ship was hit, replace with X and decrease remaining ships
90. //and announce hit was successful
91. **if** **(**board**[**row**][**col**]==**piece**[**2**]){**
92. board**[**row**][**col**]=**piece**[**0**];**
93. ships**--;**
94. **}**
95. //if a ship wasn't hit, replace with O and announce miss
96. **else{**
97. board**[**row**][**col**]=**piece**[**1**];**
98. **}**
99. **}**