|  |
| --- |
|  |
| 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.

1. Summary

|  |  |
| --- | --- |
| 2.1 Statistics | |
| Program Length (lines of code). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 479 |
| # Structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| # Variables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 32 |
| # Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 16 |

* 1. Areas of Note in Development

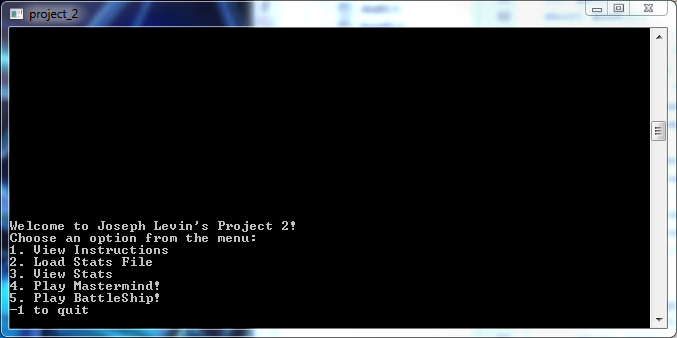
At the end of the day the most significant obstacle in the development of the project was my own indecisiveness. I debated with myself over a few different ways of implementing how to handle a user’s guesses as well as where to keep track of certain elements such as how many attempts the user has taken. I settled on the current iteration 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.

The logic I used for checking a guess against the correct pattern feels sort of gimmicky. I would have preferred to not rely as heavily on copying to temporary arrays in order to do my checking, and my solution of reassigning the elements in the temporary arrays to a negative to indicate it has already been checked feels very brute-force. However a more elegant process was and is lost to me.

From initial planning to typing this write up, the project took about a week and a half to complete. There are some features that I experimented with such as save files that I was unable to implement into the final project. Given the chance, I would like to revisit them.

1. Program Description

3.1 Program Walkthrough

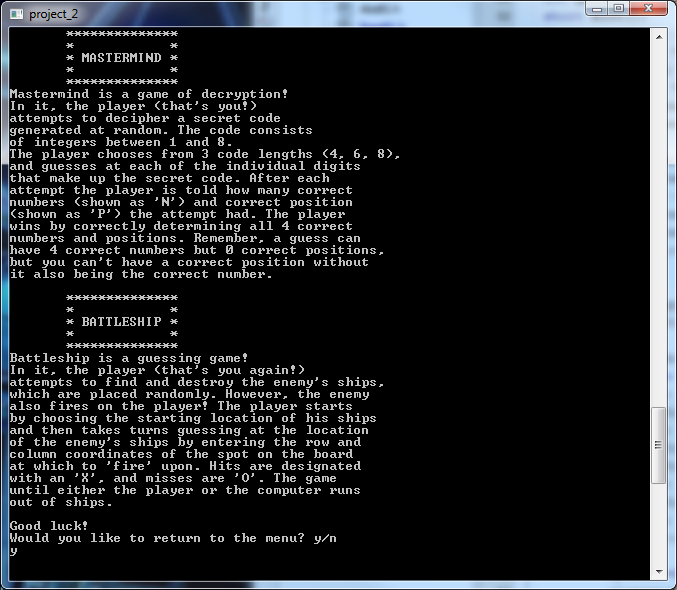


The main menu offer s5 options to the user.

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

-1 to quit

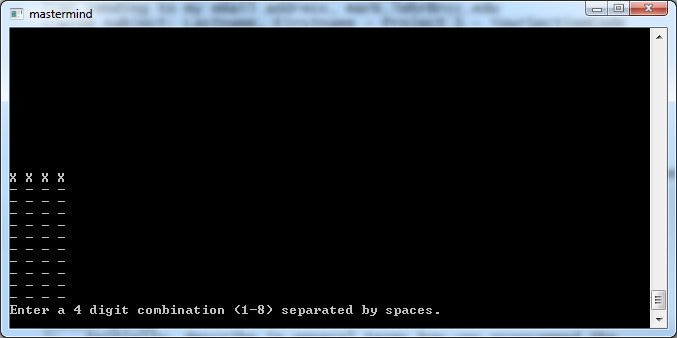
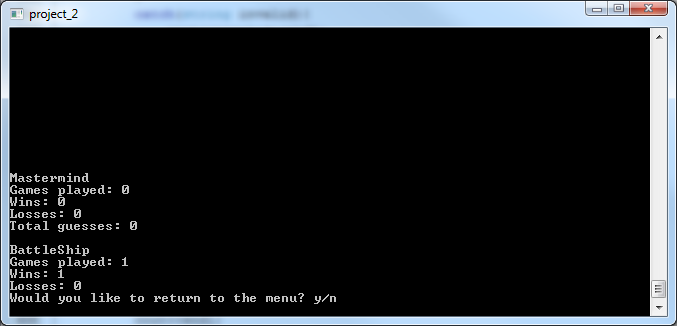
If the user inputs 1, they are shown a brief explanation of how Mastermind is played. It should be noted that the original game allows 6 different colors, but I forgot this and chose 8 instead.



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).

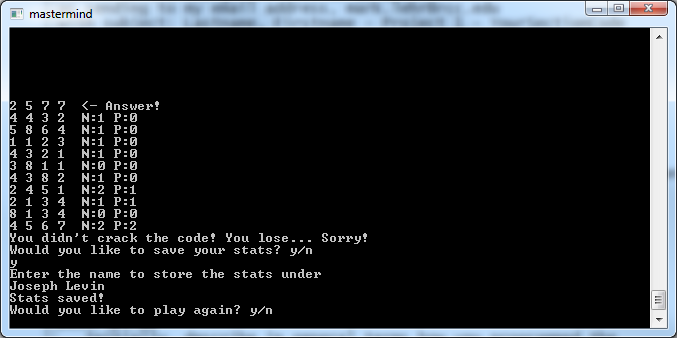
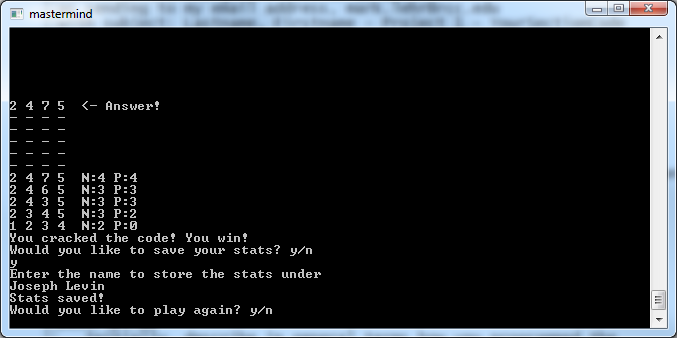
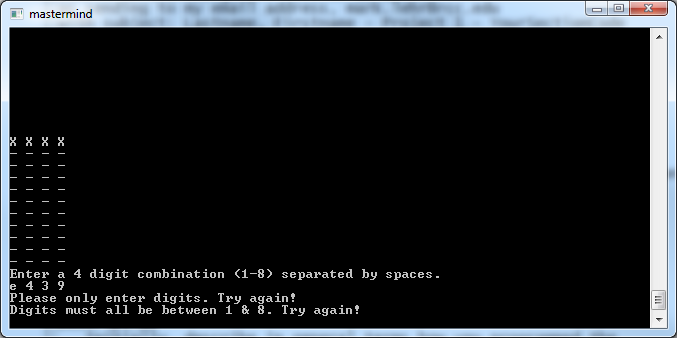
i.e, l=4, odds=1/(8^4)=0.0002

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.

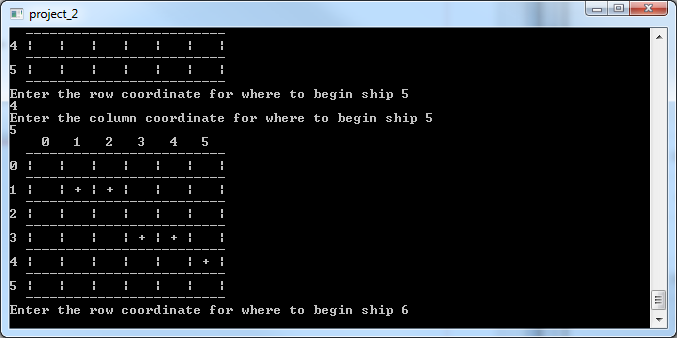
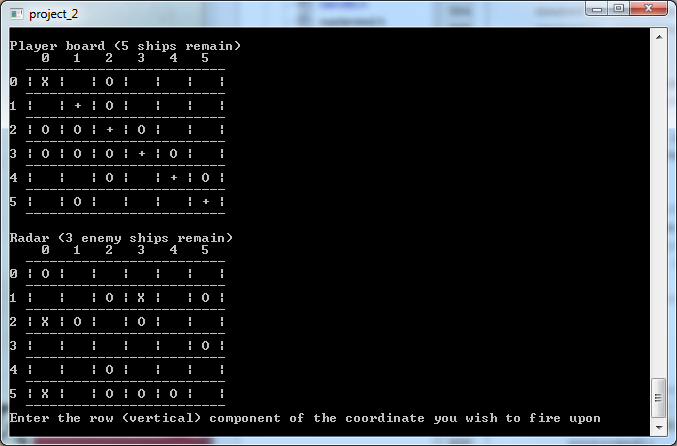
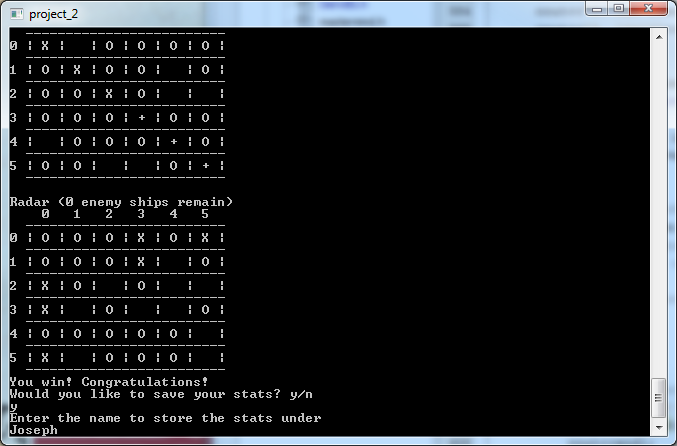
Non digits will give appropriate error, as will digits outside of range



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.

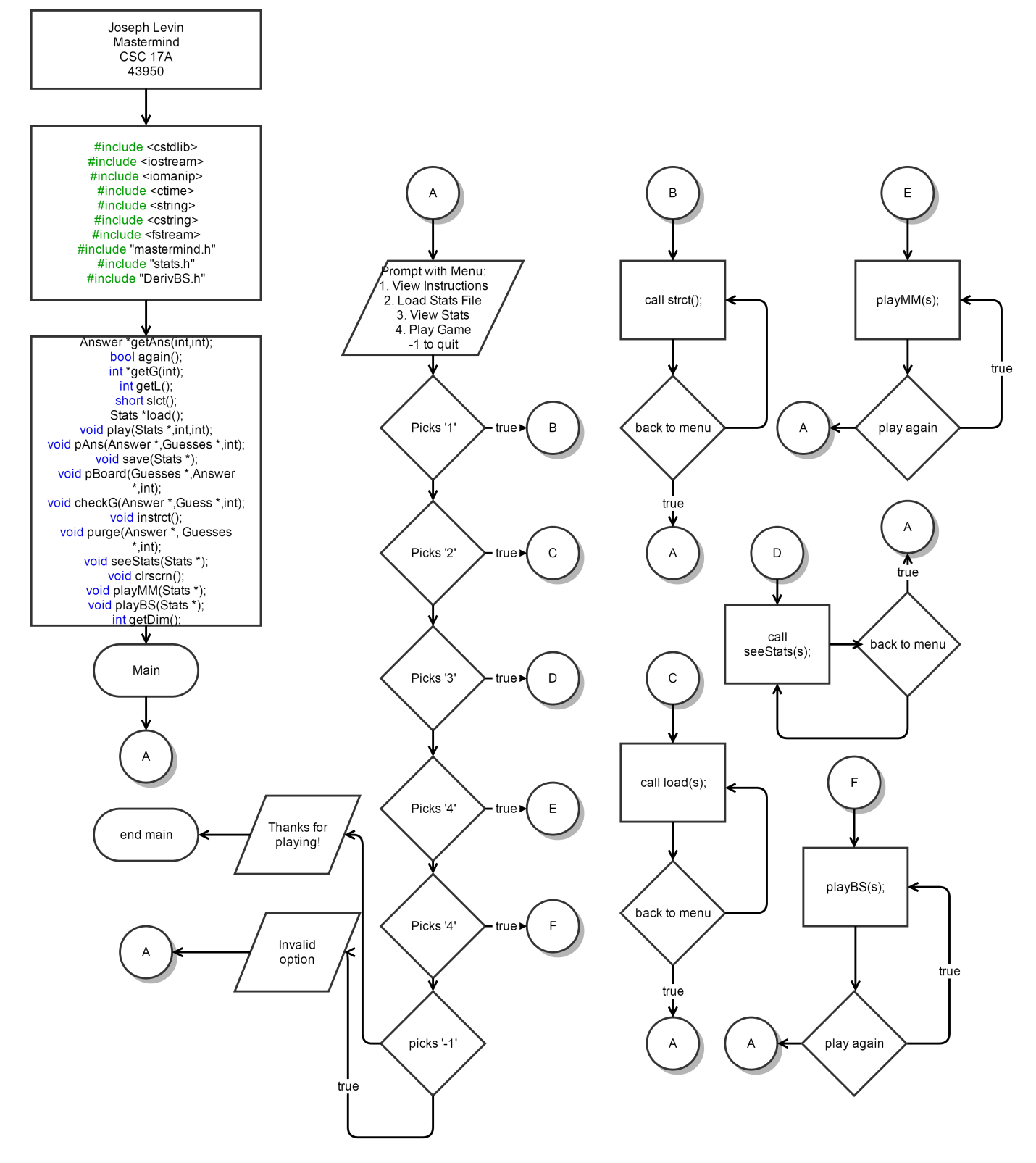
The BaseBS class contains a member function for displaying the board in a grid format. It shows the location of the ships as +, hits as X, misses as O.

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

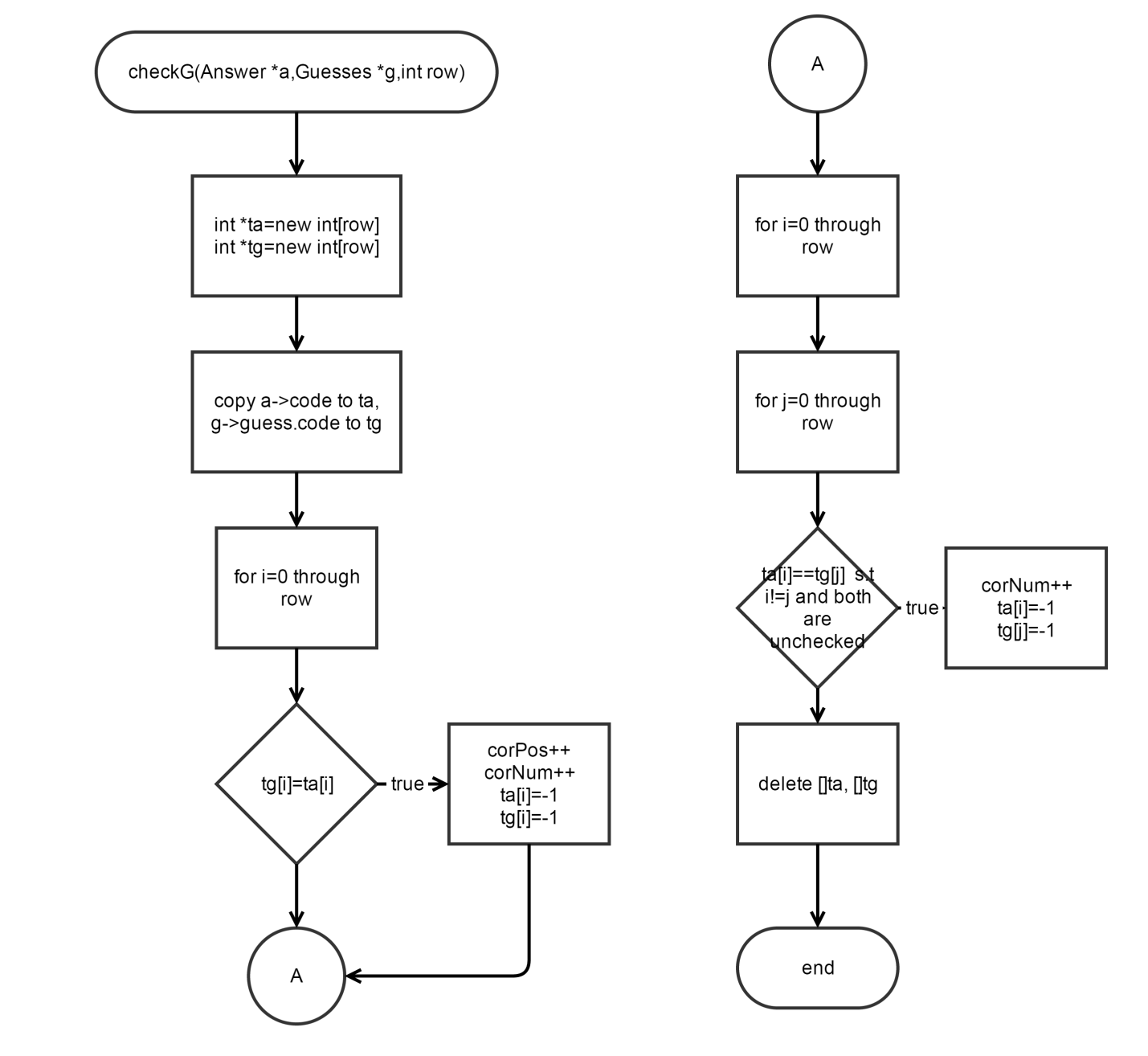
A member function is called to place the player’s ships one at time until there are ships equal to the dimension of the board.

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

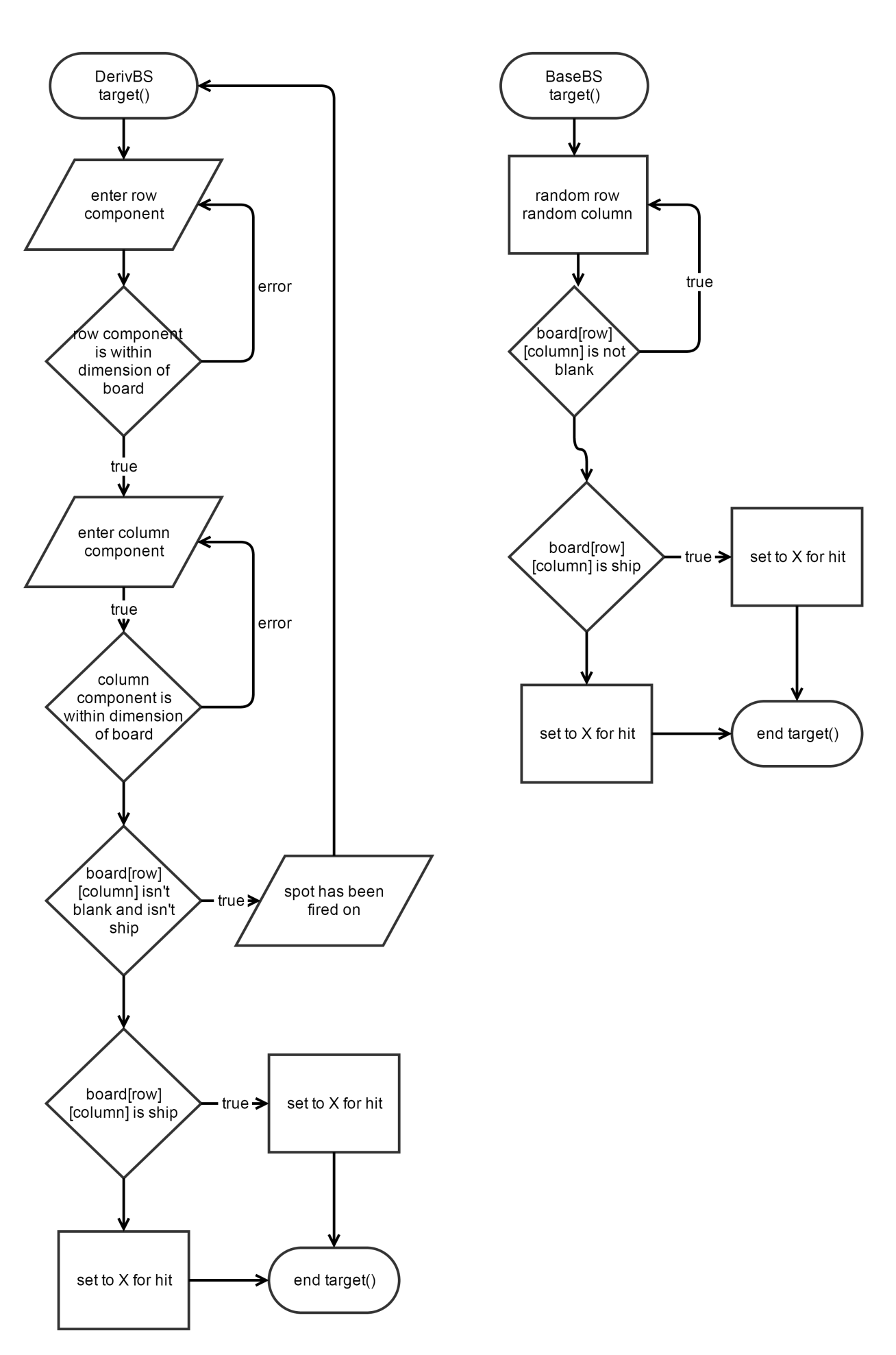
* 1. Flowcharts

Main menu:

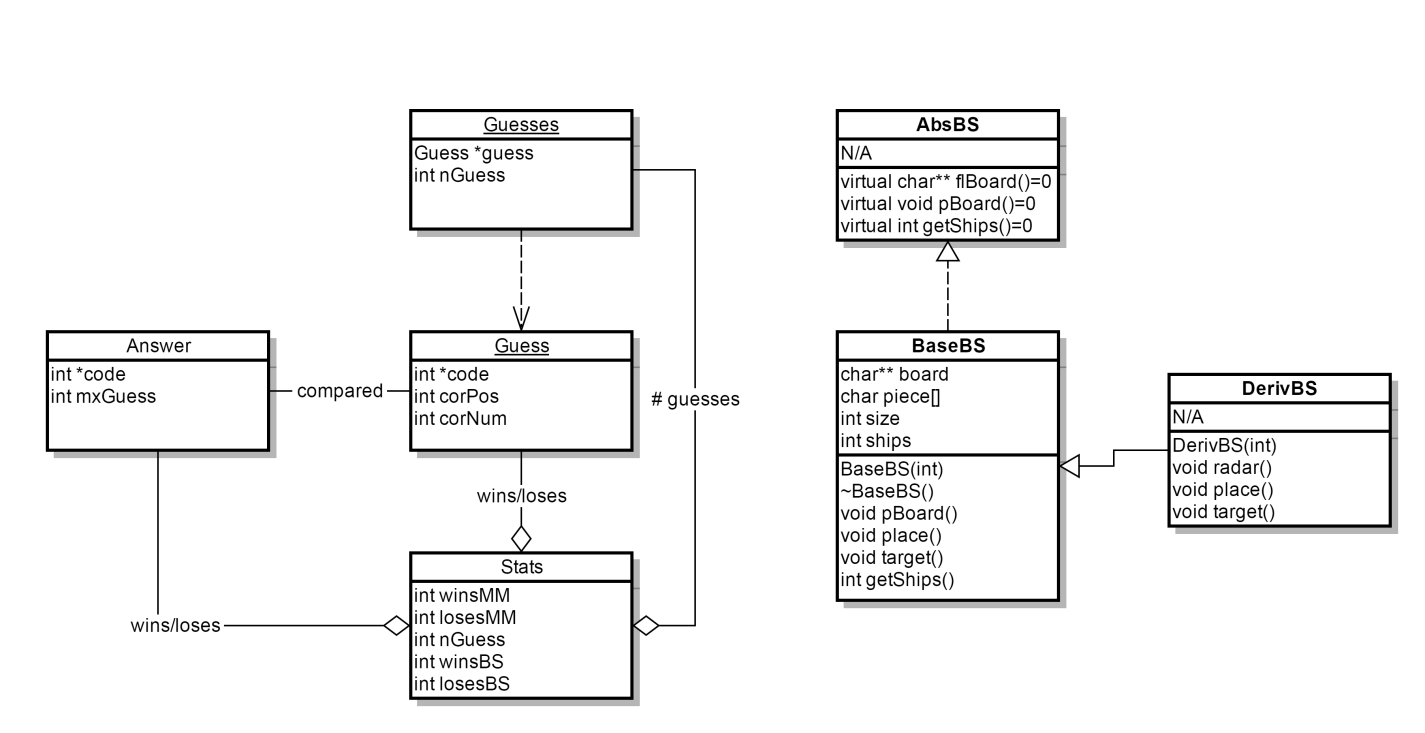
Mastermind Guess Checking Function (checkG):



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
2. /\*
3. \* File: main.cpp
4. \* Author: Joseph Levin
5. \* C++ Project 1 Mastermind – Spring 2015 43950
6. \* Created on April 26, 2015, 7:01 PM
7. \*/
8. //System Libraries
9. #include <cstdlib>
10. #include <iostream>
11. #include <iomanip>
12. #include <ctime>
13. #include <string>
14. #include <cstring>
15. #include <fstream>
16. **using** **namespace** std**;**
17. //User Libraries
18. #include "mastermind.h"
19. //Global Constants
20. //Function Prototypes
21. Answer **\***getAns**(**int**,**int**);**//generates an answer
22. bool again**();**//replay function
23. int **\***getG**(**int**);**//gets guess from user
24. int getL**();**//returns length for code combination
25. short slct**();**
26. Stats **\***load**();**//loads stats structure
27. void play**(**Stats **\*,**int**,**int**);**//launches the game
28. void pAns**(**Answer **\*,**Guesses **\*,**int**);**//prints the answer
29. void save**(**Stats **\*);**//saves the stats structure as binary file
30. void pBoard**(**Guesses **\*,**Answer **\*,**int**);**//prints the game board
31. void checkG**(**Answer **\*,**Guess **\*,**int**);**//checks guess against answer
32. void instrct**();**
33. void purge**(**Answer **\*,** Guesses **\*,**int**);**//deletes all structures related to a game
34. void seeStats**(**Stats **\*);**//Displays stats
35. void clrscrn**();**//clears screen
36. //Begin
37. int main**(**int argc**,** char**\*\*** argv**)** **{**
38. short optn**;**
39. Stats **\***s**=new** Stats**;**
40. s**->**wins**=**0**;**
41. s**->**loses**=**0**;**
42. s**->**nGuess**=**0**;**
43. **do{**
44. clrscrn**();**
45. optn**=**slct**();**
46. **switch(**optn**){**
47. **case** 1**:**
48. **do{**
49. clrscrn**();**
50. instrct**();**
51. **}while(**again**());**
52. **break;**
53. **case** 2**:**
54. s**=**load**();**
55. **break;**
56. **case** 3**:**
57. **do{**
58. clrscrn**();**
59. seeStats**(**s**);**
60. **}while(**again**());**
61. **break;**
62. **case** 4**:**
63. **do{**
64. clrscrn**();**
65. int l**=**getL**();**
66. play**(**s**,**10**,**l**);**
67. **}while(**again**());**
68. **break;**
69. **case** **-**1**:**
70. cout**<<**"Thanks for playing!"**<<**endl**;**
71. **break;**
72. **default:**
73. cout**<<**"Invalid option selected..."
74. "it's not that hard, really."**<<**endl**;**
75. **break;**
76. **}**
77. **}while(**optn**!=-**1**);**
78. //clean up!
79. **delete** s**;**
80. **return** 0**;**
81. **}**
82. //Clear screen function outputs a ton of new lines in order to clear
83. //the command prompt to look nice
84. void clrscrn**(){**
85. **for(**int i**=**0**;** i**<**100**;** i**++)**
86. cout**<<**endl**;**
87. **}**//end
88. //getAns dynamically creates a Answer struct, fills the code array with
89. //random integers 0-9, sets nGuess to max (max # of guesse) and then returns
90. Answer **\***getAns**(**int max**,** int row**){**
91. Answer **\***answer**=new** Answer**;**
92. answer**->**mxGuess**=**max**;**//max number of guesses allowed
93. answer**->**code**=new** int**[**row**];**//the combination is 'row' digits long
94. **for(**int i**=**0**;**i**<**row**;**i**++)**
95. answer**->**code**[**i**]=**rand**()%**8**+**1**;**//fill code with 1-9
96. **return** answer**;**
97. **}**//end
98. //checkG takes in an Answer struct. It copies the contents of the code array
99. //into another int array and compares each element to a Guess array
100. //It changes the elments in the temp int array to -1 as it finds matches
101. //in order to ensure no duplicate matches are found
102. //if an element in the guess matches the position and number of the answer
103. //both the correct number and correct position counters are incremented
104. //otherwise if only the number is matched then the correct number indication
105. //is incremented. Man this is a long description.
106. void checkG**(**Answer **\***a**,**Guesses **\***g**,**int row**){**
107. int **\***ta**=new** int**[**row**];**//temp array to store answer
108. int **\***tg**=new** int**[**row**];**//temp array to store guess
109. **for(**int i**=**0**;**i**<**row**;**i**++){**
110. ta**[**i**]=**a**->**code**[**i**];**//copy answer to temp answer
111. tg**[**i**]=**g**->**guess**[**g**->**nGuess**-**1**].**code**[**i**];**//copy guess to temp guess
112. **}**
113. //loop through arrays to check for both correct position and correct number
114. **for(**int i**=**0**;**i**<**row**;**i**++){**
115. **if(**tg**[**i**]==**ta**[**i**]){**
116. g**->**guess**[**g**->**nGuess**-**1**].**corPos**++;**
117. g**->**guess**[**g**->**nGuess**-**1**].**corNum**++;**
118. ta**[**i**]=-**1**;**//no duplicates
119. tg**[**i**]=-**1**;**
120. **}**
121. **}**
122. //loop through temp answer again
123. **for(**int i**=**0**;**i**<**row**;**i**++){**
124. //loop through guess
125. **for(**int j**=**0**;**j**<**row**;**j**++){**
126. //check for same number
127. **if(**ta**[**i**]==**tg**[**j**]&&**i**!=**j**&&**tg**[**j**]!=-**1**&&**ta**[**i**]!=-**1**){**//same num, diff pos
128. g**->**guess**[**g**->**nGuess**-**1**].**corNum**++;**
129. ta**[**i**]=-**1**;**
130. tg**[**j**]=-**1**;**
131. **}**
132. **}**
133. **}**
134. **delete** **[]**ta**;**
135. **delete** **[]**tg**;**
136. **}**//end
137. /\*
138. \* getG prompts the user to enter a 4 digit combination, separated by spaces
139. \* it stores it as a string, and checks if every odd element is a digit 1-8
140. \* and then checks if every even is a space
141. \* if this is true, it then converts each odd element into an int and stores
142. \* it in an int array and returns this array
143. \* takes an int to specify how big an int array to create
144. \*/
145. int **\***getG**(**int row**){**
146. //declare variables
147. int **\***temp**=new** int**[**row**];**//will be returned
148. string guess**;**//for character checking
149. bool check**;**//input validation
150. cout**<<**"Enter a "**<<**row
151. **<<**" digit combination (1-8) separated by spaces."**<<**endl**;**
152. **do{**
153. check**=true;**
154. getline**(**cin**,**guess**);**
155. **if(**guess**.**size**()!=**2**\***row**-**1**){**//row digits plus spaces between
156. cout**<<**"You must enter "**<<**row**<<**" digits separated by spaces."
157. **<<**"Try again!"**<<**endl**;**
158. check**=false;;**
159. **}**
160. **else{**
161. //every odd must be 1-8, and every even must be space
162. **for(**int i**=**0**;**i**<**2**\***row**-**1**;**i**++){**
163. //check odds for all digits
164. **if(**i**%**2**==**0**&&!**isdigit**(**guess**[**i**])){**
165. check**=false;**
166. cout**<<**"Please only enter digits. Try again!"**<<**endl**;**
167. **}**
168. //check if all digits are 1-8
169. **else** **if(**i**%**2**==**0**&&(**atoi**(&**guess**[**i**])>**8**||**atoi**(&**guess**[**i**])<**1**)){**
170. check**=false;**
171. cout**<<**"Digits must all be between 1 & 8. Try again!"**<<**endl**;**
172. **break;**
173. **}**
174. //check if every even is space
175. **else{**
176. **if(!**isspace**(**guess**[**i**])&&!**isdigit**(**guess**[**i**])){**
177. check**=false;**
178. cout**<<**"Every digit must be "
179. **<<**"separated by a space. Try again!"**<<**endl**;**
180. **}**
181. **}**
182. **}**
183. **}**
184. **}while(!**check**);**
185. //copy digits into int array to return
186. **for(**int i**=**0**;**i**<**row**;**i**++)**
187. temp**[**i**]=**atoi**(&**guess**[**2**\***i**]);**
188. //int array ready to return
189. **return** temp**;**
190. **}**//end
191. /\*
192. \* pBoard takes in pointers to an Answer and Guess, and an int
193. \* specifying the code length. It prints out dashed lines indicating
194. \* how many guesses remain and also prints out all previous guesses
195. \*/
196. void pBoard**(**Guesses **\***g**,** Answer **\***a**,** int r**){**
197. //X's represent the mystery code
198. **for(**int i**=**0**;**i**<**r**;**i**++)**
199. cout**<<**"X "**;**
200. cout**<<**endl**;**
201. //-'s represent spaces left for remaining guesses
202. **for(**int i**=**0**;**i**<(**a**->**mxGuess**)-(**g**->**nGuess**)+**1**;**i**++){**
203. **for(**int j**=**0**;**j**<**r**;**j**++)**
204. cout**<<**"- "**;**
205. cout**<<**endl**;**
206. **}**
207. //all previous guesses
208. **if(**g**->**nGuess**-**1**!=**0**){**
209. **for(**int i**=**g**->**nGuess**-**2**;**i**>=**0**;**i**--){**
210. **for(**int j**=**0**;**j**<**r**;**j**++)**
211. cout**<<**g**->**guess**[**i**].**code**[**j**]<<**" "**;**
212. cout**<<**" N:"**<<**g**->**guess**[**i**].**corNum
213. **<<**" P:"**<<**g**->**guess**[**i**].**corPos**<<**endl**;**
214. **}**
215. **}**
216. //finished
217. **}**//end
218. /\*
219. \* getL prompts the user to enter either 4, 6, or 8. It then returns
220. \* the value selected as an int. No parameters
221. \*/
222. int getL**(){**
223. int l**;**//length of code
224. bool check**=false;**//error checking flag
225. cout **<<**"Please the code length!"**<<**endl**<<**"Options: 4, 6, 8"**<<**endl**;**
226. **do{**
227. cin**>>**l**;**
228. **if(**cin**.**fail**()||(**l**!=**4**&&**l**!=**6**&&**l**!=**8**)){**
229. cin**.**clear**();**
230. cin**.**ignore**(**256**,**'\n'**);**
231. cout**<<**"Sorry! You have to choose "
232. "either 4, 6, or 8. Try again!"**<<**endl**;**
233. **}**
234. **else**
235. check**=true;**
236. **}while(!**check**);**
237. **return** l**;**
238. **}**//end
239. /\*
240. \* pAns prints the answer code
241. \*/
242. void pAns**(**Answer **\***a**,**Guesses **\***g**,**int r**){**
243. //Show Answer
244. **for(**int i**=**0**;**i**<**r**;**i**++)**
245. cout**<<**a**->**code**[**i**]<<**" "**;**
247. cout**<<**" <- Answer!"**<<**endl**;**
248. //-'s represent spaces left for remaining guesses
249. **for(**int i**=**0**;**i**<(**a**->**mxGuess**)-(**g**->**nGuess**);**i**++){**
250. **for(**int j**=**0**;**j**<**r**;**j**++)**
251. cout**<<**"- "**;**
252. cout**<<**endl**;**
253. **}**
254. //all previous guesses
255. **if(**g**->**nGuess**!=**0**){**
256. **for(**int i**=**g**->**nGuess**-**1**;**i**>=**0**;**i**--){**
257. **for(**int j**=**0**;**j**<**r**;**j**++)**
258. cout**<<**g**->**guess**[**i**].**code**[**j**]<<**" "**;**
259. cout**<<**" N:"**<<**g**->**guess**[**i**].**corNum
260. **<<**" P:"**<<**g**->**guess**[**i**].**corPos**<<**endl**;**
261. **}**
262. **}**
263. **}**//end
264. /\*
265. \* play is the main driver for Mastermind gameplay. It handles turn taking,
266. \* win/lose checks, as well as stat saving. Void function, takes in an integer
267. \* to determine max # of guesses and an integer to determine code length
268. \*/
269. void play**(**Stats **\***s**,**int m**,**int r**){**
270. //generate answer
271. cin**.**clear**();**
272. cin**.**ignore**(**265**,**'\n'**);**
273. char optn**;**
274. Answer **\***a**=**getAns**(**m**,**r**);**
275. //generate and prepare Guesses
276. Guesses **\***g**=new** Guesses**;**
277. g**->**nGuess**=**0**;**
278. g**->**guess**=new** Guess**[**m**];**
279. **for(**int i**=**0**;**i**<**m**;**i**++){**
280. g**->**guess**[**i**].**corNum**=**0**;**
281. g**->**guess**[**i**].**corPos**=**0**;**
282. **}**
283. //loop until win/lose
284. **do{**
285. clrscrn**();**
286. g**->**nGuess**++;**
287. pBoard**(**g**,**a**,**r**);**
288. g**->**guess**[**g**->**nGuess**-**1**].**code**=**getG**(**r**);**
289. checkG**(**a**,**g**,**r**);**
290. **}while(**g**->**nGuess**<**a**->**mxGuess**&&**g**->**guess**[**g**->**nGuess**-**1**].**corPos**!=**r**);**
291. clrscrn**();**
292. pAns**(**a**,**g**,**r**);**
293. **if(**g**->**guess**[**g**->**nGuess**-**1**].**corPos**==**r**){**
294. cout**<<**"You cracked the code! You win!"**<<**endl**;**
295. s**->**wins**++;**
296. **}**
297. **else{**
298. cout**<<**"You didn't crack the code! You lose... Sorry!"**<<**endl**;**
299. s**->**loses**++;**
300. **}**
301. s**->**nGuess**+=**g**->**nGuess**;**
302. cout**<<**"Would you like to save your stats? y/n"**<<**endl**;**
303. **do{**
304. cin**>>**optn**;**
305. **if(**tolower**(**optn**)!=**'y'**&&**tolower**(**optn**)!=**'n'**)**
306. cout**<<**"Sorry, that's not a valid option. Please try again."**<<**endl**;**
307. **else** **if(**tolower**(**optn**)==**'y'**){**
308. save**(**s**);**
309. cout**<<**"Stats saved!"**<<**endl**;**
310. **}**
311. **else**
312. cout**<<**"Stats not saved."**<<**endl**;**
313. **}while(**tolower**(**optn**)!=**'y'**&&**tolower**(**optn**)!=**'n'**);**
314. //clean up
315. purge**(**a**,**g**,**m**);**
316. **}**//end
317. /\*
318. \* purge takes in an Answers and Guesses pointer and deletes all dynamically
319. \* allocated elements of the struct, as well as the structs themselves
320. \* and then points them to nullptrs
321. \*/
322. void purge**(**Answer **\***a**,**Guesses **\***g**,**int m**){**
323. **delete** **[]**a**->**code**;**
324. a**->**code**=NULL;**
325. **delete** a**;**
326. **for(**int i**=**0**;**i**<**m**;**i**++){**
327. **delete** **[]**g**->**guess**[**i**].**code**;**
328. g**->**guess**[**i**].**code**=NULL;**
329. **}**
330. **delete** g**->**guess**;**
331. g**->**guess**=NULL;**
332. **delete** g**;**
333. g**=NULL;**
334. **}**
335. /\*
336. \* save takes in a Stats struct pointer. It prompts the user for a name
337. \* to store the stats struct under, and then writes the contents of the Stats
338. \* to a binary file. Returns void
339. \*/
340. void save**(**Stats **\***s**){**
341. cin**.**clear**();**
342. cin**.**ignore**(**256**,**'\n'**);**
343. ofstream out**;**//file stream
344. cout**<<**"Enter the name to store the stats under"**<<**endl**;**
345. string name**;**
346. getline**(**cin**,**name**);**
347. out**.**open**(**name**.**c\_str**(),**ios**::**binary**);**
348. out**.**write**(reinterpret\_cast<**char **\*>(**s**),sizeof(**Stats**));**
349. out**.**close**();**
350. **}**//end
351. /\*
352. \* load prompts the user for a name, and attempts to open a file
353. \* with that name. If found, it reads the contents into a Stats structure
354. \* and returns it.
355. \*/
356. Stats **\***load**(){**
357. cin**.**clear**();**
358. cin**.**ignore**(**256**,**'\n'**);**
359. string name**;**
360. ifstream in**;**
361. cout**<<**"Enter the name of the person whose stats to load"**<<**endl**;**
362. **do{**
363. getline**(**cin**,**name**);**
364. in**.**open**(**name**.**c\_str**(),**ios**::**binary**);**
365. **if(**in**.**fail**()){**
366. cout**<<**"Name not found. Try again!"**<<**endl**;**
367. **}**
368. **else{**
369. Stats **\***s**=new** Stats**;**
370. in**.**read**(reinterpret\_cast<**char **\*>(**s**),sizeof(**Stats**));**
371. in**.**close**();**
372. **return** s**;**
373. **}**
374. **}while(**in**.**fail**());**
375. **}**
376. //slct serves to take in input for menu selection, performs error checks
377. //and then returns the value if it passes checks
378. short slct**(){**
379. short pick**;** //for menu selection
380. bool check**=false;**
381. cout**<<**"Welcome to Mastermind!"**<<**endl**;**
382. cout**<<**"Choose an option from the menu: "**<<**endl
383. **<<**"1. View Instructions"**<<**endl
384. **<<**"2. Load Stats File"**<<**endl
385. **<<**"3. View Stats"**<<**endl
386. **<<**"4. Play Game!"**<<**endl
387. **<<**"-1 to quit"**<<**endl**;**
388. **do{**
389. cin**>>**pick**;**
390. **if(**cin**.**fail**()||**pick**<=**0**&&**pick**!=-**1**||**pick**>**7**){**//error checking
391. cin**.**clear**();**
392. cin**.**ignore**(**256**,**'\n'**);**
393. cout**<<**"Error. Invalid selection. Try again."**<<**endl**;**
394. **}**
395. **else**
396. check**=true;**//valid input
397. **}while(!**check**);**
398. **return** pick**;**
399. **}**
400. /\*
401. \* again asks the user if they would like to play another game
402. \* it returns true if the user does, false if they do not
403. \*/
404. bool again**(){**
405. bool check**=false;**
406. char pick**;**
407. cout**<<**"Would you like to play again? y/n"**<<**endl**;**
408. **do{**
409. cin**>>**pick**;**
410. **if(**cin**.**fail**()||**tolower**(**pick**)!=**'y'**&&**tolower**(**pick**)!=**'n'**){**//only accepts
411. cin**.**clear**();** //y or n as input
412. cin**.**ignore**(**256**,**'\n'**);**
413. cout**<<**"Error. Invalid selection. Try again."**<<**endl**;**
414. **}**
415. **else** **if(**tolower**(**pick**)==**'y'**){**//user wants to repeat
416. check**=true;**
417. cin**.**clear**();**
418. cin**.**ignore**(**256**,**'\n'**);**
419. **return** **true;**
420. **}**
421. **else{** //user does not want to repeat
422. cin**.**clear**();**
423. cin**.**ignore**(**256**,**'\n'**);**
424. check**=true;**
425. **return** **false;**
426. **}**
427. **}while(!**check**);**
429. **}**//end
430. /\*
431. \* seeStats takes in Stats pointer. It displays the elements within,
432. \* (wins/loses/total guesses) and also calculates win percentage, and
433. \* correct guess percentage
434. \*/
435. void seeStats**(**Stats **\***s**){**
436. **if(**s**->**nGuess**!=**0**){**
437. cout**<<**"Games played: "**<<**s**->**wins**+**s**->**loses**<<**endl**;**
438. cout**<<**"Wins: "**<<**s**->**wins**<<**endl**;**
439. cout**<<**"Losses: "**<<**s**->**loses**<<**endl**;**
440. cout**<<**"Total guesses: "**<<**s**->**nGuess**<<**endl**;**
441. cout**<<**setprecision**(**2**)<<**fixed
442. **<<**"% Games won: "**<<**100**\*static\_cast<**float**>(**s**->**wins**)/**
443. **(static\_cast<**float**>(**s**->**wins**)+static\_cast<**float**>(**s**->**loses**))<<**endl**;**
444. cout**<<**setprecision**(**5**)<<**fixed
445. **<<**"% Guesses correct: "**<<**100**\*static\_cast<**float**>(**s**->**wins**)/**
446. **static\_cast<**float**>(**s**->**nGuess**)<<**endl**;**
447. **}**
448. **else**
449. cout**<<**"Stats File is empty!"**<<**endl**;**
450. **}**
451. /\*
452. \* instrct displays the rules for Code Breaker (based on Mastermind)
453. \*/
454. void instrct**(){**
455. cout**<<**"Mastermind is a game of decryption!"**<<**endl**<<**endl
456. **<<**"In it, the player (that's you!)"**<<**endl
457. **<<**"attempts to decipher a secret code generated at random."**<<**endl
458. **<<**"The code consists of integers between 1 and 8."**<<**endl
459. **<<**"The player chooses from 3 code lengths (4, 6, 8),"**<<**endl
460. **<<**"and guesses at each of the individual digits that make up"
461. " the secret code."**<<**endl
462. **<<**"After each attempt the player is told how "
463. "many correct numbers"**<<**endl
464. **<<**"(shown as 'N') and correct position "
465. "(shown as 'P') the attempt had."**<<**endl
466. **<<**"The player wins by correctly determining"
467. " all 4 correct numbers and positions."**<<**endl
468. **<<**"Remember, a guess can have 4 correct numbers but 0 correct "
469. "positions,"**<<**endl
470. **<<**"but you can't have a correct position without"
471. " it also being the correct number."**<<**endl
472. **<<**endl**<<**"Good luck!"**<<**endl**;**
473. **}**