Project 1: Mastermind

Documentation

Lucas Banks

CSC 5 48102

Dr. Mark Lehr

10/28/16

Table Of Contents

Summary

This program runs a text based game called Mastermind. In this game, there will be a board with 4 spots. Each spot on the board has a letter assigned to it. The goal of the game is for the user to figure out in what order the letters are assigned to the board.

There are 4 difficulty modes. Easy, intermediate, hard, and beginner. Beginner mode is for new players to help them get a grasp of the game. At that difficulty, there are 4 letter choices, which also means that there are only 4 possible letters that could take up any given spot on the board. As the user guesses the order of the letters, the program will tell you exactly which spots are right, and which spots are wrong. The user can only guess 4 letters at a time with no repeats, which is a rule that applies to all difficulties.

In Easy mode and up, the program won't tell you exactly which spots are right or wrong. Instead, it will tell you if you have a letter on the right spot, and if you have a letter that exists somewhere on the board, but not on the right spot. It won't tell you what order. In Easy mode, there are 6 possible letter choices. In intermediate, there are 7, and Hard, 8. There are only 4 board spots for each mode.

Variables/Constants Used

************************ GLOBAL ************************

Strings:

begnner,
easy,
interm,
hard;

************************ Int Main() ************************

Strings:

guess,
spot,
chkSpt,
again=,
choices,
dif,
accura=(4, " ");

Int:

numOf,
oCount,
oOCount;

Float:

score;

Bool:

win,
quit;

******************* String setBoard() *******************

Strings:

spot,
user,
choices,
chkSpot;

Int:

boardSet,
repeat;

Bool:

exists;

******************* String checkInput() *******************

Strings:

vldList,
check,
repeat,
guess;

Bool:

quit,
check;

****************** Void printInstructions() ******************

Strings:

cont,
Contents;

Ifstream:

textFile;

Flowcharts!

Int main(); & Global https://www.gliffy.com/go/share/sw6fk0l5epuqboen7mtb

setBoard(); https://www.gliffy.com/go/share/sxqd7drg8mvwu0hx5afp

checkInput(); https://www.gliffy.com/go/share/sjxmw6i3vujihhmphked

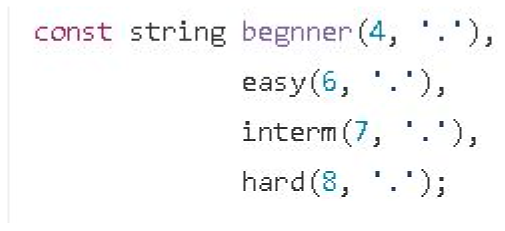
printInstructions(); https://www.gliffy.com/go/share/sn49xlkwubgeydx49wkb

*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*

*

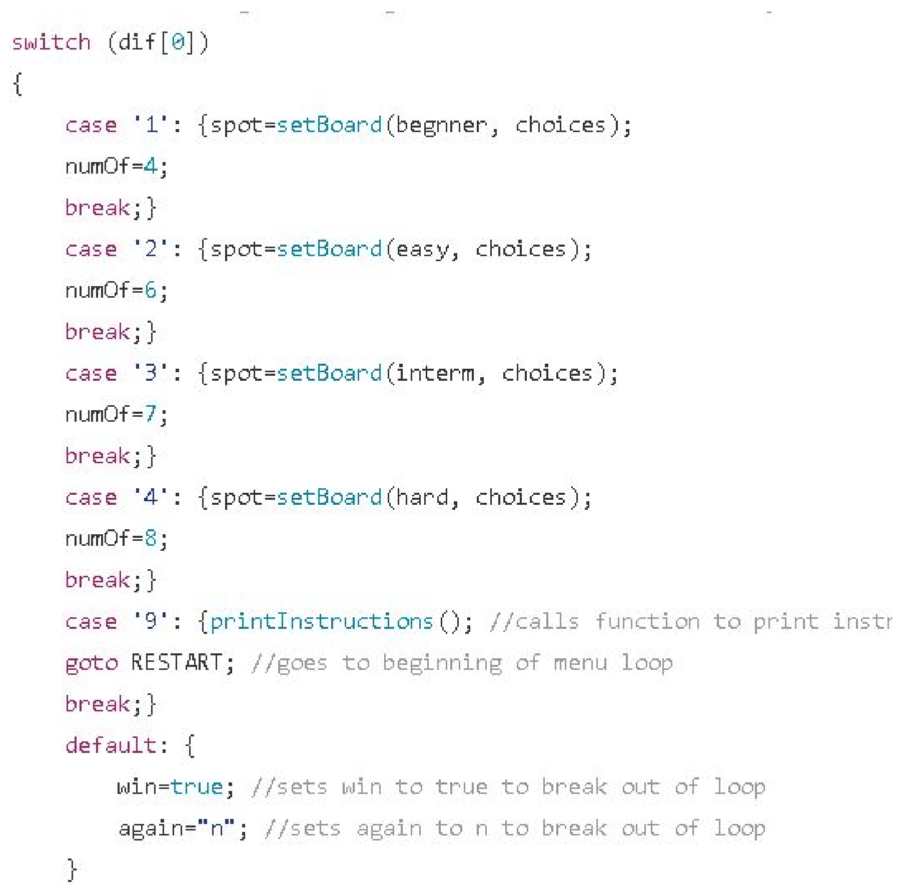
Developments

One of my main goals was to get the different difficulty settings working without having to copy and past huge blocks of code over and over again. To do this, I had to pass these values into the setBoard function.



```
const string begnner(4, '.'),
             easy(6, '.'),
             interm(7, '.'),
             hard(8, '.');
```

Whatever difficulty the user ended up picking would determine which string constant would be passed into the function.



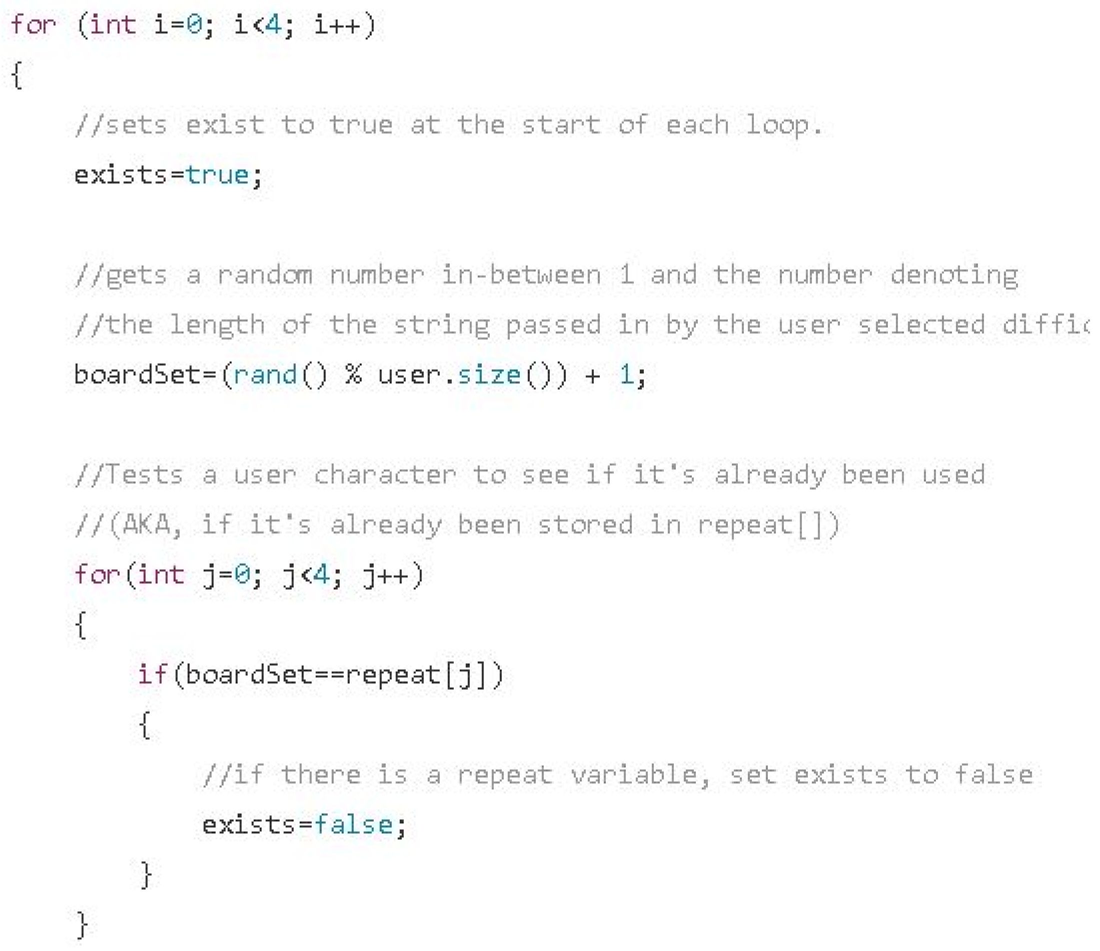
```
switch (dif[0])
{
    case '1': {spot=setBoard(begnner, choices);
    numOf=4;
    break;}
    case '2': {spot=setBoard(easy, choices);
    numOf=6;
    break;}
    case '3': {spot=setBoard(interm, choices);
    numOf=7;
    break;}
    case '4': {spot=setBoard(hard, choices);
    numOf=8;
    break;}
    case '9': {printInstructions(); //calls function to print instr
    goto RESTART; //goes to beginning of menu loop
    break;}
    default: {
        win=true; //sets win to true to break out of loop
        again="n"; //sets again to n to break out of loop
    }
```

The size of those strings are crucial to the difficulty settings.

```
string setBoard(string user, string &choices){
    string spot(user.size(),'.'), //where letters will be randomly assigned
           chkSpot; //checks how accurate the player's guess is
    int boardSet, //where random number assignment (1-4) will be stored
        repeat[user.size()]={}; //tests for number repetition
    bool exists=false;
    srand(time(NULL));

    //Will assign list of valid choices depending on what difficulty the user
    //chooses
    switch(user.size())
    {
        case 4: {choices="Q W A S"; break;}
        case 6: {choices="Q W A S E D"; break;}
        case 7: {choices="Q W A S E D R"; break;}
        case 8: {choices="Q W A S E D R F"; break;}
    }
```

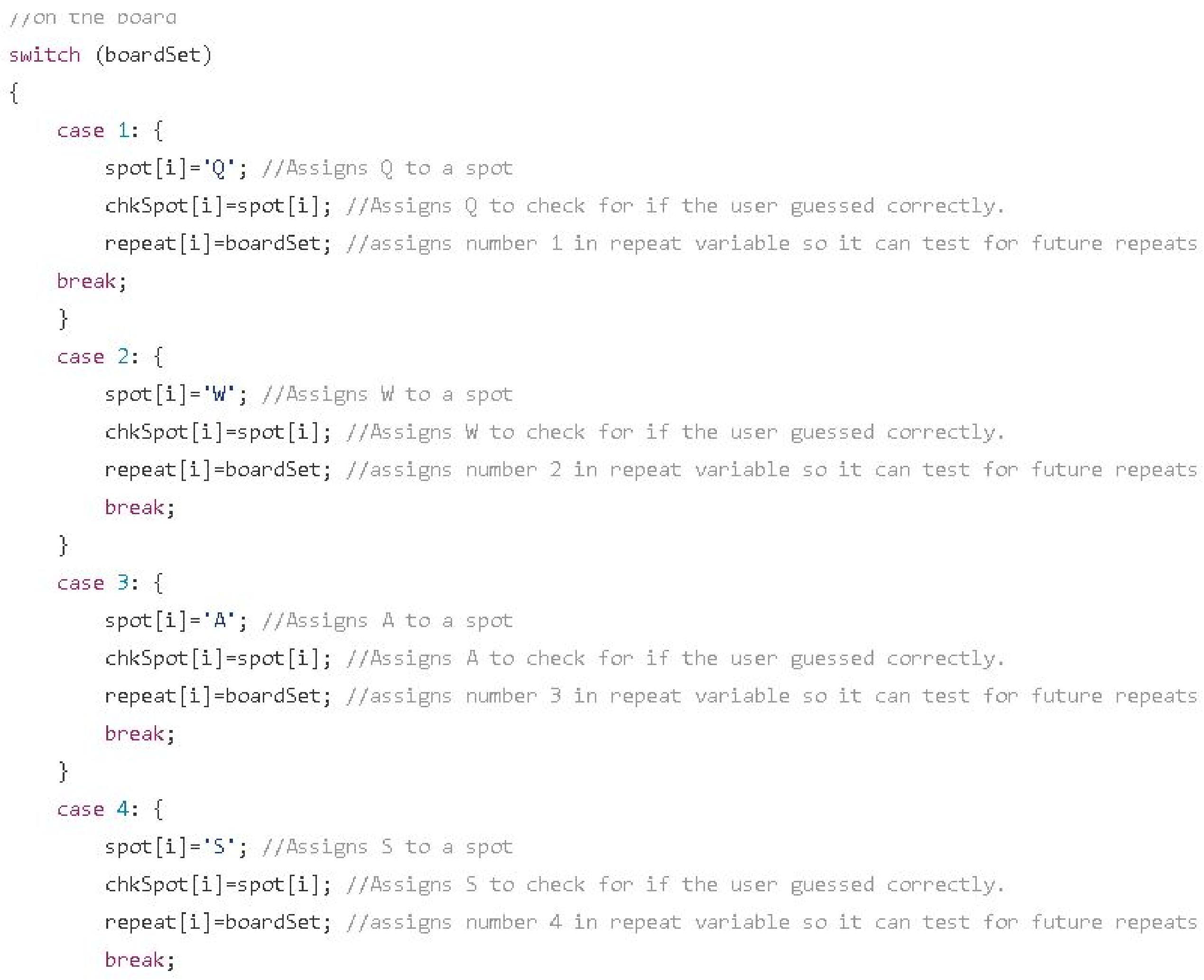
The string value would be passed into the variable “user” which is defined in the (). Then I’d use the .size() operator to get the size of string. Depending on what size it was, the switch statement would choose the appropriate list of choices which are pertinent to the multiple difficulty settings.

To make sure the program only chose the appropriate letters for each difficulty to assign to each board space without repeats, I would have it pick a random number in between 1, and the size of the strong that was passed in.

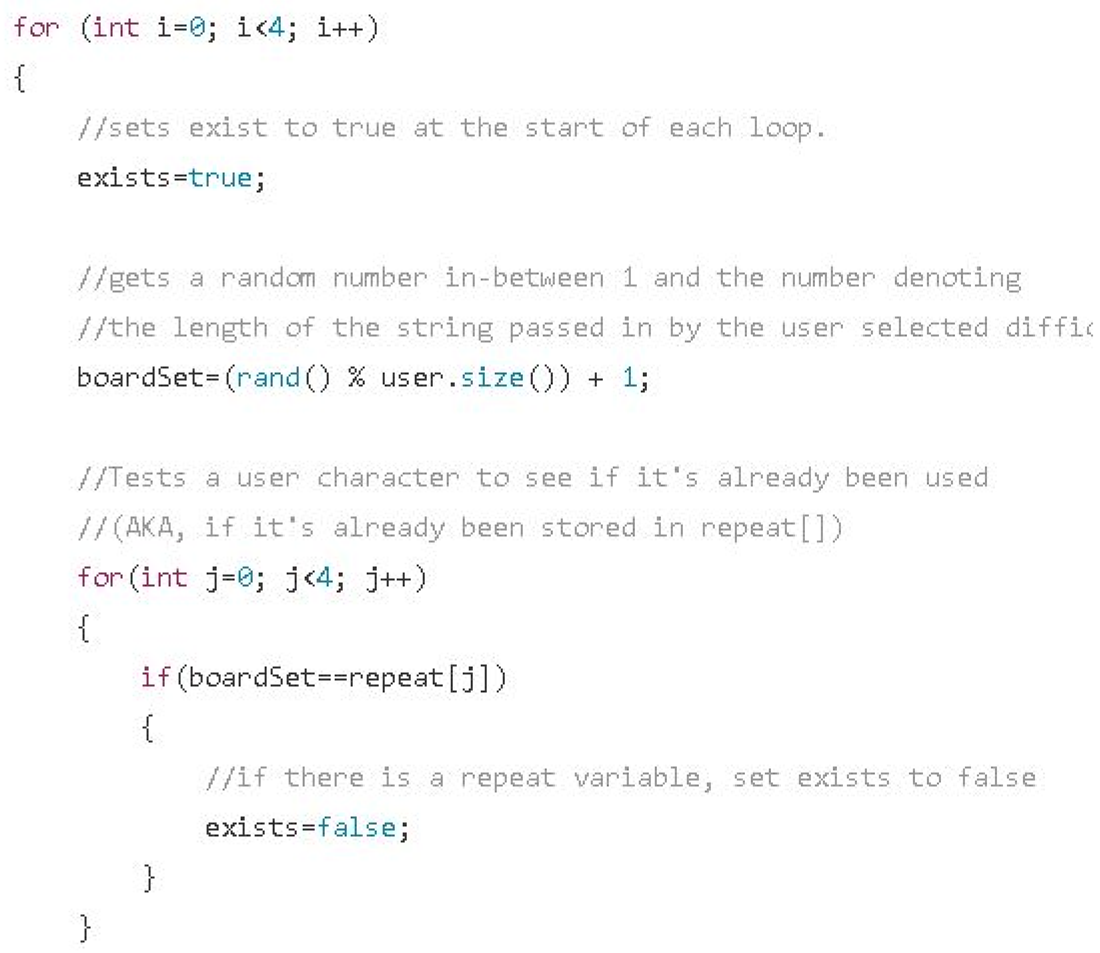


```
for (int i=0; i<4; i++)
{
    //sets exist to true at the start of each loop.
    exists=true;

    //gets a random number in-between 1 and the number denoting
    //the length of the string passed in by the user selected diffic
    boardSet=(rand() % user.size()) + 1;

    //Tests a user character to see if it's already been used
    //(AKA, if it's already been stored in repeat[])
    for(int j=0; j<4; j++)
    {
        if(boardSet==repeat[j])
        {
            //if there is a repeat variable, set exists to false
            exists=false;
        }
    }
```

It will loop 4 times to fill up four board spots with a letter. As you can see in the below image, depending on what number is chosen, a letter will be assigned. So if the user chooses beginner mode where there is only 4 letter choices, it will only pick from 4 letters. Also, it stores each letter in repeat[i], which is a variable that will be used to store past assigned values. That will be used for checking for repeats.

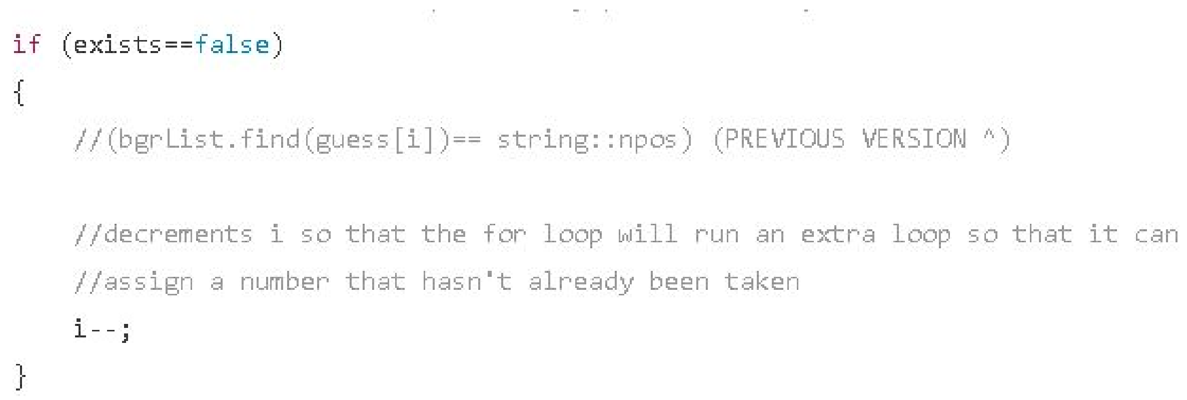


```
//on the board
switch (boardSet)
{
    case 1: {
        spot[i]='Q'; //Assigns Q to a spot
        chkSpot[i]=spot[i]; //Assigns Q to check for if the user guessed correctly.
        repeat[i]=boardSet; //assigns number 1 in repeat variable so it can test for future repeats
    break;
    }
    case 2: {
        spot[i]='W'; //Assigns W to a spot
        chkSpot[i]=spot[i]; //Assigns W to check for if the user guessed correctly.
        repeat[i]=boardSet; //assigns number 2 in repeat variable so it can test for future repeats
        break;
    }
    case 3: {
        spot[i]='A'; //Assigns A to a spot
        chkSpot[i]=spot[i]; //Assigns A to check for if the user guessed correctly.
        repeat[i]=boardSet; //assigns number 3 in repeat variable so it can test for future repeats
        break;
    }
    case 4: {
        spot[i]='S'; //Assigns S to a spot
        chkSpot[i]=spot[i]; //Assigns S to check for if the user guessed correctly.
        repeat[i]=boardSet; //assigns number 4 in repeat variable so it can test for future repeats
        break;
```



```
for (int i=0; i<4; i++)
{
    //sets exist to true at the start of each loop.
    exists=true;

    //gets a random number in-between 1 and the number denoting
    //the length of the string passed in by the user selected diffi
    boardSet=(rand() % user.size()) + 1;

    //Tests a user character to see if it's already been used
    //(AKA, if it's already been stored in repeat[])
    for(int j=0; j<4; j++)
    {
        if(boardSet==repeat[j])
        {
            //if there is a repeat variable, set exists to false
            exists=false;
        }
    }
```

In the lower for loop of this image here, if a number is = to anything stored in the repeat function, it will turn exists to false.



```
if (exists==false)
{
    //(bgrList.find(guess[i])== string::npos) (PREVIOUS VERSION ^)

    //decrements i so that the for loop will run an extra loop so that it can
    //assign a number that hasn't already been taken
    i--;
}
```

If exists is false, not only will nothing be stored a board space, but it will decrement i so that the loop will run an extra time.

CODE

```
#include <iostream> //input/output objects
#include <string> //enables string data type
#include <cstdlib> //stadndard library

#include <ctime> //enables rand

#include <iomanip> //enables stream manipulation

#include <fstream> //enables file io


using namespace std;



//passes in the appropriate values to set the board for the
difficulty
//selected by the user.

const string begnner(4, '.'),

            easy(6, '.'),

            interm(7, '.'),

            hard(8, '.');


//Function prototypes

string setBoard(string user, string &choices);

string checkInput(int, bool&);

void printInstructions();


//execution begins here

int main()

{

   string guess, //the player's guess will be stored
```

```
          spot(4,'.'), //where letters will be randomly
assigned
          chkSpot, //checks how accurate the player's guess is
          again="y", //variable that user input will determine
if he plays again
          choices, //Where list of valid choices will be stored
for the difficulty
          accuraO = (4,""), //where Os and os will be sorted
          accura=(4,""), //Where Os and os will be stored
          dif; //variable where user input to determine
difficulty will be stored
   bool win, //Keeps game in a loop until character wins.
        quit; //keeps menu in loop until user chooses to quit
   int numOf, //number of valid choices
       oCount=0, //where number of os will be stored
       oOCount=0; //where number of Os will be stored
   float score; //where score will be kept

   RESTART: //will take back to beginning of menu loop
   do{
       //sets win = false so that the game will keep looping
until player
       //wins and/or decides to quit the program.
       win=false;

       //sets quit to false so menu will keep looping until
player decides to
       //leave
       quit=false;

       //resets score
       score=100;

       //Prompts user to select difficulty or leave program.
       cout<<"Welcome to MASTERMIND!"<<endl;
```

```
        cout<<"What difficulty do you want to select?"<<endl;

        cout<<"1 = Beginner"<<endl;

        cout<<"2 = Easy"<<endl;

        cout<<"3 = Intermediate"<<endl;

        cout<<"4 = Hard"<<endl;

        cout<<"9 = INSTRUCTIONS"<<endl;

        cout<<""<<endl;

        cout<<"Input anything else to quit program."<<endl;

        getline(cin,dif);


        //Passes in strings of a length relevant to the
difficulty selected.
        switch (dif[0])

        {

            case '1': {spot=setBoard(begnner, choices);

            numOf=4;

            break;}

            case '2': {spot=setBoard(easy, choices);

            numOf=6;

            break;}

            case '3': {spot=setBoard(interm, choices);

            numOf=7;

            break;}

            case '4': {spot=setBoard(hard, choices);

            numOf=8;

            break;}

            case '9': {printInstructions(); //calls function to
print instrcutions
            goto RESTART; //goes to beginning of menu loop

            break;}

            default: {

                win=true; //sets win to true to break out of
loop
```

```
               again="n"; //sets again to n to break out of
loop
           }
   }


   do{
       //this if will skip the whole block that checks the
user's input for the
       //right letters if user decides to quit.
       if (again!="n"){
       guess=checkInput(numOf, quit);
       if(quit==true)
       {
           break;
       }


   //for loop checks each spot to see how it corresponds with
the player's
   //guess, and then will assign O or X to a variable that will
output
   //during the game so that the player can know how accurate
his guess was
       for (int j=0; j<4; j++)
       {
           (guess[j]==spot[j]) ? (chkSpot[j]='O') :
(chkSpot[j]='X');


           if(guess[j]==spot[j])
           {
               accura+="O";
               oOCount++; //increases value per every O
assigned
           }
           else if ((spot.find(guess[j])!=
string::npos)&&guess[j]!=spot[j])
           {
               accura+="o";

               oCount++; //increases value per every o assigned

           }

           else if (spot.find(guess[j])==string::npos)

           {

               accura+=" "; //puts in a blank when neither an O
or o isn't assigned
           }

       }


       //puts an amount of os corresponding to the oCount in
the front of string
       for(int i=0; i<oCount; i++)

       {

           accuraO+="o";

       }


       //after the os are put in, this does the same thing as
above, only with
       //Os. The Os will always be after the os

       for(int i=0; i<oOCount; i++)

       {

           accuraO+="O";

       }


   //if statement checks if user guessed the letters assigned
to each
   //spot on the board correctly. If the player is successful,
tell em.
   //if not, tell em.

           if
(chkSpot[0]=='O'&&chkSpot[1]=='O'&&chkSpot[2]=='O'&&chkSpot[3]=
='O')
           {

               cout<<"YOU WIN"<<endl;
               cout<<"|| [OOOO] || ";
               cout<<"== [OOOO]"<<endl;
               cout<<endl;
               //Outputs final score
               cout<<"Your final score is
"<<setprecision(2)<<score<<"!"<<endl;
               cout<<endl;
               //Sets win = true so that game loop will break
               win=true;
               cout<<"Would you like to play again on a new
board? Enter 'y' for yes, ";
               cout<<"or anything else to quit."<<endl;
               getline(cin,again);
           }
           else
           {
               //Outputs the board to show what the user got
right and the
               //list of their choices
               cout<<"||
["<<"?"<<"]["<<"?"<<"]["<<"?"<<"]["<<"?"<<"] || ";
               if(numOf!=4){
                   cout<<"== ["<<accuraO<<"]";
               }
               else
               {

                   cout<<"==
["<<chkSpot[0]<<chkSpot[1]<<chkSpot[2]<<chkSpot[3]<<"]";
               }
               cout<<"   -- CHOICES: "<<choices<<" -- Enter '1'
to return to the menu."<<endl;
           }
       //resets score, oCount, oOcount, accura & accuraO
       score/=1.2;

       oCount=0;

       oOCount=0;

       accuraO = (4, "");

       accura = (4, "");

       }

   }while(win==false);  //loop will keep players guessing until
they win.
}while(again=="y"||again=="Y"); //loop will keep players in
menu until they-
                               //decide to quit

   return 0;

}//end of main


string setBoard(string user, string &choices){

   string spot(user.size(),'.'), //where letters will be
randomly assigned
          chkSpot; //checks how accurate the player's guess is

   int boardSet, //where random number assignment (1-4) will be
stored
       repeat[user.size()]={}; //tests for number repetition

   bool exists=false;

   srand(time(NULL));


   //Will assign list of valid choices depending on what
difficulty the user
   //chooses

   switch(user.size())

   {

       case 4: {choices="Q W A S"; break;}

       case 6: {choices="Q W A S E D"; break;}

       case 7: {choices="Q W A S E D R"; break;}

       case 8: {choices="Q W A S E D R F"; break;}

   }
   //for loop for randomizing the board
   for (int i=0; i<4; i++)
   {
       //sets exist to true at the start of each loop.
       exists=true;

       //gets a random number in-between 1 and the number
denoting
       //the length of the string passed in by the user
selected difficulty
       boardSet=(rand() % user.size()) + 1;

       //Tests a user character to see if it's already been
used
       //(AKA, if it's already been stored in repeat[])
       for(int j=0; j<4; j++)
       {
           if(boardSet==repeat[j])
           {
               //if there is a repeat variable, set exists to
false
               exists=false;
           }
       }

       //first if test will test if the random number is
identical to any previous random number.
       //if the random number IS equal to any previous ones, it
will decrement 'i'
       if (exists==false)
       {
           //(bgrList.find(guess[i])== string::npos) (PREVIOUS
VERSION ^)

           //decrements i so that the for loop will run an
extra loop so that it can
```

```
            //assign a number that hasn't already been taken
            i--;
        }
        //if the random number is unique, it assigns it to a
spot on the board
        else{
        //each number 1-4 corresponds to a letter that will be
assigned to a spot
        //on the board
        switch (boardSet)
        {
            case 1: {
                spot[i]='Q'; //Assigns Q to a spot
                chkSpot[i]=spot[i]; //Assigns Q to check for if
the user guessed correctly.
                repeat[i]=boardSet; //assigns number 1 in repeat
variable so it can test for future repeats
            break;
            }
            case 2: {
                spot[i]='W'; //Assigns W to a spot
                chkSpot[i]=spot[i]; //Assigns W to check for if
the user guessed correctly.
                repeat[i]=boardSet; //assigns number 2 in repeat
variable so it can test for future repeats
                break;
            }
            case 3: {
                spot[i]='A'; //Assigns A to a spot
                chkSpot[i]=spot[i]; //Assigns A to check for if
the user guessed correctly.
                repeat[i]=boardSet; //assigns number 3 in repeat
variable so it can test for future repeats
                break;
            }
            case 4: {
                spot[i]='S'; //Assigns S to a spot
                chkSpot[i]=spot[i]; //Assigns S to check for if
the user guessed correctly.
                repeat[i]=boardSet; //assigns number 4 in repeat
variable so it can test for future repeats
                break;
            }
            case 5: {
                spot[i]='E'; //Assigns E to a spot
                chkSpot[i]=spot[i]; //Assigns E to check for if
the user guessed correctly.
                repeat[i]=boardSet; //assigns number 5 in repeat
variable so it can test for future repeats
                break;
            }
            case 6: {
                spot[i]='D'; //Assigns D to a spot
                chkSpot[i]=spot[i]; //Assigns D to check for if
the user guessed correctly.
                repeat[i]=boardSet; //assigns number 6 in repeat
variable so it can test for future repeats
                break;
            }
            case 7: {
                spot[i]='R'; //Assigns R to a spot
                chkSpot[i]=spot[i]; //Assigns R to check for if
the user guessed correctly.
                repeat[i]=boardSet; //assigns number 7 in repeat
variable so it can test for future repeats
                break;
            }
            case 8: {
                spot[i]='F'; //Assigns F to a spot
                chkSpot[i]=spot[i]; //Assigns green to check for
if the user guessed correctly.
                repeat[i]=boardSet; //assigns number 8 in repeat
variable so it can test for future repeats
```

```
                break;

            }

        }//ends switch

        }//ends else statement


    }//ends for loop

    //outputs the board

    cout<<"|| ["<<"?"<<"]["<<"?"<<"]["<<"?"<<"]["<<"?"<<"] || ";

    cout<<"== [????]";

    cout<<"    -- CHOICES: "<<choices<<" -- Enter '1' to return
to the menu."<<endl;


    return spot; //returns the string which holds the positions
of the letters
                //on the board.

}


string checkInput(int num, bool &quit)

{

    bool check; //variable to check if input is valid

    string guess, //user's guess

           vldList, //where valid list of characters will be
stored.
           repeat=(4, "");

    //assigns the string of valid characters to a variable that
will be searched
    //when it comes to checking if the user input is valid

    switch(num)

    {

        case 4: {vldList="QWASqwas1"; break;}

        case 6: {vldList="QWASEDqwased1"; break;}

        case 7: {vldList="QWASEDRqwasedr1"; break;}

        case 8: {vldList="QWASEDRFqwasedrf1"; break;}

    }
    do{
            //if nothing is invalid about the user input, this
bool allows
            //the program to pass through the loop
            check=true;

            //Prompts user to enter guess
            cout<<"Enter: ";
            getline(cin,guess);

            if(guess=="1")
            {
                quit=true;
            }

             //Checks for valid input
            while((guess.size()!=4)&&(guess!="1"))
            {
                cout<<"Invalid input. Please type in the four
valid letters only. NO REPEATS"<<endl;
                getline(cin,guess);
            }

            //converts lowercase inputs to uppercase
        for(int u=0; u<4; u++){
            switch(guess[u])
            {
            case 'q': {guess[u]='Q'; break;}
            case 'w': {guess[u]='W'; break;}
            case 'a': {guess[u]='A'; break;}
            case 's': {guess[u]='S'; break;}
            case 'e': {guess[u]='E'; break;}
            case 'd': {guess[u]='D'; break;}

            case 'r': {guess[u]='R'; break;}

            case 'f': {guess[u]='F'; break;}

            }

        }


            //Goes through the user inputted characters one by
one, searches
            //to see if any character doesn't match up with the
valid list of
            //characters.

            for(unsigned int i=0; i<guess.size(); i++)

            {

                if (vldList.find(guess[i])== string::npos)

                {

                    check=false;

                }

            }


            //check = false if user types in 2 of the same chars

            for (unsigned i=0; i<guess.size()-1; i++)

            {   if (guess.find(guess[i], i+1) != string::npos)

                check=false;

            }





            //if any erroneous character was found in previous
for loop, this while
            //will run giving the player an invalid input
message
            if(check==false)

            {

                cout<<"Invalid input. Please type in the four
valid letters only. NO REPEATS"<<endl;
```

```
           }



       }while(check==false);  //will loop if the input was
invalid
       return guess; //returns the user's BACKGROUND CHECKED
guess
}



void printInstructions(){

   string cont,    //pauses after instructions print until user
inputs anything
          contents; //where file data will be stored

   ifstream textFile;


   //opens file

   textFile.open("instructions.txt");

   if (textFile) { //If file opened successfully, continue with
what you wanna do
       while (textFile >> contents) { //While there is MORE
input to read from the file, do whatever
           cout << contents <<" "; //display data from file

       }

   } else cout << "Error opening file."; //If it doesn't open,
display error


   //close and clear file

   textFile.close();

   textFile.clear();

   cout<<endl;

   cout<<endl;

   //waits for user input to continue back to menu

   cout<<"Enter anything to continue: ";

   getline(cin,cont);
```

```
    return;
}
```