**Project Specification**

The specifications of the project were as follows:

* The player starts a new game and is presented with a word to be guessed.
* Through key presses, guesses can be made as to which letters are contained in the word.
* Once all letters have been guessed, the player has won the game.
* The player can make ten wrong guesses before the game is over.
* The player’s score is entered into a scoreboard.
* A new game can be started or a quit command can be issued.
* Commands to start a new game or quit the program must be represented by command buttons on the main form.
* A list of words to be used in the game must be stored internally in an array.
* The word which is to be guessed in each game must be chosen randomly and then displayed using an underscore ( \_ ) or an asterisk ( \* ) to represent each letter. The number of characters must equal the length of the word.
* Guesses must be processed through the KeyPress event of the main form.
* Correct guesses are stored in a list structure, and then revealed in the word, replacing the special characters used to conceal the word.
* Incorrect guesses are stored in a list structure and then displayed along with other incorrect guesses.
* The scoreboard is displayed in a ListBox on the main form, and contains the best number of guesses taken for each word that was attempted. These scores are stored in a list structure.
* Collection objects that must be used are ArrayList and SortedList.

**Program Design**

The name of the program is “Hangman Game.”

\*ATTENTION\*

Please be aware that my scoring system functions a little differently from the specifications – because I use a counter and made the option available to submit a word in one click, I score attempts on a guess-by guess basis. Each letter submitted counts as 1. Each word submitted also counts as 1. Your final score is determined by how many guesses you took to find the entire word. Meaning that less score is higher, more score is lower.

The goal is to create a program that simulates a hangman game while working as close to the confines of the project specifications as possible.

The program features methods that:

* Resets the game.
* Exits the game.
* Updates the scoreboard.
* Updates the high score list.
* Handles the submission of a letter.
* Handles the submission of a word.
* Replaces a high score entry if it is beaten.
* Draws the hanged man piece by piece as chances are burned.
* Keep track of the chances and end game if the maximum is met.
* Displays an error message if anything other than a letter is input.
* Displays various messages when the game is won and lost.
* Displays a message when the wrong letter or word is input.

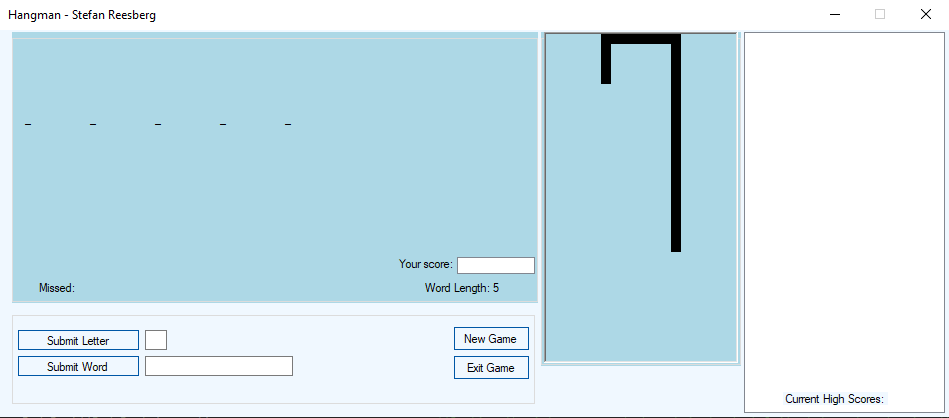
Besides these methods, correct guesses and incorrect guesses are both stored in ArrayLists as specified, with the incorrect guesses being displayed on the interface.

Scores and high scores are both stored in sortedLists (later changed to Dictionary version, but functionally the same).

**Program Use Walkthrough:**

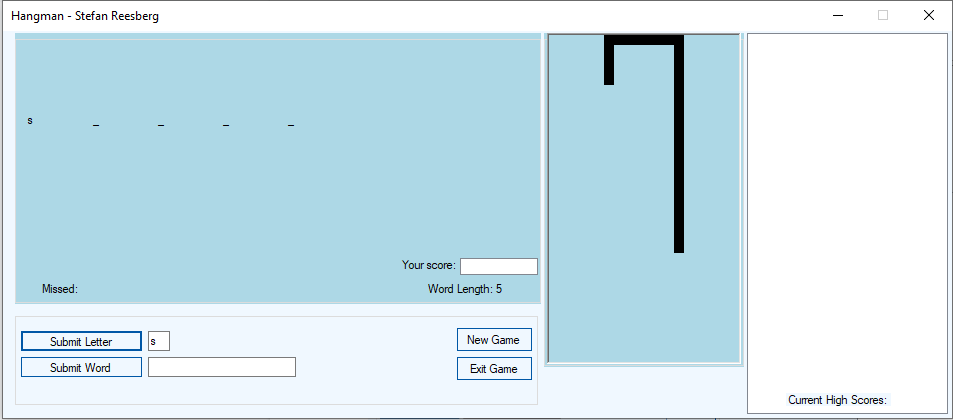
To run the program, make sure you have visual studio 2019 installed on your PC. Double click on the Hangman Game.sln file to open this project.

Press F5 to start a run of the project. The following interface should display:



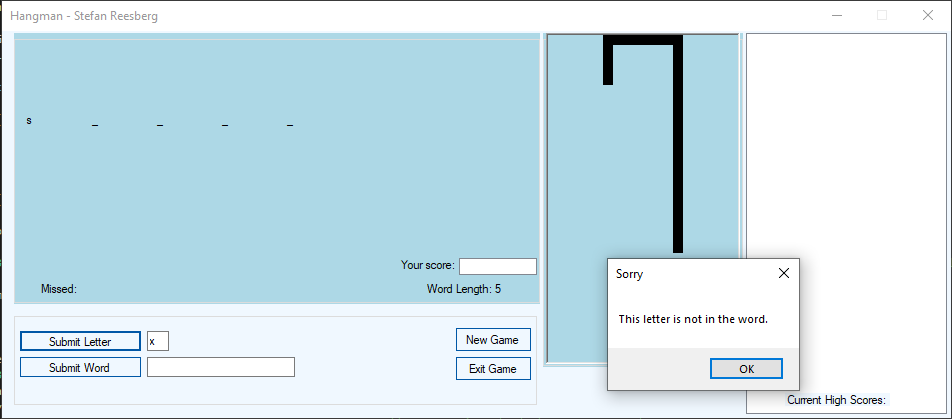
You can now navigate to either textbox to submit a letter or a word of your choice. To finalize your guess, you need to click on the submit letter/word button.

If you are correct - this displays:

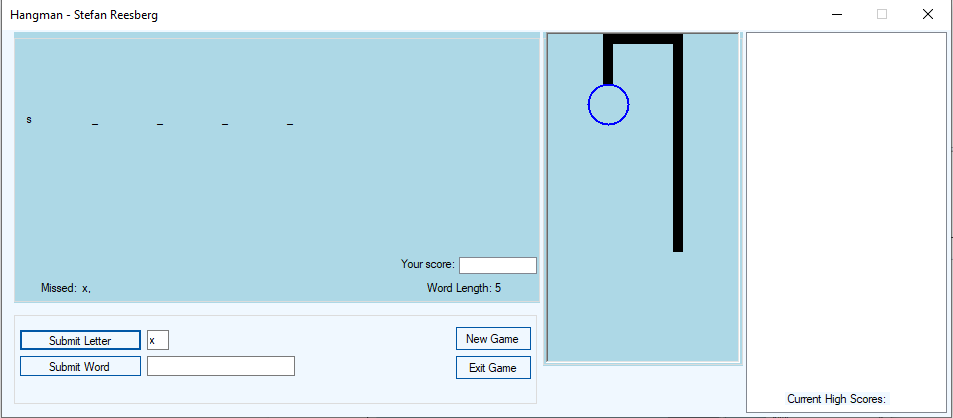


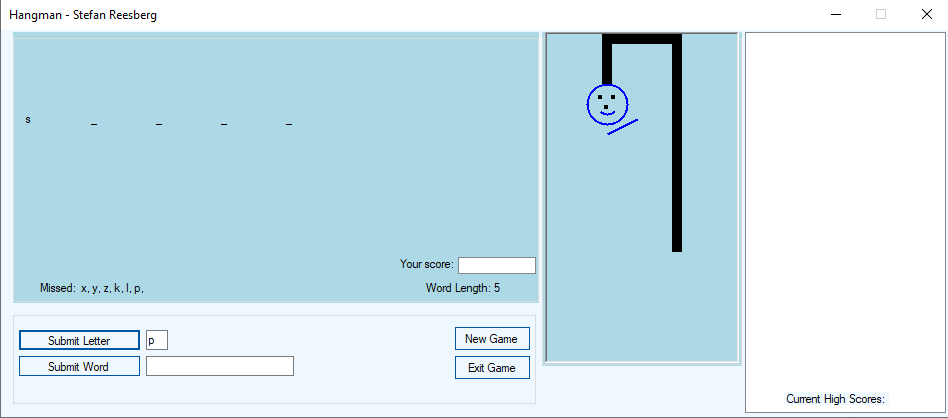
As you can see, the blank has filled in with your correct guess.

If your guess was incorrect, however:

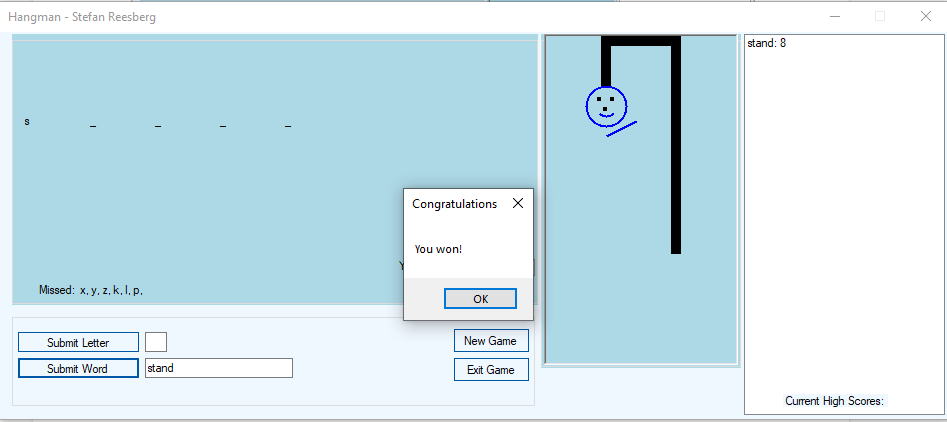


This message is displayed - and:

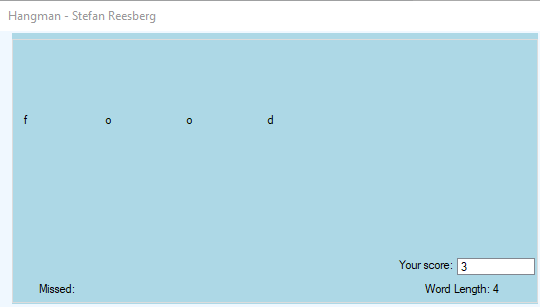


Your missed guess is displayed in the “Missed:” label. Simultaneously, the hangman itself will continue being drawn with each wrong guess.

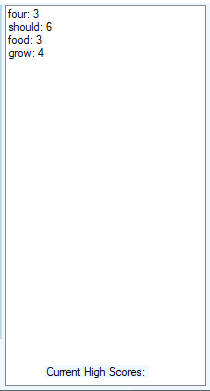
Should you manage to figure out which word it is, you can directly input the complete word to save time and achieve a higher score:



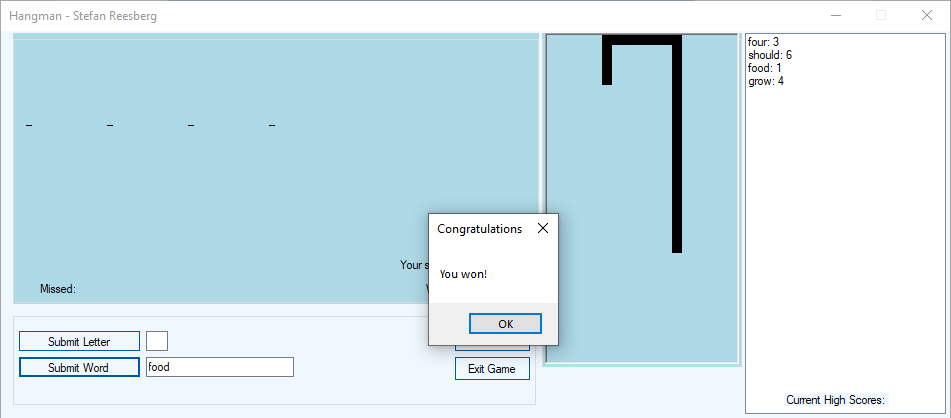
A victory message will display and the high score list will be updated with your current best attempt on that particular word. A scoreboard with your current score will also display here:



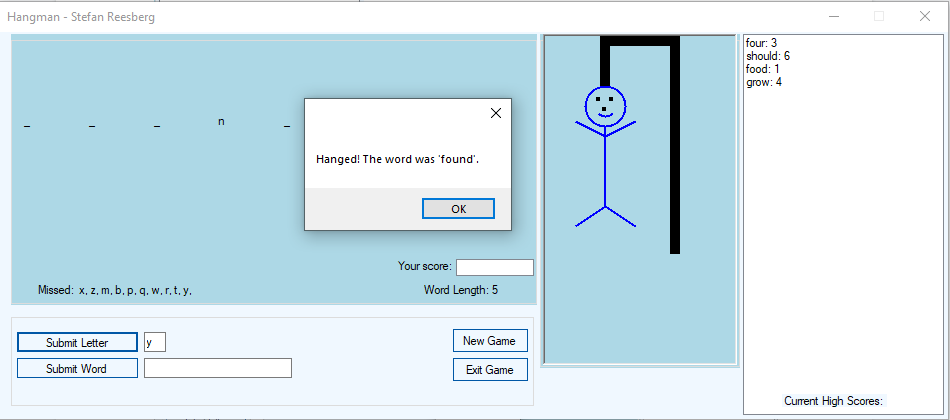
As you guess more words correctly, the high score list will start filling up:



If you come across the same word again and you manage to solve it in less guesses the high score will update:

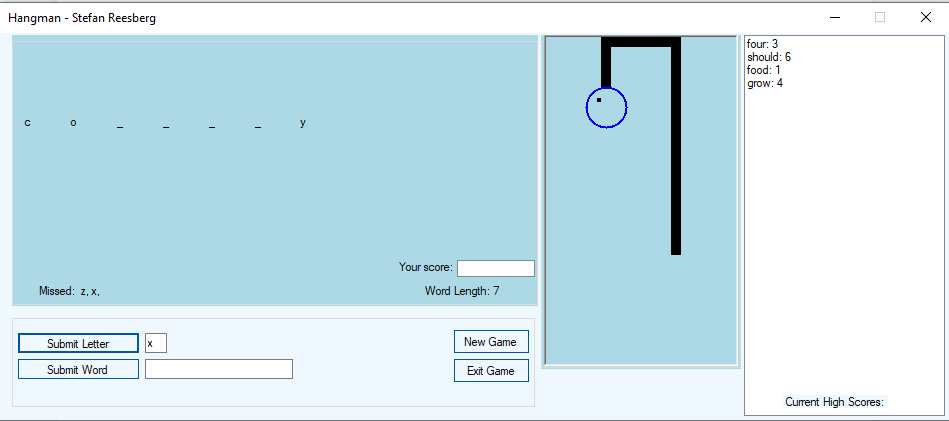


If you do not manage to guess the word within the 10 chances allotted to you, the man will hang:

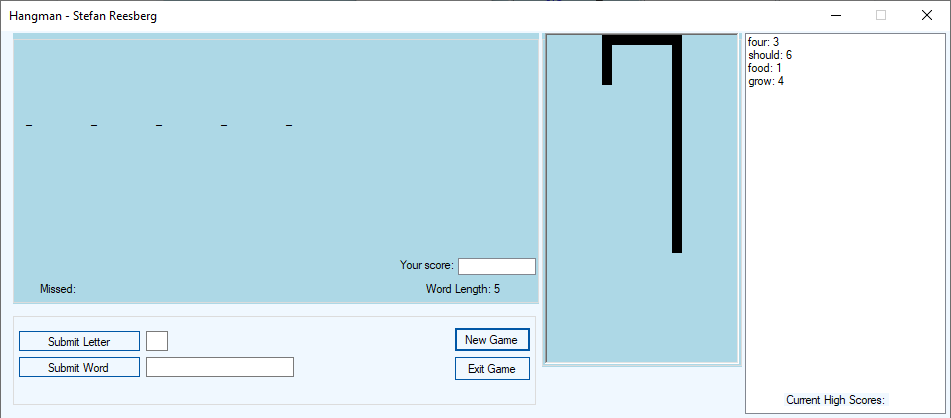


You can choose to reset the game manually by clicking on the “New Game” button:

**Game in progress**



**Upon clicking “New Game”:**



A fresh game is started with another random word and the previous attempt is scrapped.

Finally, you can exit the game by clicking on the “Exit Game” button.

**Source Code:**

/\* Filename: Hangman Game.sln

\* Author: Stefan Reesberg

\* Created: 25/06/2022

\* Operating System: Windows 10 Pro

\* Version: 21H2

\* Description: This program contains all the methods and functions that constitute a Hangman game.

\*/

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using System.Collections;

namespace Hangman\_Game

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

int amount = 0; //Initialize an integer to keep track of which bodypart needs to be drawn.

int score = 0; //Initializing the score variable for later use.

string word = ""; //Initializing the word variable for later use.

List<Label> labels = new List<Label>(); //Creating a new list of labels for the guessing game underscores.

ArrayList correctGuess = new ArrayList(); //Create an ArrayList to store correct guesses.

ArrayList wrongGuess = new ArrayList(); //Create an ArrayList to store wrong guesses.

IDictionary<string, int> scores = new Dictionary<string, int>(); //Create a SortedList to store scoreboard entries. \*Functionally the same as a sortedList, just easier to use.

IDictionary<string, int> highscores = new Dictionary<string, int>(); //Create a sortedList for high score display. \*Functionally the same as a sortedList, just easier to use.

enum BodyParts //Creating an enumeration to hold all the bodyparts that will be drawn.

{

Head,

Left\_Eye,

Right\_Eye,

Mouth,

Nose,

Right\_Arm,

Left\_Arm,

Body,

Right\_Leg,

Left\_Leg

}

void DrawHangPost() //This method draws the hang post of the game.

{

Graphics g = graphicsPanel.CreateGraphics(); //Creation of graphics to draw inside the panel.

Pen p = new Pen(Color.Black, 10); //Drawing with a pen since I will only be using lines.

g.DrawLine(p, new Point(130, 218), new Point(130, 5));

g.DrawLine(p, new Point(135, 5), new Point(65, 5));

g.DrawLine(p, new Point(60, 0), new Point(60, 50));

MakeLabels(); //Calls the method that creates labels in the for of underscores.

}

void DrawBodyPart(BodyParts bp) //This method draws each body part. The enumeration is passed to this method.

{

Graphics g = graphicsPanel.CreateGraphics(); //Creation of graphics to draw inside the panel.

Pen p = new Pen(Color.Blue, 2); //Drawing with a pen again since I will be using lines.

if (bp == BodyParts.Head) //Check which body part will be drawn using if/else if.

g.DrawEllipse(p, 40, 50, 40, 40);

else if (bp == BodyParts.Left\_Eye)

{

SolidBrush s = new SolidBrush(Color.Black); //Creating a new brush. A dot will do for the eyes.

g.FillEllipse(s, 50, 60, 5, 5); //An Ellipse this small creates a square for the eye.

}

else if (bp == BodyParts.Right\_Eye)

{

SolidBrush s = new SolidBrush(Color.Black);

g.FillEllipse(s, 63, 60, 5, 5);

}

else if (bp == BodyParts.Mouth)

{

g.DrawArc(p, 50, 60, 20, 20, 45, 90); //Drawing an arc for the mouth.

}

else if (bp == BodyParts.Body)

{

g.DrawLine(p, new Point(60, 90), new Point(60, 170)); //Lines for the body, arms and legs.

}

else if (bp == BodyParts.Left\_Arm)

{

g.DrawLine(p, new Point(60, 100), new Point(30, 85));

}

else if (bp == BodyParts.Right\_Arm)

{

g.DrawLine(p, new Point(60, 100), new Point(90, 85));

}

else if (bp == BodyParts.Left\_Leg)

{

g.DrawLine(p, new Point(60, 170), new Point(30, 190));

}

else if (bp == BodyParts.Right\_Leg)

{

g.DrawLine(p, new Point(60, 170), new Point(90, 190));

}

else if (bp == BodyParts.Nose) //For the 10th chance, added a nose in the form of a small square.

{

SolidBrush s = new SolidBrush(Color.Black);

g.FillEllipse(s, 56, 70, 5, 5);

}

}

void MakeLabels() //This method will generate the underscores for the guessing game.

{

word = GetRandomWord(); //Assigning the randomly generated word to a variable.

char[] chars = word.ToCharArray(); //Converts the string into a character array.

int between = 330 / chars.Length - 1; //Creating an integer that determines how many pixels will be between the labels that hold the underscores.

labels.Clear(); //Clearing the labels every New Game to prevent hiccups.

for (int i = 0; i < chars.Length; i++) //Creating a for loop that will iterate through each character and create a label for it.

{

labels.Add(new Label()); //Adding a new label to the list.

labels[i].Location = new Point((i \* between) + 10, 80); //Specify the location of the label.

labels[i].Text = "\_"; //Assigning the underscore appearance for the label's text.

labels[i].Parent = displayBox; //Set parent of the control.

labels[i].BringToFront(); //Bring the label on top of every other control.

labels[i].CreateControl(); //Create the control and show it on the form.

}

wordLengthLabel.Text = "Word Length: " + (chars.Length).ToString();//Changing the text of label1 to display the current word's length.

}

string GetRandomWord() //This method will draw a random word from the array that I populated manually.

{

string[] words = { "head", "stand", "own", "page", "should", "country",

"found", "school", "grow", "study", "still", "learn", "plant", "cover",

"food", "sun", "four", "between", "state", "keep", "eye", "never", "last", "let", "thought" };

Random ran = new Random(); //Creating a new instance of the random class.

return words[ran.Next(0, words.Length - 1)]; //Returning a random index from the array. Negative 1 because the array starts at 0.

}

private void MainForm\_Shown(object sender, EventArgs e)

{

DrawHangPost(); //Calling the hangpost method.

MessageBox.Show(word); //Answer for testing

}

void ResetGame() //This method resets the game to a new state.

{

Graphics g = graphicsPanel.CreateGraphics(); //Recreate graphics for the panel.

g.Clear(graphicsPanel.BackColor); //Clear everyting in the panel.

GetRandomWord(); //Generate another random word.

MakeLabels(); //Generate the next set of labels for the new word.

DrawHangPost(); //Draw the hangpost again.

scoreboard.Items.Clear(); //Clears the scoreboard.

score = 0; //Reset the counter for score.

amount = 0; //Reset the bodypart counter.

missedLabel.Text = "Missed: "; //Reset the missed guesses label.

letterTextBox.Text = ""; //Reset textboxes.

wordTextBox.Text = "";

MessageBox.Show(word); //Answer for testing

}

private void SubmitLetter\_Click(object sender, EventArgs e) //Handles the Submit Letter button click event.

{

score++; //Increments the score variable by 1 for the scoreboard display.

char letter = letterTextBox.Text.ToLower().ToCharArray()[0]; //Get the character that the user submitted, forced to lowercase and saved to a variable. The textBox is limited to 1 element in properties.

if (!char.IsLetter(letter)) //Check if the character entered is actually a letter.

{

MessageBox.Show("You can only submit letters!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error); //Messagebox displaying the error message.

return;

}

if (word.Contains(letter)) //Check if the word contains the letter submitted.

{

correctGuess.Add(letter); //Adds the letter to the correct ArrayList.

char[] letters = word.ToCharArray(); //Create a character array of containing all the characters in the word.

for (int i = 0; i < letters.Length; i++) //Iterate through the entire array.

{

if (letters[i] == letter) //Check if the letter submitted is the correct letter.

labels[i].Text = letter.ToString(); //Changes the label's text from underscore to the correctly submitted letter.

}

foreach (Label l in labels) //Check to see if the user has won upon every letter entry.

if (l.Text == "\_") return; //If the text is equal to an underscore, return so that it never reaches the win code.

UpdateWordScore(); //Updates the scoreboard and highscore list.

MessageBox.Show("You won!", "Congratulations"); //Once all underscores are changed to the correct letter, this message will display.

ResetGame(); //Reset the game when you win.

}

else //If the letter is not in the word:

{

wrongGuess.Add(letter); //Adds the letter to the incorrect ArrayList.

MessageBox.Show("This letter is not in the word.", "Sorry"); //Display a message to the user.

missedLabel.Text += " " + letter.ToString() + ","; //Adding the incorrect word to a "Missed" label to display incorrect guesses.

DrawBodyPart((BodyParts)amount); //Casting the integer as a bodypart instead of 9 if statements, drawing the next bodypart upon incorrect guess.

amount++; //Incrementing the bodypart counter by 1.

if (amount == 10) //Reaching the final bodypart triggers the game lost message.

{

MessageBox.Show("Hanged! The word was '" + word + "'.");

ResetGame(); //Call the method to reset the game state when you lose.

}

}

}

private void UpdateWordScore() //This method updates the scoreboard and the highscore list.

{

if (!scores.ContainsKey(word)) //Checks if the scores dictionary does not contain the current word's key.

{

scores.Add(word, score); //Adds the score to the sortedList. \*The dictionary version of it.

}

else //Otherwise, set the value to score.

{

scores[word] = score;

}

if (!highscores.ContainsKey(word)) //Checks if the highscores dictionary does not contain the word's key.

{

highscores.Add(word, score); //Adds the high score to the Dictionary for display later.

}

if (score < highscores[word]) //Check if the current score is lower than the previous high score.

{

highscores[word] = score; //Replace the score in the highscore dictionary.

}

scoreboard.Items.Add(scores[word]); //Add the score entry to the scoreboardListBox.

UpdateHighScoreListBox(); //Update the high score listBox using the method.

}

private void UpdateHighScoreListBox() //This method updates the high score listBox, displaying the high scores of each word attempted.

{

highScoresBox.Items.Clear(); //Clears the highscore listBox.

foreach (var word in highscores.Keys) //Iterates through the keys in highscores Dictionary.

{

var score = highscores[word]; //Sets the score variable to the highscore key's value.

highScoresBox.Items.Add($"{word}: {score}"); //Adds the word and it's score as a string to display in the highScoresBox.

}

}

private void SubmitWord\_Click(object sender, EventArgs e) //Handles when the user submits a full word.

{

score++; //Increments the score variable by 1 for the scoreboard display.

if (wordTextBox.Text == word) //Check to see if the word the user has submitted is correct.

{

correctGuess.Add(word); //Adds the word to the correctGuess ArrayList.

UpdateWordScore(); //Updates the scoreboard and highscore list.

MessageBox.Show("You won!", "Congratulations"); //If it is, display victory message.

ResetGame(); //Reset game upon victory.

}

else //What happens when the word is wrong.

{

wrongGuess.Add(word); //Adds the word to the wrongGuess ArrayList.

MessageBox.Show("This word is incorrect.", "Sorry"); //Display wrong guess message.

DrawBodyPart((BodyParts)amount); //Draws the next body part.

amount++; //Incrementing amount by 1 for the next bodypart in the enumeration.

if (amount == 10) //Check if the user has lost.

{

MessageBox.Show("Hanged! The word was '" + word + "'."); //Loss message.

ResetGame(); //Reset the game.

}

}

}

private void NewGame\_Click(object sender, EventArgs e)

{

ResetGame(); //Resets the game upon clicking 'New Game'.

}

private void ExitGame\_Click(object sender, EventArgs e)

{

this.Close(); //Closes the game upon clicking the 'Exit Game' button.

}

}

}