

**Artificial Intelligence (MOD004553)**

Element 010: Documentation Report

**SID: 1839616**

**Due Date: Canvas, 2pm, 14th December 2021**

**Faculty: Science and Technology**

**Department: Computing & Technology**

**Module Code: MOD004553**

**Academic Year: 2021/22**

**Trimester: 1**

**Word Count: 714 words**

**This count excludes Figures, Headings, Contents and References.**

**Contents**

**1. Introduction ..……………………………………………………...…….……. 3**

**2. Evaluation and Implementation ………………………………...………….. 3**

**3. Strengths, Limitations of the Program …….…………….…...…...……… 6**

**4. Appendix 1 …………………………………………………..….……..……….. 7**

**5. Appendix 2 …………………………………………….…………..…………... 17**

**6. References ……………………………………….………..……...…….….…. 17**

1. **Introduction**

In this documentation report, I will show the approach I have taken in designing and implementing a program for the **Guess The Word** application and also state the strengths and limitations while designing the program.

1. **Implementation and Evaluation**

The program built executes cleanly and follows all word-family rules with no run-time errors. The project consists of 4 classes Character.cs, MyExpection.cs, WordFamily.cs, and the main file Program.cs. The program starts with picking a random number between 4 and 12, which is the length of the word. All of the words of that length will then be picked from the dictionary and be stored in the ArrayList called wordFamily. The user is then asked to pick a difficulty level, either easy or hard (i.e., e or h).

**For Easy difficulty level:**

It starts off by calculating the number of the guesses the user has which is the double of the length of the word. We then declare a few ArrayLists A0 - A6, the number after the letter “A” represents the number of times the letter is repeated in the word. A while loop is then initiated which starts a counter from 0 to no. of guesses the user has. The while loop starts out by printing out the number of guesses the user has, then all the letters guessed by the user are displayed in alphabetical order. The program then asks the user to enter a guess. The words in the word family are then filtered based on how many times the guessed letter is in the word and stored in the respective ArrayList A0-A6. A0 stores all the words that do not contain the letter, A1 stores all the words that contain a single copy of the letter, A2 stores all the words that contain 2 copies of the letter and so on up until A6. Then the size of the ArrayLists are checked and the ArrayList with the maximum length is stored in ArrayList A along with the user guessed letter being added to the correctGuessedChars if it is correct and also a message being displayed on how many times the letter is in the word. The wordFamily ArrayList containing all the words of the length picked is then emptied and contents of the ArrayList A is stored in wordFamily. The program then checks if the size of the wordFamily is greater than 1, then a random word with the letter at a specific position in the wordFamily arraylist is picked and stored as the current word and is displayed in a hidden format. The users guessed letter is then compared to the all the words in the wordFamily. If the guessed letter is present in the word then the input gets stored in the correctGuessedChars and the message stating how many times that letter is present, if the guessed letter is not present in the word then it gets stored into the incorrectGuessedChars and a message saying “no copies of this letter” is printed, this process will be done in a loop until either the user has run of guesses or the If the user manages to successfully guess the word a message congratulating the user is displayed if not the user is shown the “You Lose” message along with any random word from the wordFamily ArrayList.

**For the Hard difficulty level:**

The computer picks all the words that contain X, J, Q, Z, and the program then runs with the same algorithm as the easy level (i.e., largest word family).

According to a blog post by Nick Berry on the best Hangman approach, one of the most common guesses is the letter 'S,' but the most common letter guessed is the letter 'E'. (Berry, 2012).

Jon McLoone, a mathematician, created a Hangman computer game to mimic how real human players may guess. McLoone simulated 50 games of Hangman for every word in his 90,000-word dictionary, based on the idea that the average individual would predict common letters like R, S, T, and the five vowels more often than rare letters like Q, Z, J, and X. After nearly five million Hangman games, When the word containing uncommon letters like J, Q, X, and Z, guessing correctly becomes even more difficult (Specktor, 2021). Fizz, buzz, jink, hajj, and quiz were among the top 25 words on McLoone's list of unguessable words (Moazzam et al., 2015). Regardless of whether the computer was given 8, 9, 10, 11, 12, or even 13 guesses, one word reigned supreme. According to science, the most difficult word to guess in hangman is jazz (Moazzam et al., 2015). The screenshots shown below illustrate that words that contain letters like J, Q, X, and Z have higher incorrect guesses than words that do not have these characters.

Table

Description automatically generated

Figure 1: Screenshot showing guesses taken to guess words with letters X, Z, Q, and Z

A picture containing text, computer

Description automatically generated

Figure 2: Screenshot showing guesses taken to guess words with letters X, Z, Q, and Z

1. **Strengths, Limitations of the Program**

**Limitations of the Program**

* The program does not check for word families of the same sizes.

**Strengths of the Program**

* The program runs and executes cleanly.
* The program does check for if there was no Word Family made, If not then a random word is used for the game.

1. **Appendix 1**

**Character.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WordGame

{

class Character

{

public static bool Compare(char x, char y)

{

if (x == y)

return true;

else

return false;

}

public static char toLower(char ch)

{

if (ch >= 65 && ch <= 90)

return (Char)(ch + 32);

else

return ch;

}

}

}

**MyExpection.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WordGame

{

class InvalidInputException : Exception

{

public InvalidInputException()

{

Console.WriteLine("Invalid input character");

}

}

}

**WordFamily.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections;

namespace WordGame

{

class WordFamily

{

ArrayList wordFamily;

public WordFamily ()

{

wordFamily = new ArrayList();

}

public void Add(string word)

{

wordFamily.Add(word);

}

public virtual Array ToArray()

{

return wordFamily.ToArray();

}

public int Count()

{

return wordFamily.Count;

}

public void RemoveRange(int start, int end)

{

wordFamily.RemoveRange(start, end);

}

public void AddRange(WordFamily cons)

{

foreach (string word in cons.wordFamily)

this.Add(word);

}

public string getWord(int pos)

{

return wordFamily[pos].ToString();

}

}

}

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Collections;

namespace WordGame

{

class Program

{

static void Main(string[] args)

{

int length, chances, counter = 0;

string chosenWord = "";

Random r = new Random();

char choice = 'y', ch, level;

do

{

counter = 0;

length = r.Next(4, 12);

string[] list = File.ReadAllLines("dictionary.txt");

WordFamily wordFamily = new WordFamily();

foreach (string word in list)

{

int len = word.Length;

if (length == len)

wordFamily.Add(word);

}

do

{

Console.WriteLine("Welcome to Guess The Word Game....Good Luck :)");

Console.WriteLine();

Console.WriteLine("Choose level : Easy or Hard (e/h) : ");

bool val = Char.TryParse(Console.ReadLine(), out level);

try

{

if (!val)

throw new InvalidInputException();

else

{

level = Character.toLower(level);

break;

}

}

catch (InvalidInputException)

{

///Console.WriteLine(e.Message);

}

} while (true);

//===================Hard Level===================

if (level == 'h')

{

WordFamily conchars = new WordFamily();

//scan the list if entered character presents in the word consecutively, add it to the new

// family

foreach (string word in wordFamily.ToArray())

{

for (int x = 0; x < length - 1; x++)

{

if(Character.Compare(word[x], 'j') || Character.Compare(word[x], 'q') ||

Character.Compare(word[x], 'x') || Character.Compare(word[x], 'z') )

{

conchars.Add(word);

break;

}

}

}

//make it a main family , if it has words

if (conchars.Count() > 0)

{

wordFamily.RemoveRange(0, wordFamily.Count());

wordFamily.AddRange(conchars);

}

}

chances = length \* 2;

IDictionary<int, char> correctGuessedCharsWithPos = new Dictionary<int, char>();

ArrayList correctGuessedChars = new ArrayList();

ArrayList incorrectGuessedChars = new ArrayList();

WordFamily A = new WordFamily();

WordFamily A0 = new WordFamily();

WordFamily A1 = new WordFamily();

WordFamily A2 = new WordFamily();

WordFamily A3 = new WordFamily();

WordFamily A4 = new WordFamily();

WordFamily A5 = new WordFamily();

WordFamily A6 = new WordFamily();

while (counter < chances)

{

Console.WriteLine("You have " + (chances - counter) + " guesses left.");

Console.Write("Used letters: ");

List<char> usedLetters = new List<char> { };

foreach (char c in correctGuessedChars)

usedLetters.Add(c);

foreach (char c in incorrectGuessedChars)

usedLetters.Add(c);

usedLetters.Sort();

foreach (char c in usedLetters)

Console.Write(c + " ");

Console.WriteLine();

//Console.WriteLine("Number of words in word family is : " + wordFamily.Count);

if (wordFamily.Count() > 1)

{

int no = r.Next(0, wordFamily.Count() - 1);

chosenWord = (string)wordFamily.getWord(no);

}

//Console.WriteLine(chosenWord);

//display the chosen word to user in hidden form

Console.Write("Word: ");

int i = 0;

foreach (char c in chosenWord)

{

if (correctGuessedChars.Contains(c))

{

Console.Write(c); //expose the character to user

if (!correctGuessedCharsWithPos.ContainsKey(i))

correctGuessedCharsWithPos.Add(i, c);

}

else

Console.Write("-"); // hide the character

i++;

}

Console.WriteLine();

do

{

Console.Write("Enter guess: ");

bool val = Char.TryParse(Console.ReadLine(), out ch);

try

{

if (!val)

throw new InvalidInputException();

else

break;

}

catch(InvalidInputException)

{

///Console.WriteLine(e.Message);

}

} while (true);

//filter the words depending on the number of times the character is present in the

// word to different arraylist

//the successiding number in the name of the array list is the number of times the

// character is present in it.

//Arraylist A is to store the list with maximum number of words

//empty all sub word families

A.RemoveRange(0, A.Count());

A0.RemoveRange(0, A0.Count());

A1.RemoveRange(0, A1.Count());

A2.RemoveRange(0, A2.Count());

A3.RemoveRange(0, A3.Count());

A4.RemoveRange(0, A4.Count());

A5.RemoveRange(0, A5.Count());

A6.RemoveRange(0, A6.Count());

foreach (string word in wordFamily.ToArray())

{

int number = 0;

foreach (var c in correctGuessedCharsWithPos)

{

int y = c.Key;

char x = c.Value;

for (int a = 0; a < word.Length; a++)

{

if(Character.Compare(word[a], x) && a == y)

{

number++;

}

}

}

if (number == correctGuessedCharsWithPos.Count)

A.Add(word);

}

if (A.Count() != 0)

{

wordFamily.RemoveRange(0, wordFamily.Count());

wordFamily.AddRange(A);

A.RemoveRange(0, A.Count());

}

if (wordFamily.Count() > 1)

{

foreach (string word in wordFamily.ToArray())

{

int cnt = 0;

foreach (char c in word)

{

if (c == ch)

cnt++;

}

switch (cnt)

{

case 0: A0.Add(word);

break;

case 1: A1.Add(word);

break;

case 2: A2.Add(word);

break;

case 3: A3.Add(word);

break;

case 4: A4.Add(word);

break;

case 5: A5.Add(word);

break;

case 6: A6.Add(word);

break;

}

}//end of foreach loop

//find out the list with maximum number of words and assign it to list A.

int max = 0;

int listNo = 0;

if (max < A6.Count())

{

max = A6.Count();

listNo = 6;

}

if (max < A5.Count())

{

max = A5.Count();

listNo = 5;

}

if (max < A4.Count())

{

max = A4.Count();

listNo = 4;

}

if (max < A3.Count())

{

max = A3.Count();

listNo = 3;

}

if (max < A2.Count())

{

max = A2.Count();

listNo = 2;

}

if (max < A1.Count())

{

max = A1.Count();

listNo = 1;

}

if (max < A0.Count())

{

max = A0.Count();

listNo = 0;

}

switch(listNo)

{

case 0: A = A0;

incorrectGuessedChars.Add(ch);

Console.WriteLine("Sorry, there are no " + ch + "'s.");

break;

case 1: A = A1;

correctGuessedChars.Add(ch);

Console.WriteLine("Yes, there is 1 copy of " + ch + ".");

break;

case 2: A = A2;

correctGuessedChars.Add(ch);

Console.WriteLine("Yes, there are 2 copies of " + ch + ".");

break;

case 3: A = A3;

correctGuessedChars.Add(ch);

Console.WriteLine("Yes, there are 3 copies of " + ch + ".");

break;

case 4: A = A4;

correctGuessedChars.Add(ch);

Console.WriteLine("Yes, there are 4 copies of " + ch + ".");

break;

case 5: A = A5;

correctGuessedChars.Add(ch);

Console.WriteLine("Yes, there are 5 copies of " + ch + ".");

break;

case 6: A = A6;

correctGuessedChars.Add(ch);

Console.WriteLine("Yes, there are 6 copies of " + ch + ".");

break;

}

}

else

{

int temp = 0;

foreach (char c in chosenWord)

{

if (c == ch)

temp++;

}

if(temp != 0)

{

Console.WriteLine("Yes, there is/are " + temp + " copies of " + ch + ".");

correctGuessedChars.Add(ch);

}

else

{

Console.WriteLine("There are no copies of " + ch + "'s.");

incorrectGuessedChars.Add(ch);

}

}

Console.WriteLine();

wordFamily.RemoveRange(0, wordFamily.Count());

wordFamily.AddRange(A);

//check if user has guessed the word!

if (correctGuessedCharsWithPos.Count == length)

{

Console.WriteLine("Congratulations!!!! You guessed the word correct!!!");

goto replay;

}

counter++;

}//end of while

if (counter == chances)

{

if(wordFamily.Count() > 1)

{

int no = r.Next(0, wordFamily.Count() - 1);

chosenWord = (string)wordFamily.getWord(no);

}

Console.WriteLine("You lose!!! The word was " + chosenWord);

}

replay:

Console.WriteLine();

Console.WriteLine("Play again !!! (y/n) : ");

choice = Char.Parse(Console.ReadLine());

}while(choice != 'n');

}

}

}

1. **Appendix 2**

**Instructions**

* Include the dictionary.txt file in the working directory and necessary libraries.

1. **References**

*Berry, N., 2012. A Better Hangman Strategy. [online] Datagenetics.com. Available at: <https://datagenetics.com/blog/april12012/index.html> [Accessed 7 December 2021].*

*Moazzam, W., Riegler, M., Sen, S. and Nygård, M., 2015. [pdf] Scientific Hangman: Gamifying Scientific Evidence for General Public. In GamifIR@ ECIR (pp. 26-33). Available at:*

*< http://ceur-ws.org/Vol-1345/gamifir15\_4.pdf > [Accessed 10 December 2021].*

*Specktor, B., 2021. Here Is the Hardest Word to Guess in Hangman, According to Science. [online] Reader's Digest. Available at: <https://www.rd.com/article/hardest-word-guess-hangman/> [Accessed 11 December 2021].*