Huy Quang Tran s327944

Huy Quang Tran

Software Unit Testing Report

Hangman game

Contents

[Workflow of the game 1](#_Toc50002737)

[Test cases 2](#_Toc50002738)

[Refactor code 9](#_Toc50002739)

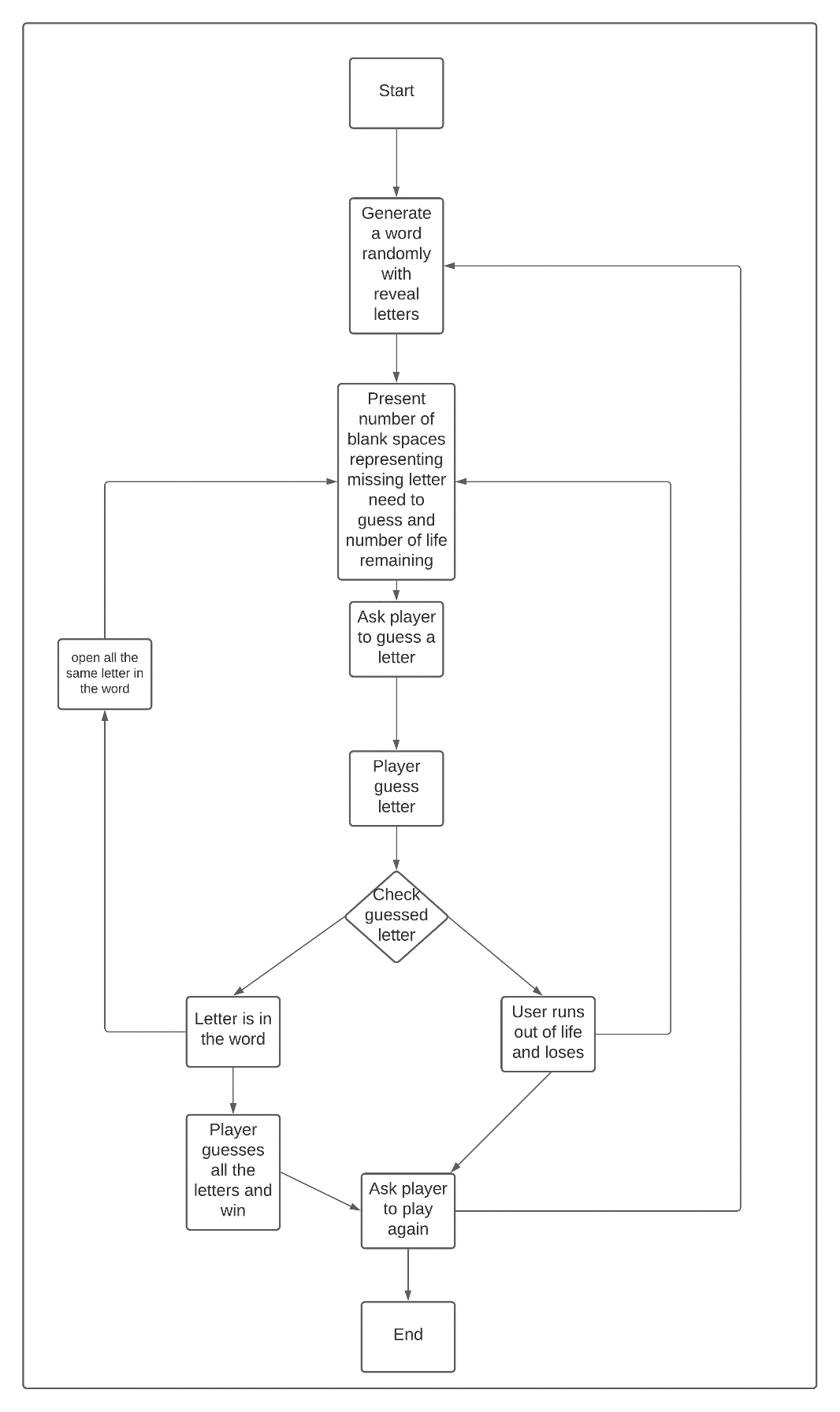
[How TDD has been implemented to creating the game 9](#_Toc50002740)

[References 9](#_Toc50002741)

Software Unit Testing Report

Hangman game

# Workflow of the game



# 

# Github link

All the files of the game are stored in GitHub which can be accessed through the link:

<https://github.com/huytranhanoi85/Hangman-Game.git>

# How TDD has been implemented to creating the game

The test cases has been developed firstly, then the game code follows to make sure all the tests are passed. Once a test has passed, then the next one is implemented, if it is failed, the code needs to change until it passes the tests. When all the tests are passed, the game is completed. The hangman game is quite a small project, however, there would be many bugs if the first stages are not well prepared. Applying TDD methods helps to reduce the coding time by reducing number of bugs and improve code quality ( [Oak](https://www.techrepublic.com/search/?a=harshad+oak) H, 2003).

# Test cases

|  |  |
| --- | --- |
| Unit test | Description |
|  | This test is to make sure when user guesses a letter correctly in the given word it will notice it |
|  | This test is to make sure when user guesses a letter which is not in the given word it will notice it. |
|  | Test if the user does not enter any letter but press enter so it will repeat asking he/she does not leave blank. |
|  | Test when user has spent all their chance but still cannot get the word so the game will be over  Expected output: true |
|  |  |
|  | The test is to check when user inputs correctly all letters so he/she will win |
|  | The test is to check when the game is over |
|  |  |
|  | The test is to check if user input wrongly all the time with the difficult level medium. |
|  | Test number of player life remaining with the difficult level is easy |
|  |  |
|  | Test number of player life remaining with the difficult level is medium |
|  |  |
|  | Test number of player life remaining with the difficult level is hard |
|  |  |
|  | To test the function Founword() when the game has started. If player has not found the word yet it will return false. It means the word has not been found yet. |
|  |  |
|  | To test the function Founword() when the word is guessed correctly. |
|  |  |
|  |  |
|  |  |
|  | The test to check the function Loop() . |

# Hangman game code

// Project name: Hangman game

// Author: Huy Quang Tran

// Created date: 28/08/2020

using System;

using System.Collections.Generic;

using System.Security.Cryptography.X509Certificates;

namespace HangmanGame

{

public class Hangman

{

static string theSecretWord = "";

public Hangman()

{

}

public bool GuessLetter(string letter)

{

bool foundLetter = false;

if (String.IsNullOrEmpty(letter))

return false;

if (letter.Length > 1)

{

Console.WriteLine("please enter only 1 letter to play");

return false;

}

for (int i = 0; i < Guesses.Count; i++)

//checking missing word is correct

if (Guesses[i] == '\_' && SecretWord[i].ToString() == letter.ToUpper())

{

Guesses[i] = char.Parse(letter.ToUpper());

Console.WriteLine(letter + " is correct!");

return true;// return when letter matched

}

//retun when letter not match.

this.NumberOfGuesses--;

Console.WriteLine(letter + " is incorrect!");

return foundLetter;

}

public bool EnterWord(string secretWord)

{

if (!string.IsNullOrWhiteSpace(secretWord))

{

Random randGen = new Random();

foreach (var item in secretWord.ToUpper().ToCharArray())

{

var idx = randGen.Next(0, 9);

if (idx % 2 ==0)

this.Guesses.Add('\_');

else

this.Guesses.Add(item);

this.SecretWord.Add(item);

}

return true;

}

return false;

}

private List<char> Guesses

{

get;

set;

} = new List<char>();

private List<char> SecretWord

{

get;

set;

} = new List<char>();

private int NumberOfGuesses

{

get;

set;

}

public string Showword(string mysteryWord)

{

theSecretWord = mysteryWord;

return theSecretWord;

}

public bool GameOver()

{

bool isOver = false;

if (NumberOfRemainingGuesses() <= 0 || FoundWord())

{

isOver = true;

}

if (NumberOfRemainingGuesses() <= 0)

{

Console.WriteLine("You lose!");

} else if (FoundWord())

{

Console.WriteLine("");

Console.WriteLine("Congratiation!You won!\n");

Console.WriteLine("the secret word is " + theSecretWord.ToUpper() + "\n");

}

return isOver;

}

public void SetDifficulty(Enums.Difficulty difficulty)

{

NumberOfGuesses = (int)difficulty;

}

public int NumberOfRemainingGuesses()

{

return NumberOfGuesses;

}

public bool FoundWord()

{

bool hasFoundWord = false;

foreach (char c in SecretWord)

{

if (Guesses.Contains(c))

{

hasFoundWord = true;

}

else

{

hasFoundWord = false;

break;

}

}

return hasFoundWord;

}

public bool Loop()

{

//Setup Game

//Play Game

for (int i = 0; i < this.SecretWord.Count; i++)

if (this.SecretWord[i]!=this.Guesses[i])

return false;

return true;

}

public bool PlayGame()

{

// TODO Outputs to concole current guesses, found letters, and remaing free guesses

// Guess letters (win/lose)

// Validation checks?

while (true)

{

Console.WriteLine("The word is (times remain: "+this.NumberOfRemainingGuesses().ToString()+"):" +string.Join("",this.Guesses));

Console.Write("Please enter guess word: ");

GuessLetter(Console.ReadLine());

if (GameOver())

{

Console.WriteLine("The game has finished!");

break;

}

if (Loop())

{

Console.WriteLine("The GameOver method is failing.");

break;

}

}

return false;

}

public bool SetupGame(string mysteryWord,Enums.Difficulty difficulty)

{

//enter word

string[] listwords = new string[10];

listwords[0] = "sheep";

listwords[1] = "goat";

listwords[2] = "computer";

listwords[3] = "america";

listwords[4] = "watermelon";

listwords[5] = "icecream";

listwords[6] = "jasmine";

listwords[7] = "pineapple";

listwords[8] = "orange";

listwords[9] = "mango";

Random randGen = new Random();

var idx = randGen.Next(0, 9);

if (mysteryWord=="")

mysteryWord = listwords[idx];

theSecretWord = mysteryWord;

//enter difficulty

this.EnterWord(mysteryWord);

this.NumberOfGuesses = (int)difficulty;

//check values

return false;

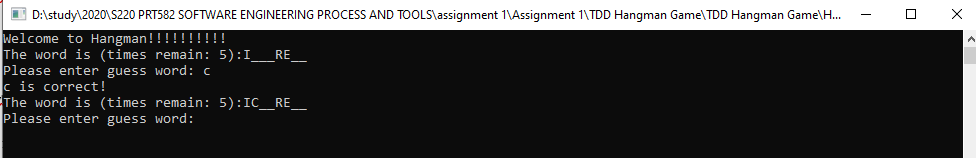
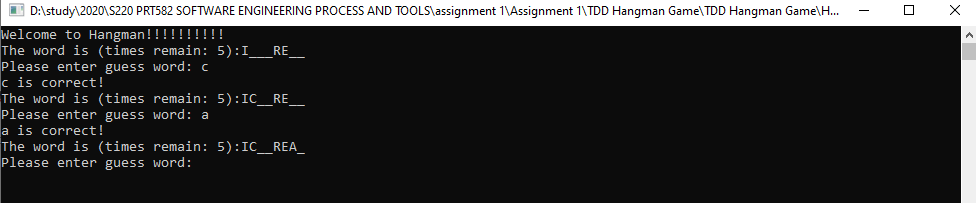
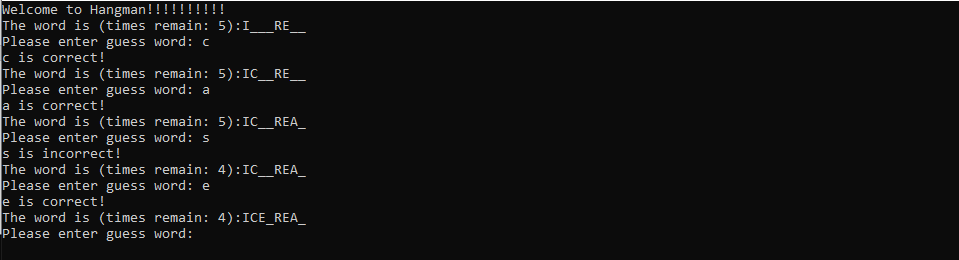
}

}

}

# Game running

# 



# Refactor code

Code refactoring method is used to optimize the code by reducing unnecessary redundancy and reducing number of coding lines.

|  |  |
| --- | --- |
| Before refactor code |  |
| This code are using two if. It can remove one if and use “&&” | After refactor and run unit test still Passed |
|  |  |
|  | Reduce the code size |

# References

Oak H, 2003, *Test-driven development lets you eliminate bugs as you code*, available at <https://www.techrepublic.com/article/test-driven-development-lets-you-eliminate-bugs-as-you-code/>