```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Reflection.Emit;
using System.Xml.Schema;
using System.ComponentModel;
using System.Diagnostics.Eventing.Reader;
using System.Security.Policy;

namespace Hangman
{
   public class User
   {
      //Declared the user information variables
      public static string userName { get; set; }
      public static string ini;
      public static string gameMode { get; set; }
      public static List<string> wordTypes = new List<string>();

      public static int score { get; set; }
      public static int allTimeScore { get; set; }
      public static int highScore { get; set; }
      public static int lowScore { get; set; }
      public static int gamesPlayed { get; set; }
      public static int gamesWon { get; set; }
      public static int gamesLost { get; set; }
```

```
public static void SetUserName(string name) // Set the user's name and the ini file name.
{
    userName = name;
    ini = name + ".ini";
    score = 0;
}

public static void loadUSerInfos() // Load the user's information from the ini file. If the file doesn't exist, create it.
{
    bool fileExists = File.Exists(User.ini); // If the user file exist its true
    List<string> tempListUserData = new List<string>();
    try //Reads the user file if it exists
    {
        StreamReader sr = new StreamReader(User.ini);
        string line = sr.ReadLine();
        while (line != null)
        {
            tempListUserData.Add(line);
            line = sr.ReadLine();
        }
        sr.Close();
    }
    catch (Exception)
    {

    }

    if (fileExists) // Set the user infos into the User class if the user file exist
    {
        gameMode = tempListUserData[1].Substring(11);
```

```
      string[] temp = tempListUserData[2].Substring(12).Split(';');

      for (int i = 0; i < temp.Length - 1; i++)
      {
        wordTypes.Add(temp[i]);
      }
      allTimeScore = int.Parse(tempListUserData[6].Substring(15));
      highScore = int.Parse(tempListUserData[7].Substring(12));
      lowScore = int.Parse(tempListUserData[8].Substring(11));
      gamesPlayed = int.Parse(tempListUserData[9].Substring(14));
      gamesWon = int.Parse(tempListUserData[10].Substring(11));
      gamesLost = int.Parse(tempListUserData[11].Substring(12));
    }
    else // Creates a new user file, if the user file does not exists
    {
      gameMode = "Normal";
      wordTypes.Add("All");
      allTimeScore = 0;
      highScore = 0;
      lowScore = 0;
      gamesPlayed = 0;
      gamesWon = 0;
      gamesLost = 0;
      User.saveUserInfos();
    }
  }

  public static void saveUserInfos() // Save the user's information to the ini file.
  {
    StreamWriter sw = new StreamWriter(User.ini);
    sw.WriteLine("Username: " + User.userName);
```

```
        sw.WriteLine("Game Mode: " + User.gameMode);
        string temp = "";
        for (int i = 0; i < User.wordTypes.Count; i++)
        {
            temp += User.wordTypes[i] + "; ";
        }
        sw.WriteLine("Word Types: " + temp);
        sw.WriteLine("\n");
        sw.WriteLine("User Stats:");
        sw.WriteLine("All Time Score: " + User.allTimeScore);
        sw.WriteLine("High Score: " + User.highScore);
        sw.WriteLine("Low Score: " + User.lowScore);
        sw.WriteLine("Games Played: " + User.gamesPlayed);
        sw.WriteLine("Games Won: " + User.gamesWon);
        sw.WriteLine("Games Lost: " + User.gamesLost);
        sw.Close();
    }
}

public class rankingStats // The class for the values of the rankings
{
    public string userName { get; set; }
    public int allTimeScore { get; set; }
    public int highScore { get; set; }
    public int gamesPlayed { get; set; }
    public int gamesWon { get; set; }
    public int winRate { get; set; }

    public rankingStats(string UserName, int AllTimeScore, int HighScore, int GamesPlayed, int
GamesWon, int WinRate)
    {
```

```
        userName = UserName;
        allTimeScore = AllTimeScore;
        highScore = HighScore;
        gamesPlayed = GamesPlayed;
        gamesWon = GamesWon;
        winRate = WinRate;
    }
}

internal class Program
{
    public static bool exit; // It is false if the user wants to log out

    public static List<rankingStats> listRankingStats = new List<rankingStats>(); // Created a ranking list from the Ranking class (all best stats will be loaded here for the users)

    public static void loadRankings() //Loads the best stats for every user
    {
        try
        {
            StreamReader sr = new StreamReader("ranking.txt");
            string line = sr.ReadLine();
            while (line != null) // Add the best stats to the listRankingStats list
            {
                string[] temp = line.Split(';');
                listRankingStats.Add(new rankingStats(temp[0], int.Parse(temp[1]), int.Parse(temp[2]), int.Parse(temp[3]), int.Parse(temp[4]), int.Parse(temp[5])));
                line = sr.ReadLine();
            }
            sr.Close();
        }
```

```
        catch (Exception) // Loads the current user's stats to the listRankingStats if the ranking.txt file does not exists
        {
            int temp = (User.gamesWon * 100) / User.gamesPlayed;
            listRankingStats.Add(new rankingStats(User.userName, User.allTimeScore, User.highScore, User.gamesPlayed, User.gamesWon, temp));
        }
    }

    public static void updateRanking() // Updates the listRankingStats for the current user
    {
        bool isInList = false; // it is false, if the user is not in the listRankingStats yet
        for (int i = 0; i < listRankingStats.Count; i++)
        {
            if (listRankingStats[i].userName == User.userName) // Finds the current user in the listRankingStats and change the stats for tha latest
            {
                listRankingStats[i].allTimeScore = User.allTimeScore;
                listRankingStats[i].highScore = User.highScore;
                listRankingStats[i].gamesPlayed = User.gamesPlayed;
                listRankingStats[i].gamesWon = User.gamesWon;
                listRankingStats[i].winRate = (User.gamesWon * 100) / User.gamesPlayed;
                isInList = true;
                break;
            }
        }
        if (!isInList) // Loads the current user's stats to the listRankingStats if the ranking.txt file does not exists
        {
            int temp = (User.gamesWon * 100) / User.gamesPlayed;
            listRankingStats.Add(new rankingStats(User.userName, User.allTimeScore, User.highScore, User.gamesPlayed, User.gamesWon, temp));
        }
```

```
        saveRanking(listRankingStats);
    }

    public static void saveRanking(List<rankingStats> listRankingStats) // Saves the listRankingStats to the ranking.txt file
    {
        string[] tempArray = new string[listRankingStats.Count()];
        for (int i = 0; i < listRankingStats.Count(); i++)
        {
            string temp = listRankingStats[i].userName + ";" + listRankingStats[i].allTimeScore + ";" + listRankingStats[i].highScore + ";" + listRankingStats[i].gamesPlayed + ";" + listRankingStats[i].gamesWon + ";" + listRankingStats[i].winRate;
            tempArray[i] = temp;
        }
        File.WriteAllLines("ranking.txt", tempArray);
    }

    public static void changeGameMode() // Changes the game mode
    {
        // Change the game mode.
        Console.Clear();
        Console.WriteLine("Change Game Mode\n");
        Console.WriteLine("Please enter the game mode you want to play.\n1. Easy, 2. Normal, 3. Hard, 4. Back");
        int choice;
        int.TryParse(Console.ReadLine(), out choice);
        switch (choice) // Saves the user infos and goes back to the settings
        {
            case 1:
                User.gameMode = "Easy";
```

```
            User.saveUserInfos();
            settings();
            break;
        case 2:
            User.gameMode = "Normal";
            User.saveUserInfos();
            settings();
            break;
        case 3:
            User.gameMode = "Hard";
            User.saveUserInfos();
            settings();
            break;
        case 4:
            settings();
            break;
        default:
            changeGameMode();
            break;
    }
}

    public static List<string> wordTypeList = new List<string>(); // Contains the available word types

    public static void loadWordTypes() // Loads the word types from the wordTypes.txt
    {
        wordTypeList.Clear();
        StreamReader sr = new StreamReader("wordTypes.txt");
        string line = sr.ReadLine();
        while (line != null)
```

```
        {
            wordTypeList.Add(line);
            line = sr.ReadLine();
        }
    }

    public static void changeWordTypes() // Changes the word type for the current user
    {
        Console.Clear();
        Console.WriteLine("Change Word Types\n");
        loadWordTypes();
        string userWordtypes = ""; //Put the current user's word types into a sting
        for (int i = 0; i < User.wordTypes.Count(); i++)
        {
            if (User.wordTypes.Count() == i + 1)
            {
                userWordtypes += User.wordTypes[i];
            }
            else
            {
                userWordtypes += User.wordTypes[i] + ", ";
            }
        }
        Console.WriteLine("Current word types: " + userWordtypes);

        Console.WriteLine("Please enter the word types you want to play.");
        string allWordTypes = ""; //Gather all the available word types into a string (allWordTypes) and writes it out to the console
        for (int i = 0; i < wordTypeList.Count(); i++)
        {
            if (i != wordTypeList.Count()-1)
```

```
            {
                allWordTypes += (i + 1) + ". " + wordTypeList[i] + ", ";
            } else
            {
                allWordTypes += (i + 1) + ". " + wordTypeList[i] + ", " + (i+2) + ". back";
            }
        }
        Console.WriteLine(allWordTypes);
        int choice;
        if (int.TryParse(Console.ReadLine(), out choice)) // Checks all the possibilities for the word type changes (if the choise is an int)
        {
            if (choice == wordTypeList.Count()+1) // If chanes the "back"
            {
                settings();
            }else if(choice > wordTypeList.Count()+1 || choice <= 0) // If choice is not an available number
            {
                changeWordTypes();
            }
            else if ((User.wordTypes.Count() == wordTypeList.Count()-1 && !User.wordTypes.Contains(wordTypeList[choice-1])))
            {
                //If the last not used word type is added
                User.wordTypes.Clear();
                User.wordTypes.Add("All");
                User.saveUserInfos();
                changeWordTypes();
            }
            else if ((User.wordTypes.Count() == 1 && User.wordTypes.Contains(wordTypeList[choice - 1])) && (User.wordTypes.Count() == 1 && User.wordTypes[0] != "All"))
            {
```

```
            //If removes the only set word type it set all the word types on
            User.wordTypes.Clear();
            User.wordTypes.Add("All");
            User.saveUserInfos();
            changeWordTypes();
        }
        else if (User.wordTypes[0] == "All") // If all the word types are set, it removes only the
choosen one
        {
            User.wordTypes.Clear();
            for (int i = 0; i < wordTypeList.Count(); i++)
            {
                User.wordTypes.Add(wordTypeList[i]);
            }
            User.wordTypes.Remove(wordTypeList[choice - 1]);
            User.saveUserInfos();
            changeWordTypes();
        }
        else if (User.wordTypes.Contains(wordTypeList[choice-1])) // Removes the choosen word
type
        {
            User.wordTypes.Remove(wordTypeList[choice - 1]);
            User.saveUserInfos();
            changeWordTypes();
        }
        else // Add the choosen word type
        {
            User.wordTypes.Add(wordTypeList[choice - 1]);
            User.saveUserInfos();
            changeWordTypes();
        }
    }
```

```
    else // Calls the changeWordTypes again, when the choise is not an int
    {
        changeWordTypes();
    }

}

public static void settings() // Settings menu
{
    Console.Clear();
    Console.Title = "Hangman - Settings";
    Console.WriteLine("Settings\n");
    Console.WriteLine("1. Change Game Mode\n2. Change Word Types\n3. Back");
    int choice;
    int.TryParse(Console.ReadLine(), out choice);
    switch (choice)
    {
        case 1:
            changeGameMode();
            break;
        case 2:
            changeWordTypes();
            break;
        case 3:
            menu();
            break;
        default:
            settings();
            break;
    }
```

```
    }

    public static void ranking() // Ranking
    {
        Console.Clear();
        Console.Title = "Hangman - Ranking";
        Console.WriteLine("Ranking\n");
        Console.WriteLine("1. All Time Score\n2. High Score\n3. Games Played\n4. Games
Won\n5. Win Rate\n6. Back");
        int choice;
        int.TryParse(Console.ReadLine(), out choice);
        updateRanking();
        switch (choice) //Write out the rank lists based on the users's choice
        {
            case 1:
                Console.Clear();
                List<rankingStats> allTimeScoreList = listRankingStats;
                for (int i = 0; i < allTimeScoreList.Count(); i++)
                {
                    rankingStats tempRankingStatsElement;
                    for (int j = 1; j < allTimeScoreList.Count(); j++)
                    {
                        if (allTimeScoreList[j-1].allTimeScore < allTimeScoreList[j].allTimeScore)
                        {
                            tempRankingStatsElement = allTimeScoreList[j - 1];
                            allTimeScoreList[j - 1] = allTimeScoreList[j];
                            allTimeScoreList[j] = tempRankingStatsElement;
                        }
                    }
                }
                Console.WriteLine("Name\tAll Time Score");
```

```
            for (int i = 0; i < allTimeScoreList.Count(); i++)
            {
                Console.WriteLine("{0}\t{1}", allTimeScoreList[i].userName,
allTimeScoreList[i].allTimeScore);
            }
            Console.WriteLine("\nPress any key to continue");
            Console.ReadKey();
            ranking();
            break;
          case 2:
            Console.Clear();
            List<rankingStats> highScoreList = listRankingStats;
            for (int i = 0; i < highScoreList.Count(); i++)
            {
              rankingStats tempRankingStatsElement;
              for (int j = 1; j < highScoreList.Count(); j++)
              {
                if (highScoreList[j - 1].highScore < highScoreList[j].highScore)
                {
                  tempRankingStatsElement = highScoreList[j - 1];
                  highScoreList[j - 1] = highScoreList[j];
                  highScoreList[j] = tempRankingStatsElement;
                }
              }
            }
            Console.WriteLine("Name\tHigh Score");
            for (int i = 0; i < highScoreList.Count(); i++)
            {
              Console.WriteLine("{0}\t{1}", highScoreList[i].userName, highScoreList[i].highScore);
            }
            Console.WriteLine("\n Press any key to continue");
```

```
            Console.ReadKey();
            ranking();
            break;
        case 3:
            Console.Clear();
            List<rankingStats> gamesPlayedList = listRankingStats;
            for (int i = 0; i < gamesPlayedList.Count(); i++)
            {
                rankingStats tempRankingStatsElement;
                for (int j = 1; j < gamesPlayedList.Count(); j++)
                {
                    if (gamesPlayedList[j - 1].gamesPlayed < gamesPlayedList[j].gamesPlayed)
                    {
                        tempRankingStatsElement = gamesPlayedList[j - 1];
                        gamesPlayedList[j - 1] = gamesPlayedList[j];
                        gamesPlayedList[j] = tempRankingStatsElement;
                    }
                }
            }
            Console.WriteLine("Name\tAll Games Played");
            for (int i = 0; i < gamesPlayedList.Count(); i++)
            {
                Console.WriteLine("{0}\t{1}", gamesPlayedList[i].userName,
gamesPlayedList[i].gamesPlayed);
            }
            Console.WriteLine("\nPress any key to continue");
            Console.ReadKey();
            ranking();
            break;
        case 4:
            Console.Clear();
```

```
            List<rankingStats> gamesWonList = listRankingStats;
            for (int i = 0; i < gamesWonList.Count(); i++)
            {
              rankingStats tempRankingStatsElement;
              for (int j = 1; j < gamesWonList.Count(); j++)
              {
                if (gamesWonList[j - 1].gamesWon < gamesWonList[j].gamesWon)
                {
                  tempRankingStatsElement = gamesWonList[j - 1];
                  gamesWonList[j - 1] = gamesWonList[j];
                  gamesWonList[j] = tempRankingStatsElement;
                }
              }
            }
            Console.WriteLine("Name\tGames Won");
            for (int i = 0; i < gamesWonList.Count(); i++)
            {
              Console.WriteLine("{0}\t{1}", gamesWonList[i].userName,
gamesWonList[i].gamesWon);
            }
            Console.WriteLine("\nPress any key to continue");
            Console.ReadKey();
            ranking();
            break;
          case 5:
            Console.Clear();
            List<rankingStats> winRateList = listRankingStats;
            for (int i = 0; i < winRateList.Count(); i++)
            {
              rankingStats tempRankingStatsElement;
              for (int j = 1; j < winRateList.Count(); j++)
```

```
            {
                if (winRateList[j - 1].winRate < winRateList[j].winRate)
                {
                    tempRankingStatsElement = winRateList[j - 1];
                    winRateList[j - 1] = winRateList[j];
                    winRateList[j] = tempRankingStatsElement;
                }
            }
        }
        Console.WriteLine("Name\tWin rate");
        for (int i = 0; i < winRateList.Count(); i++)
        {
            Console.WriteLine("{0}\t{1}%", winRateList[i].userName, winRateList[i].winRate);
        }
        Console.WriteLine("\nPress any key to continue");
        Console.ReadKey();
        ranking();
        break;
    case 6:
        menu();
        break;
    default:
        ranking();
        break;
  }
}

public static void startGame() // Starts the game
{
    List<char> approvedChars = new List<char> { 'a', 'á', 'b', 'c', 'd', 'e', 'é', 'f', 'g', 'h', 'i', 'í', 'j', 'k', 'l',
'm', 'n', 'o', 'ó', 'ö', 'ő', 'p', 'r', 's', 't', 'u', 'ú', 'ü', 'ű', 'v', 'z' };
```

```
Console.Clear();
Console.Title = "Hangman - Game";
Console.WriteLine("Game\n");
Console.WriteLine("Game Mode: " + User.gameMode);
Console.WriteLine("Word Types: " + string.Join(", ", User.wordTypes));
Console.WriteLine("Please enter your choice.\n1. Start Game, 2. Back");
int choice;
int.TryParse(Console.ReadLine(), out choice);
List<string> words = new List<string>();

//Have to revwrite

switch (choice)
{
   case 1:

      for (int i = 0; i < User.wordTypes.Count(); i++)
      {
         StreamReader sr = new StreamReader(User.wordTypes[i] + ".txt");
         string line = sr.ReadLine();
         while (line != null)
         {
            words.Add(line);
            line = sr.ReadLine();
         }
         sr.Close();
      }

      Random rnd = new Random();
      int index = rnd.Next(words.Count); // Chooses a random word from the provided list
      string word = words[index];
```

```
            char[] wordArray = word.ToCharArray(); // Adds the characters of the choosen word to a
character list

            char[] guessArray = new char[wordArray.Length];

            for (int i = 0; i < wordArray.Length; i++) // Creates a character list for the guesses

            {

               guessArray[i] = '_';

            }


            int lives; // Set the lives based on the game mode

            if (User.gameMode == "Easy")

            {

               lives = 10;

            }

            else if (User.gameMode == "Normal")

            {

               lives = 7;

            }

            else

            {

               lives = 5;

            }


            int incorrectGuesses = 0;


            bool gameWon = false;

            bool gameLost = false;

            List<char> guessedLetters = new List<char>(); // This contained the guessed
characters


            while (!gameWon && !gameLost)

            {
```

```
            Console.Clear();
            Console.WriteLine("Game - " + User.wordTypes[0] + "\n");
            Console.WriteLine("Word: " + string.Join(" ", guessArray));
            Console.WriteLine("Lives: " + lives);
            Console.WriteLine("Please enter your guess: ");
            char guess = Console.ReadKey().KeyChar; // REads a character from the console - for the guess
            bool correctGuess = false;
            for (int i = 0; i < wordArray.Length; i++) // Checkes if the guesed character is in the guess word and places it in the correct place
            {
              if (wordArray[i] == Char.ToLower(guess))
              {
                guessArray[i] = Char.ToLower(guess);
                correctGuess = true;
              }
            }
            bool guessArrayContains = guessedLetters.Contains(Char.ToLower(guess));
            bool containedApproved = approvedChars.Contains(Char.ToLower(guess));
            if ((!correctGuess && !guessArrayContains) && containedApproved) // Set the lives, exit if 0 lives is left and set the gameWon to True if guessed all the characters
            {
              lives--;
              incorrectGuesses++;
            }
            if (lives == 0)
            {
              gameLost = true;
            }
            if (!guessArray.Contains('_'))
            {
              gameWon = true;
```

```
    }
    guessedLetters.Add(Char.ToLower(guess));
}

if (gameWon) // If the user wins the game - set the points, saves the stats
{
    Console.Clear();
    Console.WriteLine("Game\n");
    Console.WriteLine("Word: " + string.Join(" ", guessArray));
    Console.WriteLine("Congratulations! You won the game!");
    if (User.gameMode == "Easy")
    {
        User.score = (10 - incorrectGuesses) * 10;
        User.allTimeScore += User.score;
    }
    else if (User.gameMode == "Normal")
    {
        User.score = (7 - incorrectGuesses) * 25;
        User.allTimeScore += User.score;
    }
    else
    {
        User.score = (5 - incorrectGuesses) * 50;
        User.allTimeScore += User.score;
    }
    Console.WriteLine("Your Score is: " + User.score);
    User.gamesPlayed++;
    User.gamesWon++;
    if (User.score > User.highScore)
    {
        User.highScore = User.score;
```

```
        }
        if (User.score < User.lowScore || User.lowScore == 0)
        {
            User.lowScore = User.score;
        }
        User.saveUserInfos();
        updateRanking();
        Console.WriteLine("Press any key to continue.");
        Console.ReadKey();
        menu();
    }
    else if (gameLost) // If the user lost the game - saves the stats
    {
        Console.Clear();
        Console.WriteLine("Game\n");
        Console.WriteLine("Word: " + string.Join(" ", wordArray));
        Console.WriteLine("You lost the game!");
        User.gamesPlayed++;
        User.gamesLost++;
        User.saveUserInfos();
        updateRanking();
        Console.WriteLine("Press any key to continue.");
        Console.ReadKey();
        menu();
    }
    break;
case 2:
    menu();
    break;
default:
    startGame();
```

```
            break;
        }
    }

    public static void stats() // Write out the stats
    {
        Console.Clear();
        Console.Title = "Hangman - Stats";
        Console.WriteLine("Stats - " + User.userName + "\n");
        Console.WriteLine("All Time Score: " + User.allTimeScore);
        Console.WriteLine("High Score: " + User.highScore);
        Console.WriteLine("Low Score: " + User.lowScore);
        Console.WriteLine("Games Played: " + User.gamesPlayed);
        Console.WriteLine("Games Won: " + User.gamesWon);
        Console.WriteLine("Games Lost: " + User.gamesLost);
        Console.WriteLine("Win rate: " + (User.gamesWon * 100) / User.gamesPlayed + "%");
        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
        menu();
    }

    public static void menu() // Main Menu
    {
        Console.Clear();
        Console.Title = "Hangman - Menu";
        Console.WriteLine("Welcome " + User.userName);
        Console.WriteLine("\nMenu");
        Console.WriteLine("\nPlease enter your choice.\n1. Start Game, 2. Settings, 3. Stats, 4.
Ranking, 5. Log out, 6. Exit");
        int choice;
        int.TryParse(Console.ReadLine(), out choice);
```

```
switch (choice)
{
    case 1:
        startGame();
        break;
    case 2:
        settings();
        break;
    case 3:
        stats();
        break;
    case 4:
        ranking();
        break;
    case 5:
        Console.Clear();
        Console.WriteLine("You are now logged out.");
        Console.WriteLine("Thank you for playing with my game.");
        Console.WriteLine("We are looking forward to seeing you again.\n");
        Console.WriteLine("\nPress any key to continue\n");
        Console.ReadKey();
        exit = false;
        break;
    case 6:
        Console.Clear();
        Console.WriteLine("Thank you for playing with my game.");
        Console.WriteLine("We are looking forward to seeing you again.\n");
        Console.WriteLine("Created by BPatrik");
        Console.WriteLine("\nPress any key to terminate the app\n");
        Console.ReadKey();
```

```
            Environment.Exit(0);
            break;
        default:
            menu();
            break;
    }

  }

  static void Main(string[] args) // Main program
  {
    do
    {
      exit = true;
      string temporaryInput;
      Console.Clear();
      Console.Title = "Hangman";

      Console.WriteLine("Welcome to Hangman!\n");
      Console.Write("Please enter your Username: ");
      temporaryInput = Console.ReadLine();
      Console.Clear();
      User.SetUserName(temporaryInput);
      User.loadUSerInfos();
      loadRankings();
      updateRanking();
      menu();
    }
    while (!exit);
  }
}
```

}