ASSIGNMENT-2-GAME DEVELOPMENT

Name:B.VENKATA ROSHAN

REG:211211101040

Hangman is a classic word-guessing game that involves two main elements: a secret word and a series of guesses. The objective is for the player to correctly guess the secret word within a limited number of incorrect guesses. Here's a detailed explanation of how the game works, its rules, and the typical flow:

Game Components

1. **Secret Word**:
    - This is the word that the player needs to guess. In a physical game, one player chooses the word, and in a computerized version, the word is often selected randomly from a predefined list.
2. **Guessed Word Display**:
    - The representation of the secret word with the correct guesses filled in and the remaining letters hidden. Initially, all letters are hidden, often represented by underscores (e.g., "_____" for a 5-letter word).
3. **Incorrect Guesses**:
    - A record of letters that have been guessed but are not in the secret word. These guesses typically result in a "part" of a hangman being drawn, representing the player's remaining chances.
4. **Attempts (Lives)**:
    - The number of incorrect guesses allowed before the game is lost. This is typically represented by a stick figure being drawn piece by piece, culminating in a complete figure that "hangs" when the player runs out of guesses.

Game Rules

1. **Initial Setup**:
    - A secret word is chosen, and its length is displayed with underscores.
    - The player is informed of the number of incorrect guesses (or lives) they have.
2. **Gameplay Loop**:
    - The player guesses a letter.
    - If the guessed letter is in the secret word, it is revealed in the corresponding positions in the guessed word display.
    - If the guessed letter is not in the word, it is added to the list of incorrect guesses, and one attempt is subtracted.

    - The game continues until the player either guesses the entire word or runs out of attempts.

3. **Winning Condition**:
    - The player wins if they correctly guess all the letters in the word before using all their attempts.
4. **Losing Condition**:
    - The player loses if they run out of attempts before guessing the word.

Example Gameplay

1. **Setup**:
    - Secret Word: "apple"
    - Maximum Attempts: 6
2. **Turn 1**:
    - Player guesses: "e"
    - Result: The word display changes to "____e", and there are no incorrect guesses.
3. **Turn 2**:
    - Player guesses: "o"
    - Result: Incorrect guess. Incorrect guesses list: "o", remaining attempts: 5.
4. **Turn 3**:
    - Player guesses: "p"
    - Result: The word display changes to "_pp_e", and the incorrect guesses list remains "o".
5. **Turn 4**:
    - Player guesses: "l"
    - Result: The word display changes to "_pple", incorrect guesses list remains "o".
6. **Turn 5**:
    - Player guesses: "a"
    - Result: The word display changes to "apple", player wins.

Variations and Customizations

- **Word Difficulty**: Words can vary in difficulty, with longer words or less common words making the game more challenging.
- **Hints**: The game can include hints to assist players, such as the category of the word (e.g., "fruit" for "apple").

- **Graphical Representation**: In physical or graphical versions, incorrect guesses may draw parts of a hangman (head, body, arms, legs, etc.), adding visual suspense.
- **Multiplayer**: Hangman can be played with two or more players, where one player chooses the word and the other(s) guess.

Conclusion

Hangman is a simple yet engaging game that tests a player's vocabulary and deductive reasoning. It's often used as an educational tool to enhance language skills, particularly spelling and word recognition. The game can be easily adapted for different age groups and skill levels by adjusting the word list and the number of allowed incorrect guesses.

Code:

```
using System;
using System.Collections.Generic;


namespace HangmanGame
{
   class Program
   {
      static void Main(string[] args)
      {
         // List of possible words
         List<string> wordList = new List<string> { "apple", "banana", "orange", "grape", "strawberry" };
         Random rand = new Random();
         string secretWord = wordList[rand.Next(wordList.Count)];
         char[] guessedWord = new string('_', secretWord.Length).ToCharArray();
         List<char> incorrectGuesses = new List<char>();
         int maxAttempts = 6;
         int attempts = 0;
         bool isWordGuessed = false;
```

```
        Console.WriteLine("Welcome to Hangman!");

        while (attempts < maxAttempts && !isWordGuessed)
        {
            Console.WriteLine("\nGuessed word: " + new string(guessedWord));
            Console.WriteLine("Incorrect guesses: " + string.Join(", ",
incorrectGuesses));
            Console.WriteLine($"Attempts remaining: {maxAttempts - attempts}");
            Console.Write("Enter a letter: ");
            char guess = char.ToLower(Console.ReadKey().KeyChar);
            Console.WriteLine();

            if (secretWord.Contains(guess))
            {
                for (int i = 0; i < secretWord.Length; i++)
                {
                    if (secretWord[i] == guess)
                    {
                        guessedWord[i] = guess;
                    }
                }
            }
            else
            {
                if (!incorrectGuesses.Contains(guess))
                {
```

```
                incorrectGuesses.Add(guess);
                attempts++;
            }
            else
            {
                Console.WriteLine("You already guessed that letter.");
            }
        }

        isWordGuessed = !new string(guessedWord).Contains('_');
    }

    if (isWordGuessed)
    {
        Console.WriteLine("\nCongratulations! You've guessed the word: " +
secretWord);
    }
    else
    {
        Console.WriteLine("\nSorry, you've run out of attempts. The word
was: " + secretWord);
    }

    Console.WriteLine("Thank you for playing Hangman!");
  }
 }
}
```