**Description of Game**

1. **Problem Definition :**

Design and implement a Hangman game using a client-server architecture. The game should allow multiple players to connect and play simultaneously.

1. **Motivation of Game**

Playing Hangman can also help to improve cognitive skills such as problem-solving, critical thinking, and decision-making. It can also be a fun way to pass the time, especially when waiting in line or commuting.

From an educational perspective, Hangman can be a useful tool for teachers to help students learn and reinforce new vocabulary words and spelling rules.

1. **Game Elements**

* **Secret word or phrase:** The word or phrase that the player(s) must guesses
* **Letters:** The individual letters of the alphabet that players can guess.
* **Hidden word display:** The representation of the secret word or phrase, with blanks or underscores for unguessed letters.
* **Incorrect guesses:** The letters guessed by the player(s) that are not present in the secret word or phrase
* **Correct guesses:** The letters guessed by the player(s) that are present in the secret word or phrase.

1. **Game Requirements**

* **Word Bank:** The game should have a collection of words or phrases from which the secret word or phrase is randomly selected. The word bank should be diverse, with varying difficulty levels and categories.
* **Game Modes:** The game should support both single-player and multiplayer modes. In single-player mode, the player plays against the computer-generated word. In multiplayer mode, multiple players can take turns guessing the word or phrase.
* **Input Mechanism:** The game should provide a way for players to input their guesses, typically by selecting letters from a virtual keyboard or by typing them manually.

**Game Design**

1. **Define the rules and objectives of the game**

* The objective of the game is for the player to guess the hidden word or phrase before the hangman is fully drawn.
* The player has a limited number of attempts to guess letters in the word or phrase.
* Each incorrect guess results in the addition of a body part to the hangman.
* If the player correctly guesses the word or phrase, they win. If the hangman is fully drawn, they lose.
* Choose a list of words or phrases to use in the game:
* Select a list of words or phrases to use as the hidden words or phrases in the game.
* The words or phrases should be varied in length and difficulty.

1. **Create a function to select a random word or phrase from the list:**

* Create a function to check if the game has been won or lost:
* Create a function that checks if the player has correctly guessed the hidden word or phrase or if the hangman is fully drawn.
* If the game has been won or lost, display a message indicating the outcome and prompt the player to play again.
* Combine the functions and run the game:
* Prompt the player to enter their guesses and update the display and game state accordingly.
* Once the game has been won or lost, prompt the player to play again or exit the game

**Implementation**

**HangmanInterface.java**

import java.rmi.\*;

import java.rmi.server.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

import java.util.Scanner;

// Define the remote interface

interface HangmanInterface extends Remote {

    String startGame() throws RemoteException;

    String guessLetter(char letter) throws RemoteException;

}

**HangmanImpl.java**

import java.rmi.\*;

import java.rmi.server.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

import java.util.Scanner;

//Implement the remote interface

class HangmanImpl extends UnicastRemoteObject implements HangmanInterface {

    private List<String> words;

    private String word;

    private String guessedWord;

    private int attemptsLeft;

    public HangmanImpl() throws RemoteException {

        words = new ArrayList<>();

        words.add("hangman");

        words.add("computer");

        words.add("programming");

        // Add more words here if needed

        word = "";

        guessedWord = "";

        attemptsLeft = 6;

        selectRandomWord();

    }

    private void selectRandomWord() {

        Random random = new Random();

        int index = random.nextInt(words.size());

        word = words.get(index);

    }

    public String startGame() throws RemoteException {

        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < word.length(); i++) {

            sb.append("\_");

        }

        guessedWord = sb.toString();

        return guessedWord;

    }

    public String guessLetter(char letter) throws RemoteException {

        StringBuilder sb = new StringBuilder(guessedWord);

        boolean letterFound = false;

        for (int i = 0; i < word.length(); i++) {

            if (Character.toLowerCase(word.charAt(i)) == Character.toLowerCase(letter)) {

                sb.setCharAt(i, word.charAt(i));

                letterFound = true;

            }

        }

        if (!letterFound) {

            attemptsLeft--;

        }

        guessedWord = sb.toString();

        if (guessedWord.equals(word)) {

            selectRandomWord(); // Select a new random word for the next game

            return "Congratulations! You guessed the word: " + word;

        } else if (attemptsLeft == 0) {

            selectRandomWord(); // Select a new random word for the next game

            return "Game over! The word was: " + word;

        } else {

            return "Attempts left: " + attemptsLeft + "\nGuessed word: " + guessedWord;

        }

    }

}

**HangmanServer.java**

import java.rmi.\*;

import java.rmi.server.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

import java.util.Scanner;

public class HangmanServer {

    public static void main(String[] args) {

        try {

            // Create an instance of the remote object

            HangmanInterface hangman = new HangmanImpl();

            // Bind the remote object's stub in the registry

            Naming.rebind("HangmanServer", hangman);

            System.out.println("Hangman Server is running...");

        } catch (Exception e) {

            System.err.println("Error: " + e.getMessage());

            e.printStackTrace();

        }

    }

}

**HangmanClient.java**

import java.rmi.\*;

import java.util.Scanner;

public class HangmanClient {

    public static void main(String[] args) {

        try {

            // Lookup the remote object from the registry

            HangmanInterface hangman = (HangmanInterface) Naming.lookup("rmi://localhost/HangmanServer");

            // Start the game

            String guessedWord = hangman.startGame();

            System.out.println("Hangman game started!");

            System.out.println("Guessed word: " + guessedWord);

            // Play the game

            Scanner scanner = new Scanner(System.in);

            while (true) {

                System.out.print("Enter a letter: ");

                char letter = scanner.nextLine().charAt(0);

                String result = hangman.guessLetter(letter);

                System.out.println(result);

                if (result.startsWith("Congratulations") || result.startsWith("Game over")) {

                    break;

                }

            }

            scanner.close();

        } catch (Exception e) {

            System.err.println("Error: " + e.getMessage());

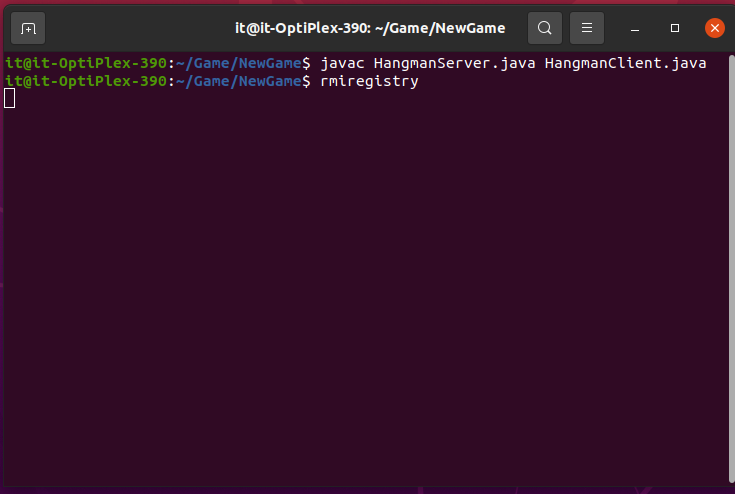
            e.printStackTrace();

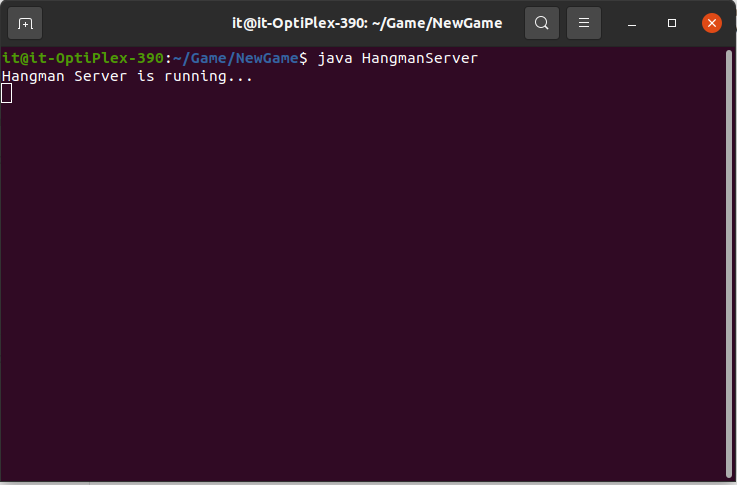
        }

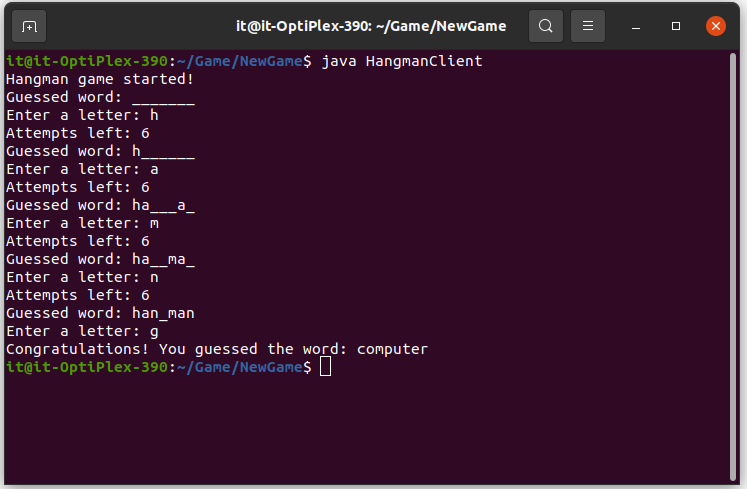
    }

}

**Output –**







**Conclusion**

In this assignment we learn to use distributed architecture to design a Distributed

Application for Interactive Multiplayer Games.

**References**

* “Distributed Systems: Concepts and Design” by George Coulouris, J Dollimore and TimKindberg, Pearson Education, ISBN: 9789332575226, 5th Edition, 2017
* “Distributed Systems”, Maarten van Steen, Andrew S. T, Third edition Version