**NATIONAL UNIVERSITY OF MODERN LANGUAGES**

FACULTY OF ENGINEERING AND COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

BSSE35-1-A-Mor

**Semester Project**

(PROGRAMMING FUNDAMENTALS)

**Project Title:**

Develop a C++ program that offers users the option to choose from collection of five different games. Upon selecting a game to play, the program executes the chosen game's logic.

**Course Instructor:**

Mam Mariyam Imtiaz

**Group Members:**

|  |  |  |
| --- | --- | --- |
| Name | Roll No | System I’d |
| Nimra Nisab | SP23869 | Numl-S24-14659 |
| Hamna Khan | SP23856 | Numl-S24-10119 |
| Aniqa Jamil | SP23868 | Numl-S24-14229 |
| Jahan Zaib | SP23870 | Numl-S24-15519 |
| Rania Nisar | SP23891 | Numl-S24-22001 |
| Zulqarnain Haider | SP23859 | Numl-S24-11357 |

**Integrated Development Environment:**

Visual Studio (C++ Language), Visual Studio Code (MinGW).

**Domain:**

Entertainment

**Session:**

Spring 2024 BSSE35-1-A-Morning

**How to run the games on your system Steps: -**

To run our games, I am using VS CODE, With Min GW compiler. To compile all files together and run it, follow the following steps: -

+--------------------- At first you don't need to compile anything ----------------------+

Simply type the following command in your terminal:

Command = ./menu

Notice: - ( It’s written as dot slash menu )

If the program doesn't run do the following steps: -

+-------- First enter the following command in your terminal----------------+

1. cd {enter here the directory of ur folder where u are running} For example: -

(/mnt/c/Coding\_Files/PF\_Games\_Project/)

g++ menu.cpp hang\_man\_HK-NM.cpp tic\_tac\_toe\_AJ.cpp number\_guessing\_ZH.cpp rock\_scissor\_RN.cpp pac\_man\_JZ-AJ.cpp -o menu

Full command will be like: -  
  
cd /mnt/c/Coding\_Files/PF\_Games\_Project/ g++ menu.cpp hang\_man\_HK-NM.cpp tic\_tac\_toe\_AJ.cpp number\_guessing\_ZH.cpp rock\_scissor\_RN.cpp pac\_man\_JZ-AJ.cpp -o menu

+--------- Run the file using this command in your terminal -------------------+

2. ./menu

3. Enjoy.

**Project Overview:**

Develop a C++ program that offers users a diverse collection of five different games: Hangman, Pacman, Number Guessing, Tic Tac Toe, and Rock Paper Scissors. This program will provide a user-friendly menu for users to select and play their preferred game. Upon selecting a game, the program will execute the chosen game's logic, allowing users to engage in an immersive gaming experience.

**Key Features:**

**- Game Menu:** A user-friendly menu that displays the five available games, allowing users to select their preferred game.

**- Game Logic Execution:** The program will execute the chosen game's logic, providing an engaging gaming experience.

**Games Collection:**

**- Hangman:** A classic word-guessing game where users attempt to guess a hidden word by suggesting letters.

**- Pacman:** A classic arcade game where the user navigates Pacman through a maze, avoiding ghosts and collecting pellets.

**- Number Guessing:** A game where users attempt to guess a randomly generated number within a specified range.

**- Tic Tac Toe:** A classic board game where users play as X or O, aiming to win by getting three in a row.

**- Rock Paper Scissors:** A classic hand game where users play against the computer, selecting one of rock, paper, or scissors to win.

**Program Requirements:**

- The program should provide a clear and user-friendly menu for game selection.

- Each game should be implemented with its respective logic and rules.

- The leaderboard feature should display previous players' scores upon exiting the game menu.

- The program should handle user input and validate user selections.

**-:Hangman:-**

**Problem Statement:**

* Select a word from a predefined list or generate a random word for the player to guess.
* Represent the word as a series of blanks and reveal letters as the player makes correct guesses.
* Implement a mechanism to track incorrect guesses and display a hangman figure accordingly.
* Create functions to handle player input, validate guesses, and update the game state.
* End the game when the word is guessed correctly or when the hangman figure is complete, and display the outcome to the player.

**Analysis:**

**Word Selection**: How to choose a word for the player to guess from a predefined list.

**Tracking Correct and Incorrect Guesses**: How to manage the state of the game by keeping track of correct letters and incorrect guesses.

**Game Logic and Termination:** How to handle game progression, including updating the visual representation of the "hangman" and determining when the game is over.

**User Interface:** How to provide a visual representation of the word and any graphical elements indicating the number of incorrect guesses allowed.

**Algorithm:**

* Define a list of words to be guessed in the game.
* Choose a random word from the list as the target word.
* Initialize a variable to store the maximum number of incorrect guesses allowed.
* Initialize a variable to store the number of incorrect guesses made.
* Initialize a string to store the current state of the word being guessed, with underscores representing unrevealed letters.
* Initialize a set to store the letters that have already been guessed.
* Start a loop that continues until the word is guessed or the maximum number of incorrect guesses is reached.
* Display the current state of the word being guessed and prompt the player to guess a letter.
* Check if the guessed letter is not a single alphabet character. If it is not, inform the player and continue to the next iteration of the loop.
* Check if the guessed letter has already been guessed. If it has, inform the player and continue to the next iteration of the loop.
* Check if the guessed letter is present in the target word. If it is, update the current state of the word to reveal the guessed letter in all appropriate positions
* If the guessed letter is not present in the target word, increment the number of incorrect guesses made. o Add the guessed letter to the set of guessed letters.
* Check if the word has been completely guessed. If it has, inform the player that they have won the game.
* Check if the number of incorrect guesses made is equal to the maximum number of incorrect guesses allowed. If it is, inform the player that they have lost the game and reveal the target word.
* End the game.

**Code Visual Representation:**

****

****

**-:Tic Tac Toe Game:-**

**Problem Statement:**

* Design a data structure to represent the game board and store player moves.
* Implement logic to check for win conditions after each player's move.
* Develop algorithms to determine the optimal move for the computer player, considering potential win scenarios and blocking the opponent.
* Create functions to handle player input and update the game board accordingly.
* Display the game board and messages to guide players through the game, including prompts to input moves and announcing the winner.

**Analysis:**

**Game Board Representation:** How to represent the 3x3 grid to allow players to make their moves.

Player Input and Validation: How to ensure players make valid moves and don't overwrite existing marks.

**Game Logic:** How to check for winning combinations or determine if the game is a tie.

**Player Interaction:** How to manage turns between two players and provide appropriate prompts or feedback.

**Algorithm:**

* Create a 3x3 grid to represent the Tic-Tac-Toe board.
* Assign a unique identifier for each player (e.g., 'X' for Player 1 and 'O' for Player 2).
* Display the empty board to the players.
* Start a loop that continues until the game is won, drawn, or ended.
* Prompt the current player to make a move by choosing a row and column on the board to place their marker.
* Check if the chosen location on the board is valid (i.e., within the bounds of the grid and not already occupied).
* If the move is valid, update the board with the player's marker at the chosen location.
* Check if the current player has won the game by achieving a horizontal, vertical, or diagonal line of their markers.
* If the game is won by the current player, display a winning message and end the game.
* If all cells on the board are filled and no player has won, declare the game as a draw and end the game.
* Switch the current player for the next turn.
* Repeat steps 5-11 until the game is won, drawn, or ended.
* End the game.

**Code Visual Representation : -**

****

****

****

**-:Number Guessing Game:-**

**Problem Statement:**

* Generate a random number within a specified range using a random number generator.
* Implement a loop to allow the player to input guesses until the correct number is guessed or the maximum number of attempts is reached.
* Compare the player's guess with the generated number and provide feedback on whether the guess is too high or too low.
* Display messages to guide the player through the game, including prompts to input guesses and informing the player of the outcome.
* Keep track of the number of attempts taken and congratulate the player upon guessing the correct number.

**Analysis:**

**Random Number Generation:** How to generate a random number within a specified range.

**Player Input and Feedback:** How to accept player guesses and provide feedback (whether the guess is too high, too low, or correct).

**Game Loops and Termination:** How to manage the game loop and end the game when the correct guess is made.

**Score or Attempts Tracking:** How to track the number of guesses or the number of attempts the player took to guess the correct number.

**Algorithm:**

* Generate a random number between a specified range
* Prompt the player to guess a number within the range. Compare the player's guess with the randomly generated number.
* Provide feedback to the player if the guess is too high, too low, or correct.
* Continue prompting the player for guesses until they guess the correct number.
* Keep track of the number of attempts taken by the player.
* Display the number of attempts taken to guess the correct number.
* End the game.

**Code Visual Representation:**

****

****

****

**-:Rock Paper Scissors Game:-**

**Problem Statement:**

* Define variables to represent the player's and computer's choices (rock, paper, or scissors).
* Implement a random number generator to determine the computer's choice.
* Develop conditional statements to compare the player's and computer's choices and determine the winner.
* Create functions to handle player input, validate choices, and display the result of each round.
* Track and update the overall score based on the outcome of each round.

**Analysis:**

**Computer Choice Simulation:** How to simulate the computer's choice, typically using randomization.

**Outcome Determination:** How to compare player and computer choices to determine the winner of each round.

**Score Keeping and Game Flow:** How to maintain the score and decide when the game ends or continues to another round.

**Player Interaction:** How to accept and validate player input and provide feedback on the results of each round.

**Algorithm:**

* Display the rules of the Rock-Paper-Scissors game to the player.
* Prompt the player to choose one of the three options: rock, paper, or scissors.
* Generate a random choice for the computer opponent.
* Compare the player's choice with the computer opponent's choice to determine the winner.
* Display the result of the game (win, lose, or draw) to the player.
* Allow the player to play multiple rounds if desired.
* Keep track of the number of wins, losses, and draws.
* Display the final score at the end of the game.
* End the game.

**Code Visual Representation: -**

****

**-:Pac-Man Game:-**

**Problem Statement:**

* Implement a game loop to continuously update the game state and handle player input.
* Create classes to represent Pac-Man, ghosts, pellets, power pellets, and the game board.
* Define collision detection mechanisms to handle interactions between Pac-Man, ghosts, and other game elements.
* Implement algorithms for ghost movement patterns, including chasing Pac-Man and scattering when scared.
* Incorporate scoring mechanisms to track points earned by eating dots, ghosts, and completing levels.

**Analysis:**

**Maze Structure**: How to represent the maze's walls, corridors, and pellets in a data structure.

**Player Movement:** How to allow the player to move through the maze, including collision detection to prevent passing through walls.

**Ghost Behavior:** How to create ghost movement patterns, possibly with varying behaviors depending on game state (e.g., random wandering, chasing Pac-Man).

**Game Mechanics**: How to implement scoring, power-ups, and game-end conditions (like losing a life if caught by a ghost).

**User Interface:** How to visually represent the maze and game elements on a console or graphical interface.

**Algorithm:**

* Create a grid-based game board for the Pac-Man game
* Place the Pac-Man character and ghosts on the game board.
* Allow the player to control the movement of Pac-Man using arrow keys (up, down, left, right).
* Implement AI for the movement of the ghosts on the game board.
* Define the rules for interaction between Pac-Man and ghosts (e.g., collision detection).
* Implement power-ups and special items on the game board.
* Keep track of the player's score based on the items collected and ghosts eaten.
* Display the game board, player score, and remaining lives during gameplay.
* End the game when the player has lost all lives or completed the objectives.

**Code Visual Representation: -**

****

****

****