Simple Number Guessing Game

Tech Ngoun Leang

Department of Information Technology, American University of Phnom Penh

Ms. Monyrath Buntoun

INF 653: Back-end Web Development

February 2, 2025

Introduction

In almost every programming language class, a number guessing game is often introduced as a practical exercise to teach control flow, loops, and error handling. To demonstrate the concepts and techniques we have covered in class, I have written a number guessing game to apply these concepts. The game involves a list of secret numbers, and the player must guess one of these numbers in order to win. The game continues prompting the player to guess until they get it right and also keeps track of the number of attempts made. As required by the assignment, the game must include concepts such as conditional statements, loops, jumping statements, and error handling. This report explains how each of the game's features works in detail and how they relate to the concepts mentioned above.

Objectives

- Develop an interactive Number Guessing Game using conditional statements, loops, jumping statements, and error handling.
- Ensure input validation and handle invalid inputs correctly.
- Use a for loop to iterate over secret numbers and a while loop for validating user input in the playAgain() function.
- Incorporate jumping statements like break and return to optimize game logic and performance.
- Provide engaging user experience with clear prompts and emojis, delays for a smooth flow, and informative feedback on player performance.

Approach

In most number guessing games, the player is asked to guess a randomly generated number within a specific range, typically between 1 and 100, and the program provides feedback on whether the guess is too high or too low to help the player win. This is usually achieved using a random number generator function, such as Math.random(), to create the secret number. The game checks if the player's guess is correct and provides feedback, while also keeping track of the number of attempts made. It then prompts the player to decide whether they want to play the game again or not (GeeksforGeeks, 2024).

My approach to implementing the game follows a similar structure but with some slight adjustments. Instead of generating a random number, I set an array of predefined secret numbers for the player to guess. This removes the randomness element and requires the player to guess one of the specific numbers in the list. The program provides feedback on whether the guess is correct or incorrect, without giving any detailed hints. Additionally, the game allows the player to choose whether to play again after completing a round.

Features

Here is a basic demonstration of my number guessing game:

1. The player is prompted to guess a number.

```
PS C:\Users\winOS\Desktop\Assignment1> node script.js

==========================================
🔥 WELCOME TO THE NUMBER GUESSING GAME! 🔥
==========================================
🎯 Try to guess one of the secret numbers!
🔢 The number is between 0 and 100.
💡 Can you guess it in the fewest number of attempts?
==========================================

Enter your guess: 20
```

2. Their input is validated.

```
Enter your guess: #

Validating Input...

🚨 Error: Invalid Input. Please enter a valid number.
Enter your guess: Hi

Validating Input...

🚨 Error: Invalid Input. Please enter a valid number.
Enter your guess: -50

Validating Input...

🚨 Error: Guess must be between 0 and 100.
Enter your guess: 999

Validating Input...

🚨 Error: Guess must be between 0 and 100.
```

3. The validated input is then compared to the secret number. If incorrect, the player is prompted again to guess until they are correct.

```
Enter your guess: 20

Validating Input...

❌ Wrong guess! Keep trying. Number of attempts tried: 1
Enter your guess: 30

Validating Input...

❌ Wrong guess! Keep trying. Number of attempts tried: 2
```

4. Once they guess correctly, the game congratulates the player.

```
Enter your guess: 50

Validating Input...
✅ You guessed correctly in 3 attempts!
🏆 Your skill level: You are good.
📜 The numbers in the list were: 50, 97, 80, 21, 76
```

5. The player is prompted if they want to start another round.

```
🔄 Do you want to play again? (y/n): y

🎮 New game! Try again.
Enter your guess: 90
```

I used some modules that we did not cover in class to help with the functionality of the game such as:

```
JS script.js > ...
1   const readline = require('readline'); // Imports the built-in readline module to handle user input from the terminal
2
3   /**
4    * Creates an interface for user input and output.
5    * @param {Object} options - Configuration options for the interface.
6    * @param {stream.Readable} options.input - The readable stream to listen to.
7    * @param {stream.Writable} options.output - The writable stream to write output to.
8    */
9   const rl = readline.createInterface({
10      input: process.stdin,  // Reads input from the terminal
11      output: process.stdout // Writes output to the terminal
12  });
```

readline Module and rl.question() Function: The readline module is a built-in Node.js module that can be imported to provide an interface for reading input from a readable stream. The createInterface() function initializes the interface, where input and output can be read and written in the terminal. I used readline to capture the user's guesses and their decision to play again. The rl.question() function is used to prompt the player for input.

```
55          let guess = parseInt(input); // Converts the input string to a number
```

parseInt() Function: The parseInt() function is used to convert the user's input and ensure that it is an integer.

```
60          // Validates the input
61          if (isNaN(guess)) {
62              throw new Error("Invalid Input. Please enter a valid number."); // Throws an error if the input is not a number
```

isNaN() Function: The isNaN() function checks whether a value is a valid number. If the user inputs anything besides a number, the game will alert and prompt the user to input a valid guess.

Here is the explanation for how the code works and what have been implemented:

```
14  let secretNumbers = [50, 97, 80, 21, 76]; // List of secret numbers to guess from
15  let guessCount = 0;  // Tracks the number of guesses made by the user
```

secretNumbers is declared as an array to contain the list of secret numbers to guess from, in this case, 50, 97, 80, 21, and 76 are the numbers. guessCount is declared as 0 to help keep track of the number of attempts made by the players.

```
/**
 * Determines the player's skill level based on their number of attempts.
 * @param {number} count - The number of guesses made by the player.
 * @returns {string} - A string representing the player's skill level.
 */
Complexity is 9 It's time to do something...
const skillLvl = count => {
    switch (true) {
        case count === 1:
            return "Magnificent! You should consider a career in fortune telling!"; // If guessed correctly on the first try
        case count >= 2 && count <= 5:
            return "You are good."; // If guessed correctly within 2 to 5 attempts
        default:
            return "You need to play something else..."; // If guessed correctly after more than 5 attempts
    }
}
```

The skillLvl() function accepts one parameter, count, which can take the global variable guessCount as an argument. It then uses a switch case, a type of conditional statement, to determine how well the player has performed. In case 1, if the player guesses correctly on the first try, the highest-tiered message will be returned. In case 2, if the player guesses correctly within 2 to 5 attempts, a mid-tiered message will be returned. For the last case, or the default case, a low-tiered message will be returned if the player guesses incorrectly more than 5 times.

```
/**
 * Formats and returns a string of secret numbers for display.
 * @param {number[]} array - The array of secret numbers.
 * @returns {string} - A formatted string of secret numbers separated by commas.
 */
Complexity is 4 Everything is cool!
const printAnswer = array => {
    let result = ''; // Initializes an empty string to store the secret numbers for display
    array.forEach((num, index) => {
        result += num; // Appends the current secret number to the result string
        if (index < array.length - 1)
            result += ', ';  // Adds commas between the secret numbers
    });
    return result; // Returns the string of secret numbers
}
```

The printAnswer() function returns the list of secret numbers and is used at the end to display the secret numbers for the player. It accepts one parameter, array, which corresponds to the global array secretNumbers and can be passed as an argument. The variable result is initialized as

an empty string to store each of the secret numbers. The array is then iterated using the forEach function, a higher-order function, to append each number in the array to the result variable. printAnswer also formats the string to display the result in a more acceptable format by concatenating commas between the numbers in the array.

```
const askGuess = () => {
    Complexity is 14 You must be kidding
    rl.question("Enter your guess: ", (input) => {
        let guess = parseInt(input); // Converts the input string to a number

        try {
            console.log("\nValidating Input..."); // Notifies the user that input validation is in progress

            // Validates the input
            if (isNaN(guess)) {
                throw new Error("Invalid Input. Please enter a valid number."); // Throws an error if the input is not a number
            }
            while(guess < 0 || guess > 100){
                throw new Error("Guess must be between 0 and 100."); // Throws an error if the input is not within the range of 0 or 100
            }

            guessCount++;  // Increments the guess count since the input is valid

            let isCorrect = false; // Tracks if the guess is correct

            // Checks if the guessed number is in the list
            for (let i = 0; i < secretNumbers.length; i++) {
                if (secretNumbers[i] === guess) {
                    isCorrect = true; // Sets to true if the guess matches a number in the list
                    break; // Stop checking if a match is found
                }
            }

            // If the guess is correct, prints success, tracks performance, and asks to play again
            if (isCorrect) {
                setTimeout(() => {
                    console.log(`✅ You guessed correctly in ${guessCount} attempts!
🏆 Your skill level: ${skillLvl(guessCount)}
📜 The numbers in the list were: ${printAnswer(secretNumbers)}`);
                    playAgain();
                }, 2000);
            } else {
                setTimeout(() => console.log(`\n❌ Wrong guess! Keep trying. Number of attempts tried: ${guessCount}`), 2000);
                setTimeout(askGuess, 3000); // Prompts the user again after 3 seconds
            }
        } catch (error) {
            console.log("\n🚨 Error: " + error.message);
            askGuess(); // Prompts again if the input was invalid
        }
    });
};
```

The askGuess() function prompts the user to input their guess and performs several steps to validate and process the input. It uses the rl.question method to get the player's guess, then parses it into an integer using parseInt. Inside the function, a try-catch block and some if-else statements (conditional statements) are used for error handling.

```
57          try {
58              console.log("\nValidating Input..."); // Notifies the user that input validation is in progress
59
60              // Validates the input
61              if (isNaN(guess)) {
62                  throw new Error("Invalid Input. Please enter a valid number."); // Throws an error if the input is not a number
63              }
64              while(guess < 0 || guess > 100){
65                  throw new Error("Guess must be between 0 and 100."); // Throws an error if the input is not within the range of 0 or 100
66              }
67
68              guessCount++;  // Increments the guess count since the input is valid
69
70              let isCorrect = false; // Tracks if the guess is correct
71
72              // Checks if the guessed number is in the list
73              for (let i = 0; i < secretNumbers.length; i++) {
74                  if (secretNumbers[i] === guess) {
75                      isCorrect = true; // Sets to true if the guess matches a number in the list
76                      break; // Stop checking if a match is found
77                  }
78              }
79
80              // If the guess is correct, prints success, tracks performance, and asks to play again
81              if (isCorrect) {
82                  setTimeout(() => {
83                      console.log(`✅ You guessed correctly in ${guessCount} attempts!
84  🏆 Your skill level: ${skillLvl(guessCount)}
85  📜 The numbers in the list were: ${printAnswer(secretNumbers)}`);
86                      playAgain();
87                  }, 2000);
88              } else {
89                  setTimeout(() => console.log(`\n❌ Wrong guess! Keep trying. Number of attempts tried: ${guessCount}`), 2000);
90                  setTimeout(askGuess, 3000); // Prompts the user again after 3 seconds
91              }
```

In the try block, a message is printed to notify the player that input validation is in progress, enhancing the user experience. The function then validates the input using isNaN. If the input is not a valid number, it throws an error with a message that is defined accordingly. It also checks if the number is within the range of 0 to 100 using the while loop, then throws its specific message if the guess is not within that range.

Once the input is validated, the guessCount is incremented to track the number of attempts. The isCorrect variable is initialized as false to keep track of whether the guess is correct. A for loop is then used to iterate through the secretNumbers array, and an if-else statement checks if the guess matches any number in the list. If it matches, isCorrect is set to true, and the break statement (a jumping statement) is used to exit the loop.

When the player guesses correctly, a success message is displayed after a 2-second delay using the setTimeout() function. The message includes the number of attempts, the player's skill

level (determined by the skillLvl() function), and the list of secret numbers (formatted by the printAnswer() function). If the player guesses incorrectly, a 2-second delayed message is shown, notifying them to try again. The function then prompts the player again by calling askGuess() after a 3-second delay.

```
92          } catch (error) {
93              console.log("\n🚨 Error: " + error.message);
94              askGuess(); // Prompts again if the input was invalid
95          }
```

In the catch block, any errors thrown in the try block are caught. The error message is logged, and the player is prompted again to make a valid guess by calling the askGuess() function.

```
103 const playAgain = () => {
        Complexity is 3 Everything is cool!
104     rl.question("\n🔄 Do you want to play again? (y/n): ", answer => {
105         answer = answer.toLowerCase();
106         if (answer === "y") {
107             guessCount = 0; // Resets attempt counter
108             console.log("\n🎮 New game! Try again.");
109             askGuess(); // Restarts the game
110         } else if (answer === "n") {
111             console.log("\n👋 Thanks for playing! Have a nice day!");
112             rl.close(); // Closes the interface
113         } else {
114             console.log("\n❌ Invalid input. Please enter 'y' or 'n'.");
115             playAgain(); // Re-ask the question if input is invalid
116         }
117     });
118 };
```

The playAgain() function allows the player to choose whether to play again. It uses rl.question() to prompt the player.

The. toLowerCase() function ensures consistent processing by converting any capitalized input into lowercase. If the player enters 'y', the guessCount is reset to 0, and the game restarts by calling the askGuess() function. If the player decides to stop playing by entering 'n', the game logs a thank-you message and closes the terminal interface.

If the player provides anything other than 'y' or 'n', the loop continues prompting them until a valid response is received.

```
127  console.log(`
128  ============================================
129  🔥 WELCOME TO THE NUMBER GUESSING GAME! 🔥
130  ============================================
131  🎯 Try to guess one of the secret numbers!
132  🔢 The number is between 0 and 100.
133  💡 Can you guess it in the fewest number of attempts?
134  ============================================
135  `); // Prints the welcome message
136
137  askGuess(); // Starts the game by calling the askGuess function
```

This section prints the welcome message of the game to the terminal and starts the game by calling the askGuess() function.

Improvements after Feedback

```
13  const skillLvl = count => count === 1 ? "Magnificent! You should consider a career in fortune telling!" : count <= 5 ? "You are good." : "You need to play something else...";
14
```

The skillLvl() function has been switched to use a ternary operator, and the conditions have been rewritten for better readability.

```
20  const printAnswer = array => array.join(', ');
```

Instead of using a for loop, the printAnswer() function now uses the .join() method, reducing its code to a single line.

```
27  const askGuess = () => {
        Complexity is 7 It's time to do something...
28      rl.question("Enter your guess: ", input => { // Prompts user for input
29          let guess = parseInt(input); // Converts input string to a number
30          console.log("\nValidating Input..."); // Notifies the user that input validation is in progress
31
32          if (isNaN(guess) || guess < 0 || guess > 100) { // Validates the input
33              console.log("❌ Invalid input. Please enter a number between 0 and 100."); // Displays error message for invalid input
34              return askGuess(); // Prompts again if the input was invalid
35          }
36
37          guessCount++; // Increments the guess count since the input is valid
38          let isCorrect = secretNumbers.includes(guess); // Checks if the guessed number is in the list
39
40          setTimeout(() => {
41              if (isCorrect) { // If the guess is correct, prints success, tracks performance, and asks to play again
42                  console.log(`✅ You guessed correctly in ${guessCount} attempts! \n🏆 Your skill level: ${skillLvl(guessCount)} \n📜 The numbers in the list were: ${printAnswer(secretNumbers)}`);
43                  playAgain(); // Calls function to ask if the player wants to play again
44              } else { // If the guess is incorrect, prompts again after a delay
45                  console.log(`\n❌ Wrong guess! Keep trying. Number of attempts tried: ${guessCount}`);
46                  setTimeout(askGuess, 3000); // Prompts the user again after 3 seconds
47              }
48          }, 2000);
49      });
50  };
```

In the askGuess() function, the try-catch block has been removed. Error handling is now simplified to a single if statement with multiple conditions using logical OR (||). The .includes() method is used instead of a for loop to check if the guess is in the list.

Conclusion

In conclusion, the number guessing game has been programmed to create a fun game while staying almost true to what we have covered in class and matches the conditions set by the assignment such as having to use conditional statements, loops, the jumping statements, and error handling, with a few new additional modules.

Conditional statements play a key role in managing game flow. The askGuess() function validates user input and ensures the guessed number falls within the acceptable range before proceeding. The skillLvl() function applies a switch-case structure to evaluate player performance

based on the number of attempts taken. Additionally, The playAgain() function utilizes an if-else structure to determine whether the user wants to restart the game, exit and validate their response.

The game also effectively implements loops. The askGuess() function includes a for loop to check if the guessed number matches any in the secretNumbers array. It also uses a while loop for input validation, ensuring that only numbers within range are accepted. Furthermore, the printAnswer() function utilizes a forEach loop to format and display secret numbers.

Jumping statements like break and return are used to optimize execution. The break statement in askGuess() prevents unnecessary iterations once a correct guess is found. The return statements in functions such as skillLvl() and printAnswer() optimize execution by ensuring immediate results without unwanted checks.

Error handling enhances user experience by preventing invalid inputs from disrupting gameplay. The askGuess() function includes a try-catch block and if-else statements to detect and manage errors, ensuring that non-numeric inputs and out-of-range guesses trigger informative error messages rather than causing the game to crash. Another if-else is used in the playAgain() function to only allow for the two options listed to be entered from the player. This structured approach maintains smooth gameplay and provides clear guidance to the player.

The game features can further be improved by incorporating the traditional Math.random() to randomize the numbers inside the array, allowing for a more dynamic and unpredictable gameplay experience each time. Also, the printAnswer() function can be removed for this version of the game as it basically give away the answers for the next round, but it can be reused for the future updates. In the code, some parts can be substituted with higher-functions such as includes() and join(), making the code more readable and concise. For instance, using includes() to check if

the guessed number exists in the secret numbers array would eliminate the need for the for loop, and join() can be used to create a formatted string of secret numbers more efficiently. After your feedback, these two methods have been implemented to achieve such goals. The printAnswer() function has been reduced to a single line, along with the skillLvl() function, which uses a ternary operator instead of the switch conditions. The error-handling inside the askGuess() function is simplified to an if statement. These improvements made the code more shorter and readable.

Reference

GeeksforGeeks. (2024, September 23). *Number guessing game using JavaScript*. https://www.geeksforgeeks.org/number-guessing-game-using-javascript/