**Project Report of Number Guessing Game**

**1.1 Project Description**

The "Number Guessing Game" is a simple console-based game implemented in Java that leverages the four fundamental concepts of Object-Oriented Programming (OOP): Encapsulation, Inheritance, Polymorphism, and Abstraction. The game challenges players to guess a randomly generated number within a specified range and provides hints to assist them in the process.

**1.2 Objectives**

1. Develop a console-based game that demonstrates the principles of OOP.
2. Implement features that enhance the player's gaming experience, such as hints and retries.
3. Ensure the game is user-friendly and engaging.
4. Provide a structured and scalable codebase for future enhancements.

**1.3 Scope**

The scope of this project includes:

* Designing and implementing the game logic using Java.
* Creating a user interface for interacting with the game through the console.
* Including various hints and feedback mechanisms to guide the player.
* Offering the player an option to retry after the game ends.

**1.4 Features**

1. **Random Number Generation**: The game generates a random target number between 1 and 100 at the start of each game session. This target number is kept secret until the game ends or the player guesses it correctly.
2. **Limited Tries**: Players have a maximum of 5 attempts to guess the correct number. This constraint adds a level of challenge and excitement to the game.
3. **Hints and Feedback**:

* **Even/Odd Hint**: Provides information on whether the target number is even or odd.
* **Prime/Non-Prime Hint**: Indicates if the target number is a prime number.
* **Sum of Digits**: The game gives the sum of the digits of the target number to help narrow down guesses.
* **Proximity Hint**: Provides feedback on whether the player's guess is close to the target number.
* **Range Hint**: As the game progresses, hints include narrowing down the range where the target number lies.
* **Divisibility Hint**: Offers hints about the smallest divisor of the target number if it’s not a prime.
* **Close Guess Feedback**: Alerts the player if their guess is within 10 numbers of the target number, encouraging them to keep trying.

1. **Retry Option**: After a game session ends, players are asked if they would like to play again. This feature ensures continued engagement and replayability.
2. **User Input Handling**: The game handles user inputs robustly, ensuring that invalid entries do not crash the game and guiding users back to valid input ranges.
3. **Encapsulation of Logic**: All game logic and state are encapsulated within classes to promote reusability and maintainability.

**2.1 System Design**

The proposed system design is represented in this section with three UML diagrams.

**A screenshot of a computer

Description automatically generated2.2 Class Diagram**

**Classes:**

1. Game (Abstract Class)
   * Attributes:
     + MAX\_NUMBER: Integer (static, final) - Maximum possible number to guess.
     + MAX\_TRIES: Integer (static, final) - Maximum number of attempts allowed.
     + targetNumber: Integer - The randomly generated number to be guessed.
     + remainingTries: Integer - The number of remaining attempts.
     + random: Random - Object for generating random numbers.
     + previousGuess: Integer - The player's previous guess.
     + closeGuess: Boolean - Flag indicating if the previous guess was close.
   * Methods:
     + getGuess(): Integer - Prompts the user for input and returns their guess.

Fig 2.1: Class diagram of Number Guessing Game

* + - askToRetry(): Void - Asks the user if they want to play again.
    - sumOfDigits(int): Integer - Calculates the sum of digits of a given number.
    - isPrime(int): Boolean - Checks if a given number is prime.

1. NumberGuessingGame (Concrete Class)
   * Attributes: (None)
   * Methods:
     + playGame(): Void - The main game logic.
     + showHint(): Void - Provides hints to the player based on the remaining tries.
     + smallestDivisor(int): Integer - Finds the smallest divisor of a given number.
     + main(String[]): Static method - The entry point of the program.

Relationships:

* Inheritance: NumberGuessingGame inherits from Game. This indicates that NumberGuessingGame is a specialized version of the Game class and inherits its attributes and methods.

**2.3 Use Case Diagram**

A screen shot of a game

Description automatically generated**Actor:**

Player: Represents the person interacting with the game.

**Use Cases:**

Start Game, Make Guess, Get Hint, Win Game, Lose Game, Retry Game.

Player

Fig 2.2: Use case diagram of Number Guessing Game

* 1. **Activity Diagram**

A diagram of a flowchart

Description automatically generatedThe activity diagram illustrates the flow of control within the game.

1. **Start:** The game begins.
2. **Get User Guess:** The system prompts the user to enter their guess for the number.
3. **Evaluate Guess:**
   * The system compares the user's guess with the target number.
   * If the guess is correct:
     + **Show Win Message:** The system congratulates the player on winning the game.
     + **Ask to Retry:** The system asks the player if they want to play again.
   * If the guess is incorrect:
     + **Decrement Remaining Tries:** The system decrements the number of remaining tries.
     + **Check if Out of Tries:**
       - If the remaining tries are zero:
         * **Show Lose Message:** The system informs the player that they have run out of tries and reveals the target number.

Fig 2.3: Activity Diagram of Number Guessing Game

* + - * + **Ask to Retry:** The system asks the player if they want to play again.
      * If there are remaining tries:
        + **Show Hint:** The system provides a hint to the player based on their previous guess and the remaining tries.
        + **Get User Guess:** The system prompts the user for another guess.

1. **Ask to Retry:**
   * If the player chooses to play again:
     + **Create New Game:** The system initializes a new game with a new target number and resets the number of tries.
     + **Go to "Show Welcome Message".**
   * If the player chooses not to play again:
     + **End:** The game ends.

**2.5 Technology Stack**

* **Programming Language:** Java
* **Development Environment:** VScode, Eclipse, or any Java IDE
* **Version Control:** Git

**2.6 Implementation Plan**

**Week 1:**

* Set up the development environment.
* Create the base class Game and implement core functionalities.

**Week 2:**

* Develop the NumberGuessingGame class, inheriting from Game.
* Implement the main game logic and user input handling.

**Week 3:**

* Add hint functionalities and feedback mechanisms.
* Perform initial testing and debugging.

**Week 4:**

* Finalize the game logic and user interface.
* Conduct thorough testing and make necessary adjustments.
* Prepare documentation and user guide.

**3.1 Expected Outcome**

1. A fully functional console-based number guessing game.
2. Demonstration of OOP principles through a well-structured codebase.
3. A user-friendly game that provides a fun and engaging experience.

**3.2 Future Enhancements**

* Implement a graphical user interface (GUI) for a more interactive experience.
* Add different difficulty levels with varying ranges and attempts.
* Introduce additional hint types to increase game complexity.
* Create a high score tracking system to encourage replayability.

**3.3 Timeline (4 weeks)**

* **Week 1:** Set up environment, create Game class.
* **Week 2:** Develop NumberGuessingGame class, implement game logic.
* **Week 3:** Add hints and feedback, initial testing.
* **Week 4:** Finalize game, thorough testing, documentation.

**3.4 Conclusion**

The "Number Guessing Game" project not only serves as an educational tool for understanding OOP concepts but also provides an engaging game experience. With a structured implementation plan and clear objectives, this project is poised to be both an enjoyable and informative endeavor.