Number Guessing Game in RUST

Overview

This document describes the implementation of a Number Guessing Game written in Rust. The application generates a random secret number between 0 and 100. The user is prompted to guess this number, and the program provides feedback regarding whether the guess is too high, too low, or correct. The game continues until the user successfully guesses the secret number.

Components

1. Imports

```
use std::io;
use rand::Rng;
use std::cmp::Ordering;
```

- `std::io`: Provides input and output functionality, including reading from standard input.
- `rand::Rng`: Facilitates random number generation through the `Rng` trait.
- `std::cmp::Ordering`: Contains the `Ordering` enum used for comparing values.

2. External Crate

```
extern crate rand
```

Purpose: Declares the use of the `rand` crate, which is utilized for generating random numbers within the program.

3. Function: `input`

```
fn input(secret_num: i32) -> (i32, i32) {
    println!("Hey there, are you ready to give the input
...!");

    loop {
        let mut guess = String::new();
        io::stdin().read_line(&mut guess).expect("Failed to r
ead line");

        let guess: i32 = match guess.trim().parse() {
            Ok(num) => num,
            Err(_) => continue,
        };

        println!("Your guess is: {}", guess);

        if game(guess, secret_num) {
            return (guess, secret_num);
        }
    }
}
```

Purpose: Handles user input for the guessing game, processes the input, and checks if the user's guess matches the secret number.

- **Parameters**:
  - `secret_num` (`i32`): The randomly generated number to be guessed.
- **Returns**: A tuple `(i32, i32)`:
  - The first element represents the user's guessed number.
  - The second element is the secret number.
- **Functionality**:
  - Prompts the user to input a guess.

- Reads and processes the input, converting it to an integer and handling any errors.
- Invokes the `game` function to compare the guess with the secret number.
- Repeats the process until the correct guess is made.

4. Function: `game`

```
fn game(guess: i32, secret_num: i32) -> bool {
    match guess.cmp(&secret_num) {
        Ordering::Less => {
            println!("Too small");
            false
        }
        Ordering::Greater => {
            println!("Too big");
            false
        }
        Ordering::Equal => {
            println!("Hurray! You got it!");
            true
        }
    }
}
```

- **Purpose**: Compares the user's guess with the secret number and provides appropriate feedback.
- **Parameters**:
  - `guess` (`i32`): The number guessed by the user.
  - `secret_num` (`i32`): The secret number to be compared against.
- **Returns**: A boolean value indicating the result of the comparison:
  - `true` if the guess matches the secret number.
  - `false` otherwise.

- **Functionality**:
  - Compares the `guess` with the `secret_num` using the `cmp` method.
  - Prints feedback based on whether the guess is less than, greater than, or equal to the secret number.

5. Function: `main`

```
fn main() {
    let secret_num = rand::thread_rng().gen_range(0..=100);
    input(secret_num);
}
```

- **Purpose**: Serves as the entry point for the program, initializing the secret number and starting the guessing game.
- **Functionality**:
  - Generates a random integer between 0 and 100 (inclusive) using `rand::thread_rng().gen_range`.
  - Calls the `input` function with the generated secret number to commence the game.

Usage

1. **Execution**: Run the program within a Rust environment.
2. **User Interaction**: Provide guesses as prompted by the application.
3. **Feedback**: The program will inform the user whether their guess is too high, too low, or correct.
4. **Termination**: The game concludes when the user successfully guesses the secret number.