Rust Guessing Game Tutorial

Setting Up a New Project

Create a new binary project using Cargo.

```bash
cargo new guessing_game
cd guessing_game
```

- `cargo new guessing_game`: Creates a new project named `guessing_game`
- `cd guessing_game`: Changes into the project's directory

Cargo.toml (Manifest File)

This file contains your project's configuration.

```toml
[package]
name = "guessing_game"
version = "0.1.0"
edition = "2024"

[dependencies]
```

- `[package]`: A section for package configuration
- `[dependencies]`: Lists the external crates your project needs

src/main.rs (Initial Source Code)

Cargo generates a "Hello, world!" program to start.

Filename: `src/main.rs`

```rust
fn main() {
    println!("Hello, world!");
}
```

Run the program to see the output:

```
bash

cargo run
```

Processing a Guess

Modify the code to ask for user input and print it back.

Filename: `src/main.rs`

```
rust

use std::io;

fn main() {
    println!("Guess the number!");

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin()
        .read_line(&mut guess)
        .expect("Failed to read line");

    println!("You guessed: {guess}");
}
```

Code Breakdown

- `use std::io;`: Imports the io (input/output) library from the standard library (std)
- `let mut guess = String::new();`: Declares a mutable variable `guess` and binds it to a new, empty String
- `io::stdin()`: Returns a handle to the standard input for your terminal
- `.read_line(&mut guess)`: Reads a line from standard input and appends it into the guess string
- `.expect(...)`: Handles potential errors from `read_line`. If an error occurs, the program will crash and show the message
- `println!("You guessed: {guess}");`: Prints the user's input. `{guess}` is a placeholder that displays the value of the guess variable

Generating a Secret Number

First, add the `rand` crate as a dependency in `Cargo.toml`.

Filename: `Cargo.toml`

```toml
[package]
name = "guessing_game"
version = "0.1.0"
edition = "2024"

[dependencies]
rand = "0.8.5"
```

Run `cargo build` to download the new crate. Now, update the source code to use it.

Filename: `src/main.rs`

```rust
use std::io;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1..=100);

    println!("The secret number is: {secret_number}");

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin()
        .read_line(&mut guess)
        .expect("Failed to read line");

    println!("You guessed: {guess}");
}
```

Code Breakdown

- `use rand::Rng;`: Imports the Rng trait, which provides methods for random number generators
- `let secret_number = ...`: Declares a new variable `secret_number`
- `rand::thread_rng()`: Gets a random number generator that's local to the current thread

- `.gen_range(1..=100)`: Generates a random number. `1..=100` is a range expression inclusive of 1 and 100

Comparing the Guess to the Secret Number

Now, let's compare the user's guess to the secret number and convert the input string to a number.

Filename: `src/main.rs`

rust

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

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1..=100);

    println!("The secret number is: {secret_number}");

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin()
        .read_line(&mut guess)
        .expect("Failed to read line");

    let guess: u32 = guess.trim().parse().expect("Please type a number!");

    println!("You guessed: {guess}");

    match guess.cmp(&secret_number) {
        Ordering::Less => println!("Too small!"),
        Ordering::Greater => println!("Too big!"),
        Ordering::Equal => println!("You win!"),
    }
}
```

Code Breakdown

- `use std::cmp::Ordering;`: Imports the Ordering enum, which has the variants Less, Greater, and Equal

- `let guess: u32 = ...`: This shadows the previous guess variable with a new one of type u32 (an unsigned 32-bit integer)
- `guess.trim()`: Removes any leading/trailing whitespace and the newline character
- `.parse()`: Converts the string into a number. We specify the type with `: u32`
- `match guess.cmp(&secret_number) { ... }`: The cmp method compares two values and returns an Ordering variant. The match expression then executes code based on which variant is returned

Allowing Multiple Guesses with Looping

Let's give the user more than one chance by adding a loop.

Filename: `src/main.rs`

rust

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

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1..=100);

    println!("The secret number is: {secret_number}");

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

        io::stdin()
            .read_line(&mut guess)
            .expect("Failed to read line");

        let guess: u32 = guess.trim().parse().expect("Please type a number!");

        println!("You guessed: {guess}");

        match guess.cmp(&secret_number) {
            Ordering::Less => println!("Too small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal => {
                println!("You win!");
                break;
            }
        }
    }
}
```

Code Breakdown

- `loop { ... }`: Creates an infinite loop, allowing the user to guess repeatedly
- `break;`: When the guess is correct, this statement exits the loop

Handling Invalid Input

Instead of crashing when the user enters non-numeric text, let's ignore it and ask for another guess.

Filename: `src/main.rs`

rust

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

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1..=100);

    println!("The secret number is: {secret_number}");

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

        io::stdin()
            .read_line(&mut guess)
            .expect("Failed to read line");

        let guess: u32 = match guess.trim().parse() {
            Ok(num) => num,
            Err(_) => continue,
        };

        println!("You guessed: {guess}");

        match guess.cmp(&secret_number) {
            Ordering::Less => println!("Too small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal => {
                println!("You win!");
                break;
            }
        }
    }
}
```

Code Breakdown

- `let guess: u32 = match guess.trim().parse() { ... };`: We replace `.expect()` with a match expression to handle the Result from parse

- `Ok(num) => num,`: If parse is successful, it returns an Ok value containing the number (num). This arm returns the number
- `Err(_) => continue,`: If parse fails, it returns an Err. The `_` is a catch-all for any error. `continue` tells the program to skip to the next iteration of the loop

Final Code

Finally, remove the line that prints the secret number to make it a real game.

Filename: `src/main.rs`

rust

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

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1..=100);

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

        io::stdin()
            .read_line(&mut guess)
            .expect("Failed to read line");

        let guess: u32 = match guess.trim().parse() {
            Ok(num) => num,
            Err(_) => continue,
        };

        println!("You guessed: {guess}");

        match guess.cmp(&secret_number) {
            Ordering::Less => println!("Too small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal => {
                println!("You win!");
                break;
            }
        }
    }
}
```

Running the Final Game

To run your completed guessing game:

bash

```bash
cargo run
```

Enjoy playing your Rust guessing game!