Programando um jogo de adivinhação

Diogo Silveira Mendonça

Slides Baseados no Capítulo 2 do Rust Book
https://doc.rust-lang.org/book/ch02-00-guessing-game-tutorial.html

Introdução

- Objetivo
  - Apresentar mais conceitos sobre Rust em um exemplo real de programa.
  - Mostrar como se utiliza crates externos.
  - Introduzir a alguns conceitos, como variáveis, mudança de tipo da variável, comparação de valores, loop, entre outros.

- Implementar um exemplo clássico de programação: Um jogo de adivinhação.

- Todos os exemplos podem ser encontrados nesse link.

Iniciando um novo projeto

- $ cargo new guessing_game
- $ cd guessing_game

Código inicial do projeto.

- Dentro da pasta “src”, adicionar o código ao arquivo “main.rs”.

- Link do código

```
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}");
}
```

Variáveis

- Imutáveis - Padrão
- Mutáveis - Modificador mut

```
let mut guess = String::new();
```

```
let apples = 5; // immutable
let mut bananas = 5; // mutable
```

Entrada de Dados

- Uso da função *stdin* do módulo *io*
- Uso do método *read_line*
  - “&” Indica que o argumento é uma referência a *mut guess*
  - Retorna a String e Result
    - Result pode ser Ok ou Err
- Se o método *expect* receber OK
  - Retorna o valor de OK
- Se o método expect receber Err
  - Finaliza o programa e Printa a mensagem de falha.

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

Entrada de Dados

- Se tentarmos rodar sem o “.expect("Failed to read line");”:
  - Será retornado um warning sinalizando a falta do .expect

```
$ cargo build
   Compiling guessing_game v0.1.0 (file:///projects/guessing_game)
warning: unused `Result` that must be used
  --> src/main.rs:10:5
   |
10 |   io::stdin().read_line(&mut guess);
   |   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
   |
   = note: this `Result` may be an `Err` variant, which should be handled
   = note: `#[warn(unused_must_use)]` on by default

warning: `guessing_game` (bin "guessing_game") generated 1 warning
        Finished dev [unoptimized + debuginfo] target(s) in 0.59s
```

Imprimir mensagens com placeholders

- Utilizamos chaves para delimitar um Placeholder
  - A variável pode ser chamada dentro das chaves println!(*“{x}”*)
  - Também pode ser chamada fora das chaves println!(*“{}” , y + 2*)

```
let x = 5;
let y = 10;

println!("x = {x} and y + 2 = {}", y + 2);
```

Isso vai imprimir: x = 5 and y + 2 = 12

Então

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

Vai imprimir o valor de guess

Adicionando um crate externo

- Adicionar o crate rand:
  - Modificar o Cargo.toml:

    ```
    [dependencies]
    rand = "0.8.3"
    ```

  - Ou utilizar o comando: $ cargo add rand@0.8.3
- Rust utiliza o modelo de versão semântica.
  - MAJOR.MINOR.PATCH

Geração do Cargo.lock

- Se usarmos o comando *$ cargo run* ou *$ cargo build*, poderemos perceber que ele pegou a versão 0.8.5 do rand

  Compiling libc v0.2.150
  Compiling cfg-if v1.0.0
  Compiling ppv-lite86 v0.2.17
  Compiling getrandom v0.2.11
  Compiling rand_core v0.6.4
  Compiling rand_chacha v0.3.1
  Compiling rand v0.8.5

- O cargo tentará utilizar a versão com o patch mais atualizado (Ex: 0.8.3 < 0.8.x > 0.9.0) para gerar o Cargo.lock

- O Cargo.lock contém as informações exatas das dependências. Não deve ser editado manualmente.

Atualizando uma dependência

- Caso o Cargo.lock esteja com uma dependência desatualizada, podemos utilizar o comando *$ cargo update* para atualizar os patch da dependência.

```
$ cargo update
Updating crates.io index
Updating rand v0.8.3 -> v0.8.5
```

Adição do rand ao código

- use rand::Rng; Definição do método do rand.
- let secret_number Criação da variável imutavel secret_number.
- rand::thread_rng() Chamada da função para gerar o número aleatório.
- .gen_range(1..=100) Método que possui o argumento (começo..=fim)
- Link do código

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

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

    let secret_number = rand::thread_rng()

        .gen_range(1..=100);

    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}");
}
```

Adição de comparação no código

- Esse código ainda não compila.
- use std::cmp::Ordering;
  Utilização da standard library
- match Realiza uma comparação entre guess e secret_number.

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

fn main() {
    // --snip--

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

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

Adição de comparação no código

- let mut guess = String::new()
  - guess é do tipo string
- let secret_number = rand::thread_rng().gen_range(1..=100);
  - por padrão secret_number é do tipo i32
- Será preciso transformar para o mesmo tipo

```
error[E0308]: mismatched types
  --> src/main.rs:22:21
   |
22 |     match guess.cmp(&secret_number) {
   |                 --- ^^^^^^^^^^^^^^^ expected struct `String`, found integer
   |                 |
   |                 arguments to this function are incorrect
   |
   = note: expected reference `&String`
              found reference `&{integer}`
note: associated function defined here
  --> /rustc/d5a82bbd26e1ad8b7401f6a718a9c57c96905483/library/core/src/cmp.rs:783:8

For more information about this error, try `rustc --explain E0308`.
error: could not compile `guessing_game` due to previous error
```

Modificando o tipo de guess

- Sombreamento de guess
  - let guess : u32 Cria uma nova variável com o nome guess to tipo u32, substituindo a antiga variável guess

- guess.trim().parse() Esse guess, faz referência ao antigo guess
  - trim() Elimina os espaços em branco no início e no final
  - parse() Converte a String em outro tipo, retorna Result.

- Antes secret_number era i32, mas o Rust irá fazer a comparação como se secret_number fosse u32

```
// --snip–
 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!"),
 }
```

Adição do loop

- loop cria um loop infinito
- break quebra o loop infinito

```
// --snip–

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

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

// --snip–

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

Lidando com Erro

- Lembrando que parse retornar Result, que pode ser Ok ou Err
  - Se o parser conseguir transformar guess em um número, ele retornará o número.
  - Se o parser não conseguir transformar guess em um número, ele inicia o nova iteração do loop.
- Link do código

```
// --snip–

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}");

// --snip--
```

Rodando o programa

```
$ cargo run
   Compiling guessing_game v0.1.0 (file:///projects/guessing_game)
       Finished dev [unoptimized + debuginfo] target(s) in 4.45s
       Running `target/debug/guessing_game`
Guess the number!
The secret number is: 61
Please input your guess.
10
You guessed: 10
Too small!
Please input your guess.
99
You guessed: 99
Too big!
Please input your guess.
foo
Please input your guess.
61
You guessed: 61
You win!
```

Código Final

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

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;
            }
        }}}
```

Referências

Rust Book - Capítulo 2:
https://doc.rust-lang.org/book/ch02-00-guessing-game-tutorial.html