



## Rusting with style - Curso básico de linguagem Rust

---



[Veja no GitHub](#)[Menu do curso](#)[VÍDEO DESTA AULA](#)

# Primeiro desafio

---

Vamos implementar um jogo de **black jack** (21) simplificado. As regras são essas:

1. O "baralho" tem 52 cartas, de Ás (vale 1) até Rei (figuras valem 10). As figuras são: Valete - 10, Dama - 10, Rei - 10.
2. O jogador recebe um valor inicial para apostar.
3. Cada rodada requer uma aposta mínima de 1.
4. Se o jogador ganhar da "mesa", ele ganha o equivalente ao que apostou.
5. Cada jogador pode comprar cartas até que esteja satisfeito ou que ultrapasse 21 pontos, nesse caso ele perdeu para a mesa.
6. A mesa pode comprar do mesmo jeito. Após a mesa comprar e parar, os jogos são comparados. Se a mesa tiver mais pontos (no máximo 21) ela ganha e o jogador perde o valor apostado.
7. O jogo termina quando o baralho acaba (são 4 baralhos) ou quando o jogador ficar sem valor para apostar, ou quando ele quiser.

## Algumas dicas para implementar esse jogo

### Números aleatórios

Você vai precisar "sortear" cartas, portanto, vai precisar de um gerador de números aleatórios. Em **Rust** utilizamos o `crate rand` para isso. Para começar, no `Cargo.toml` declare a dependência:

```
[package]
name = "blackjack1"
version = "0.1.0"
edition = "2021"

[dependencies]
rand = "0.8"
```

No seu código principal, declare que vai utilizar um elemento do `crate`:

```
use rand::thread_rng;
```

E no código, sempre que precisar "sortear" um número, use:

```
let mut rng = rand::thread_rng();
let idx = rng.gen_range(0..deck.len());
```

Aqui estamos criando um gerador de intervalos e sorteando um índice inteiro de carta entre zero e o tamanho do baralho (menos 1).

Lembre-se: Variáveis declaradas com `mut` são mutáveis, as declaradas sem `mut` são imutáveis.

## Usando vetor dinâmico

Como criamos um **baralho**? Com um vetor dinâmico de cartas! Adicionamos cartas antes de iniciar o jogo (dependendo da quantidade de baralhos que vamos jogar) e vamos retirando cartas conforme você e a mesa compram cartas.

Temos o `std::Vec` para isso:

```
// Criando um vetor dinâmico:
let mut deck = Vec::new();

// Adicionando um elemento ao vetor dinâmico:
deck.push(carta);

// Obtendo um elemento do vetor dinâmico (e removendo do vetor):
let carta = deck.remove(idx);
```

Você vai precisar de 3 vetores:

- Um para o deck (conjunto de baralhos)
- Um para a sua mão
- Um para a mão da mesa (do crupier)

## Decisão

É fácil... É como em **C** ou **Java**, só que não utilizamos parêntesis na condição, se não for necessário. Ah, e todo `if` tem que ter um bloco de comandos, mesmo que seja um só comando.

```
if aposta < 1 || aposta > dinheiro_jogador {
    println!("Aposta inválida. Tente novamente.");
    continue;
}
```

E temos a disjunção **or** (`||`) e a conjunção **e** (`&&`).

## Loops

Temos o `for` que navega entre intervalos:

```
for _ in 0..num_decks {
    for _ in 0..4 { // 4 naipes
```

```
        for carta in 1..=13 {  
            deck.push(carta);  
        }  
    }  
}
```

Acho que fica bem claro não? No primeiro **for**, a iteração começa em zero e vai até **num\_decks - 1**. Usei um **underscore** (**\_**) como variável de controle porque não estou interessado no índice. Se não fizer isso, vai dizer que a variável foi definida e não utilizada.

Temos também situações nas quais queremos fazer **loop eterno** controlando quando iremos sair dele. Para isso, usamos a instrução **loop**:

```
loop {  
    ...  
    break; // sai do loop  
    ...  
    continue; // pula os comandos seguintes e volta ao loop  
    ...  
}
```

**break** e **continue** também funcionam no **for**.

Quando você for iterar sobre uma "mão", use esse tipo de **for**:

```
for carta in mao {  
    total += valor_carta(carta);  
}
```

## Lendo valores da stdin

Você já sabe como mostrar mensagens na **stdout**, agora vai precisar ler valores da **stdin** e converter em números.

1. Mostrar algo na tela:

```
println!("Digite um número:");
```

2. Ler entrada do teclado:

```
let mut entrada = String::new();  
std::io::stdin().read_line(&mut entrada);
```

3. Converter para número (aqui usamos `unwrap()` para simplificar):

```
let numero: i32 = entrada.trim().parse().unwrap();
```

4. Mostrar o número lido:

```
println!("Você digitou: {}", numero);
```

### Exemplo completo:

```
use std::io;

fn main() {
    println!("Digite um número:");
    let mut entrada = String::new();
    io::stdin().read_line(&mut entrada);
    let numero: i32 = entrada.trim().parse().unwrap();
    println!("Você digitou: {}", numero);
}
```

## Faça o exercício

Com essas dicas, você consegue implementar o jogo de maneira bem simples mesmo. Na próxima aula, vamos conferir isso.