Enum & Pattern Matching Aufgabe

Entstehen soll eine Fruchtlotterie. Weil wir sowieso mit virtuellem Geld spielen wird solange gezockt, bis alles weg ist! Es wird ein Einsatz gesetzt und dann wird eine zufällige Frucht in einer zufälligen Farbe generiert dazu ein Multiplikator (der auch 0 sein kann). Mit diesem wird am Ende der Einsatz multipliziert und zurückgegeben. Gewonnen wird nur, wenn die Frucht in dieser Farbe auch wirklich existiert. Also ein blauer Apfel verliert.

Folgender code ist schon gegeben:

```
extern crate rand;
use rand::{Rng};
use std::io::{self, BufRead, Write};


fn main() {
    println!("Fruchtlotterie! Es wird gespielt bis
alles verzockt ist! Startgeld ist 100");
    let mut balance = 100.0;

    while balance > 0.0 {
        let einsatz = get_einsatz();
        balance -= einsatz;
        match roll(einsatz) {
            Some(value) => {balance += value; println!
("{} gewonnen :) Neuer Kontostand: {}", value,
balance)},
            None=> println!("Leider verloren :( Neuer
Kontostand: {}", balance),
        }
    }

    println!("Alles verzockt!");
}
```

```
//Fragt vom Benutzer einen Zahlenwert ab. Achtung
Falscheingaben sind hier nicht behandelt!
fn get_einsatz() -> f32 {
    print!("Einsatz: ");
    io::stdout().flush().unwrap();
    let stdin = io::stdin();
    let mut line = String::new();
    stdin.lock().read_line(&mut line).unwrap();
    line = line.replace("\n", "").replace("\r", "");
    line.parse::<f32>().unwrap()
}
```

//Überprüft, ob Frucht existiert oder Multiplikator 0. Bei verloren wird None geliefert, bei Gewinn wird Einsatz * Multiplikator geliefert

```
fn roll(einsatz: f32) -> Option<f32> {
    let fruit = Fruit::generate();
    //Zu implementieren
}
```

Tipp1:
Eine Zufallszahl in einer Range lässt sich einfach erzeugen mit:
//Erzeugt eine Zufallszahl zwischen 0 und bis einschließlich 99

```
let mut rng = rand::thread_rng();
let zahl = rng.gen_range(0,100);
```

Tipp2:
Wenn man enums comparen will muss man PartialEq implementieren. Das geht easy mit einem Macro:

```
#[derive(PartialEq)]
```

welches über das Enum gesetzt wird.

Tipp3: Ihr könnt selbst entscheiden bei welcher Wahrscheinlichkeit welcher Fall eintritt. Es macht Sinn hier ein paar Früchte und Farben zu wählen, die kompatibel und inkompatibel sind. z.B. ein Apfel kann Rot, Gelb und Grün sein. Eine Birne nur grün oder gelb. Eine Traube Rot oder Grün usw..

Tipp 3.5 So könnte eine Wahrscheinlichkeitsverteilung von Früchten aussehen.

```
match rng.gen_range(0,100) {
    0...30 => //Hier wird eine Birne erzeugt
    30...70 => //hier eine Traube
    _ => //Hier etwas weiteres
}
```

Tipp 4: Dass das rand crate läuft muss es in Cargo.toml eingebunden werden. Unter Dependencies:

```
rand = "0.6.5"
```

setzen.