Aufgabe 4: Würfelglück

Team-ID: ?????

Team-Name: 234bcd

Bearbeiter*innen dieser Aufgabe:
Michael Köhler

26. September 2021

Inhaltsverzeichnis

1 Lösungsidee

Da die Beispielaufgaben unterschiedliche Spieler anzahlen enthalten, muss die Aufgabenstellung in drei Teillaufgen unterteilt werden:

1. Bestimmung der Spieler mit verwendeten Würfeln aus der gegebenen .txt Datei. So wie das Erstellen von Paarung.
2. „Spielen"von Partien in Ausreichender Menge um den besseren Würfel in der Paarung zu bestimmen.
3. Vergleich von den Ergebnissen der einzelnen Partien um den besten Würfel von allen gegeben Würfeln zu Bestimmen.

Die Eigentliche Logik der Aufgabe befindet sich im zweiten Punkt. Hier müssen die Gegeben Regeln des Mensch ärgere dich nicht Spiels berücksichtigt werden. Die hierbei wichtigsten Regeln sind:

- Bei einer gewürfelten Sechs darf erneut gewürfelt werden.
- Bei einer Sechs wird falls möglich eine eigene Figur auf das Feld A der eigenen Farbe gestellt.
- Eine Gegnerische Figur die das Zielfeld besetzt wird beim betreten geschlagen (zurück auf eines der B Felder).
- Die vorderste Figur wird immer als erstes versucht zu bewegen (Abweichung von den Offiziellen Regel)
- Die Zielfelder [a, b, c, d] müssen genau erreicht werden. Es dürfen keine „Augenzahlen "verfallen
- Kann die vorderste Figur nicht bewegt werden, wird so lange die nächste weiteste Figur versucht zu bewegen.
- Kann keine der Figuren auf der „Laufbahn "bewegt werden, verfällt der Zug.
- Felder auf denen eine eigene Figur steht können nicht betreten werden. Auf den Zielfeldern dürfen eigene Figuren nicht übersprungen werden.
- Die Spieler führen ihre Züge immer im wechsel aus.

2 Umsetzung

Die Lösungsidee wird in Python implementiert. Die Python Imaging Library (PIL) stellt viele Funktionen zur Bildverar- beitung zur Verfügung. Damit funktionieren das Öffnen und Speichern des Bildes und der Zugriff auf die Pixeldaten sehr einfach. Wir importieren dazu das Modul Image der PIL. Mithilfe zweier ineinander geschachtelter For-Schleifen werden alle Pixel einzeln betrachtet. Immer wenn ein Pixel die gleiche Farbe hat wie eines seiner Nachbarpixel, färben wir beide Pixel weiß. Da bei dieser Aufgabe alle Pixel weiß gefärbt werden sollen, die zu einem Rhinozelfant gehören könnten, müssen wir aufpassen, dass wir nicht direkt ein Pixel im Bild weiß färben, wenn wir sehen, dass es einen gleichfarbigen Nachbarn gibt. Sonst kann es passieren, dass wir bei den anderen benachbarten Pixeln nicht mehr wissen, welche Farbe das aktuelle Pixel ursprünglich hatte. Dieses Problem wird gelöst, indem nicht die Pixel im Originalbild weiß gefärbt werden, sondern in einer Kopie des Bildes ('ausgabebild'). Dadurch können wir im Originalbild immer alle Pixel in ihrer ursprünglichen Farbe vergleichen. Zu guter Letzt wird das Ausgabebild wieder in eine Datei gespeichert.

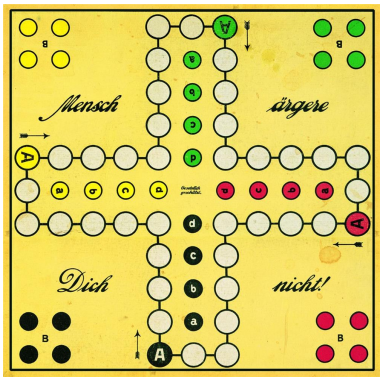

Abbildung 1: Spielbrett

3 Beispiele

Wir rufen das Programm für zwei der Beispieldateien auf und zeigen jeweils das resultierende Bild in verkleinerter Darstellung:

4 Quellcode

Beispiel-Code:

```
static void Main(string[] args)
        {
//test Comment!
            List<Player> players = SetPlayers(args);
            List<MatchUps> matchups = CreateMatchups(players);
            List<GameResults> gameresults = PlayGames(matchups);
            RankPlayers(gameresults);
    Console.WriteLine("Test string");

        }
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.IO;

namespace DiceCompare
{
    class Program
    {
        static void Main(string[] args)
        {
            var players = SetPlayers(args);
            var matchups = CreateMatchups(players);
            var gameresults = PlayGames(matchups);
            var rnd = new Random();
            RankPlayers(gameresults);

        }

        private static void RankPlayers(List<GameResult> gameresults)
        {
            throw new NotImplementedException();
        }

```

```
24          private static List<GameResult> PlayGames(List<MatchUp> matchups)
25          {
26              throw new NotImplementedException();
27          }
28
29          private static List<MatchUp> CreateMatchups(List<Player> players)
30          {
31              throw new NotImplementedException();
32          }
33
34          private static List<Player> SetPlayers(string[] args)
35          {
36              throw new NotImplementedException();
37          }
38      }
39  }
```