Soziale Medien & Kommunikationsinformatik

Programmieren II – Stefan Henkler

Das Brauseschaumspiel – Schere, Stein, Papier

Eine Dokumentation von Raffael Beyer, Kamil Betlejewski, Jonas Brinkötter, Laura Rempel & Eritrea Tekle

Gliederung

I Einführung

Die Aufgabe des semesterbegleitenden Projekts im zweiten Semester, war es ein Multiplayer-Spiel zu programmieren. Nach zahlreichen Vorschlägen, hat sich unsere Gruppe ‚Die Brausetabletten' für das Spiel ‚Schere, Stein, Papier' – oder auch ‚Schnick, Schnack, Schnuck' genannt, entschieden. ‚Schere, Stein, Papier' ist ein weltweit verbreitetes Spiel, das sowohl bei Kindern als auch bei Erwachsenen beliebt ist. Jeder der Kontrahenten entscheidet sich gleichzeitig für ein Symbol, welches normalerweise mit der Hand dargestellt wird. Jedes Symbol kann gegen eines der anderen gewinnen und gegen ein anderes verlieren. Der Spielausgang ist ungewiss, da keiner der beiden Spieler weiß, für welches Symbol sich der andere Spieler entscheiden wird. Die drei Hauptfiguren des Spiels sind *die Schere*, *der Stein* und *das Papier*. Die Schere schneidet das Papier und die Schere gewinnt. Das Papier wickelt den Stein ein – das Papier gewinnt. Und der Stein macht die Schere stumpf und dieser gewinnt. Entscheiden sich beide Spieler für dasselbe Symbol, wird das Spiel als Unentschieden gewertet und wiederholt.

Gleich zu Beginn des Projekts haben wir uns zunächst einen Github-Account erstellt und ein Gruppenrepository angelegt, in dem alle Fortschritte und die finale Version des Spiels dokumentiert werden.

Im nächsten Schritt haben wir uns mit einigen, für das Projekt relevanten Fragen wie zum Beispiel ‚Wie wird das Spielfeld aufgebaut? ', ‚Wie lauten die Siegbedingungen? ' und ‚Wie werden die Buttons angelegt? ' beschäftigt. Im weiteren Verlauf der Projektarbeit haben wir gemeinsam an dem Spiel und an Lösungen für entstandene Probleme gearbeitet.

II Spielfeld

Wie wird das Spielfeld aufgebaut?

Um ein ‚Schere, Stein, Papier' – Spiel mit grafischer Oberfläche programmieren zu können, muss man sich zu Beginn klar machen, welche Spielelemente benötigt werden. Danach ist es wichtig, sich zu überlegen wie diese Spielelemente grafisch umgesetzt werden können.

Im Falle dieser Projektarbeit besteht die Grundlage aus einem JFrame, dem JLabel (zur Ergebnisanzeige) und JButtons (zur Implementierung der Auswahlmöglichkeiten für Schere, Stein oder Papier) hinzugefügt werden. In diesem Kapitel werden die JButtons vorerst nicht behandelt.

Es werden zunächst folgende Klassen importiert:

1. java.awt.BorderLayout;
   → unterteilt einen Container in fünf Bereiche, in dem Grafikkomponenten verteilt werden können: NORTH, SOUTH, EAST, WEST, CENTER
2. java.awt.Color;
   → erlaubt das Färben von Komponenten
3. java.awt.Dimension;
   → gibt Höhe und Weite eine Komponente an (int)
4. java.awt.FlowLayout;
   → jedes Element wird damit rechts neben das zuletzt gesetzte in einer Zeile platziert
5. java.awt.event.ActionEvent;
   → erzeugt ein Event, wenn eine Aktion bestimmter GUI-Komponenten ausgeführt wird (button-click etc.)
6. java.awt.event.ActionListener;
   → interface zum Erfassen von action events
7. javax.swing.*;
   → stellt ‚leichte' Komponenten zur Verfügung, die auf allen Plattformen gleich arbeiten

Die Klasse ‚Spielfeld' muss von JFrame erben. Mit ‚setTitle' wird der Titel des Framen angegeben.

‚setDefaultCloseOperation(EXIT_ON_CLOSE);' sorgt dafür, dass das Programm nach Schließen des Fensters nicht weiterläuft. Danach werden die JLabels ‚l1', ‚gewinner' und ‚background' angelegt. JLabels können entweder einen string, ein image oder beides enthalten. ‚background' etwa, implementiert ‚Tisch.jpg' als Hintergrund für das Spielfeld.

```
JLabel background=new
JLabel(newImageIcon("F:\\workspace\\BrauseSchaumSpiel\\src\\SchaumSpiel\\Tisch.jpg"));;
```

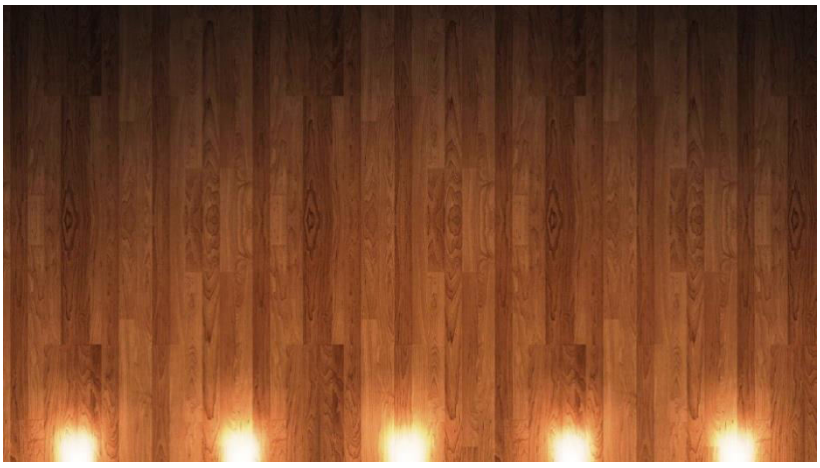

Abb.: 1

Die Auflösung wird auf FullHD gesetzt. Ein Nachteil ist, dass sie sich nicht dem Display anpasst, sollte es eine andere, native Auflösung haben.

setSize(1920, 1080);

Es wird ein absoluter Pfad verwendet um das, im Workspace liegende, Bild zu erreichen. Das Verwenden eines relativen Pfades wäre aber genauso möglich gewesen. Dem JLabel „l1" hingegen wird der String „made by Brausetabletten" übergeben

l1 = new JLabel ("made by Brausetabletten");

Die Schriftfarbe wird auf weiß geändert.

l1.setForeground(Color.white);

„gewinner" beinhaltet zunächst den String „Gewinner steht noch nicht fest" mit weißer Farbe, bevor dieser durch die gewinner-Methode verändert wird:

```
public void gewinner() {
    if (status == 1) {
        gewinner.setText("Gewonnen");
        gewinner.setForeground(Color.green);
        setVisible(true);
    }
    if (status == 2) {
        gewinner.setText("Verloren");
        gewinner.setForeground(Color.red);
        setVisible(true);
    }

    if (status == 3) {
        gewinner.setText("Unentschieden");
        gewinner.setForeground(Color.white);
        setVisible(true);
```

Abb.: 2

Falls der Spieler, der zuerst gedrückt hat gewinnt, wird die Farbe zu Grün. Bei einer Niederlage wird sie zu Rot. Bei einem Unentschieden bleibt der Text weiß. Label werden durch .add hinzugefügt.

Background.add(l1);

Abschließend wird dem Frame ein Icon gegeben, dass das Java-icon in der Fensterecke und in der Taskleiste ersetzt.

setIconImage(newImageIcon(getClass().getResource("/Schaumspiel/brause.png.")).getImage());

Abb.: 3

III JButtons

Um das Spiel „Schere, Stein, Papier" spielen zu können, benötigen wir ein Steuerelement, hierfür verwenden wir JButton. Der JButton ermöglicht es uns eine bestimmte Funktion auszuführen. Zuerst erzeugen wir die Buttons und anschließend platzieren wir sie.

```
JButton button0;
JButton button1;
JButton button2;
```

Abb.:4

Zu Beginn werden die integer sc, st und pa mit minus 1 initialisiert.

```
private boolean first
public int sc = -1;
public int st = -1;
public int pa = -1;
```

Abb.:5

Damit unser Button im Programm etwas ausführt benötigen wir einen Action Event/-Listener. Mit dem ActionListener wollen wir eine Methode in unserem Programm implementieren. Dazu benötigen wir die Methode ActionEvent, damit alle 3 Buttons ausgeführt werden. Wenn ein Button ausgeführt wird dann werden die integer je nach Button mit 0, 1 oder 2 überschrieben.

button0.addActionListener(new Action Listener() { public void actionPerformed(ActionEvent e) { sc = 0;

Als nächstes fügen wir die Button ein, denen wir einen Text zuordnen der auf den Buttons angezeigt wird, in unserem Fall Schere, Stein, Papier. Somit haben wir unsere 3 Buttons eingefügt und benannt.

```
// l2 = new JLabel("SchereTest")
// l3 = new JLabel("SteinTest");
// l4 = new JLabel("PapierTest")
button0 = new JButton("Schere");
button1 = new JButton("Stein");
button2 = new JButton("Papier");

// JLabel background = new JLabe
// ImageIcon("F:\\workspace\\Bra
```

Abb.:6

Um in unsere Buttons Grafiken anzuzeigen, benutzen wir „setIcon". Diese Methode dient dazu Icons auf Komponenten wie JButton, JLabel etc. abbilden zu lassen.

button0.setIcon(new ImageIcon("F:\\workspace\\BrauseSchaumSpiel\ \src\\SchaumSpiel\\Schereb.jpg"));

Mit „int m" geben wir die Maße für unsere Grafik ein.

```
// ImageIcon("F:\\workspace\\BrauseSchaumSpiel\\src\\Schau
button0.setIcon(new ImageIcon("F:\\workspace\\BrauseSchaum
button1.setIcon(new ImageIcon("F:\\workspace\\BrauseSchaum
button2.setIcon(new ImageIcon("F:\\workspace\\BrauseSchaum

int m = 2;
button0.setPreferredSize(new Dimension(200 * m, 300 * m));
button1.setPreferredSize(new Dimension(200 * m, 300 * m));
button2.setPreferredSize(new Dimension(200 * m, 300 * m));

button0.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        sc = 0;
        System.out.println(sc);
        // l2.setVisible(true);
```

Abb.:7

Wie man im oberen Abschnitt sehen kann (Button = new JButton) haben wir zwar die Buttons richtig angelegt, aber noch nicht in den JFrame eingefügt. Mit „background.add" verknüpfen wir die JButtons mit JFrame, nun wird der Frame ausgefüllt.

```
// 13.setVisible(false);
// 14.setVisible(false);
background.add(button0);
background.add(button1);
background.add(button2);
add(background);
```

Abb.:8

IV Spiel.java Teil I

In diesem Kapitel der Dokumentation wird erste Teil der Spiel.java erläutert.

```
package SchaumSpiel;

public class Spiel {


    static int pone;
    static int ptwo;

    public static void setzeS1(int wert) {
        pone = wert;
    }

    public static void setzeS2(int wert) {
        ptwo = wert;
    }
```

Abb.: 9

In diesem Abschnitt werden die beiden Spieler des Brauseschaumspiels festgelegt. Die erkennt man an den beiden „static ints pone und ptwo".

Anschließen bekommen beide Spieler einen Wert um später unterschieden zu werden.

```
public static int auswerten() {
    int status=0;
    String one = " ";
    String two = " ";
    String win = " ";
    // int sc = 0;
    // int st = 1;
    // int pa = 2;
```

Abb.:10

Im zweiten Abschnitt wird ein leerer String angelegt. Der Wert wird später überschrieben.

```
int pl = 1; // Int zur Unterscheidung

while (pl == 1) {
    switch (pone) {
    case 0:
        one = "Schere";
        pl++;
        break;
```

Abb.: 11

Im Abschnitt drei wird „int pl=1" angelegt, damit man pone und ptwo unterscheiden kann. Hier kann pone entscheiden welche Möglichkeiten also Schere, Stein oder Papier wählen möchte. Wenn er sich entschieden hat wird in der Konsole der Wert angezeigt, der für das jeweilige Symbol steht. Beispiel: Pone entscheidet sich für Schere wird in der Konsole 0 angezeigt. Am Endes des Zuges wird im Spiel angezeigt, das P1 wie im Beispiel Schere genommen hat und das wird wie folgt aussehen: „P1 hat Schere!".

„System.out.println(„P1 hat" + one + „!");

Die while-Schleife wurde für alle drei Cases und für beide Spieler angelegt.

```
    case 2:
        two = "Papier";
        p1++;
        break;
    }

}
System.out.println("P2 hat " + two + "!");
```

Abb.: 12

Der folgende Abschnitt unterscheidet sich fast nicht von dem dritten Abschnitt. Der Unterschied ist nur, dass hier ptwo am Zug ist und wie pone wählen kann welche Möglichkeit er nehmen möchte. Wenn dieser sich entschieden hat wird wie beim pone der Wert des gewählten Symbols angezeigt, sprich entscheidet sich der ptwo für Stein wird in der Konsole eine 1 stehen und am Ende des Zuges wird genauso wie beim pone im Spiel angezeigt was dieser gewählt hat und das wird wie folgt aussehen: „P2 hat Stein!".

V Spiel.java Teil II

Im zweiten Teil der Spiel.java erfolgt der Vergleich und anschliessend die Auswertung der jeweils von Spieler 1 und 2 gewählten Züge per Switch.

Abb.:13

```
case 1:

    if (pone == 1 && ptwo == 0 || pone == 1 && ptwo == 3) {
        win = "Du hast gewonnen!";
        status = 1;
    } else {
        if (pone == 1 && ptwo == 2 || pone == 1 && ptwo == 4) {
            win = "Du hast verloren!";
            status =2;
        } else {
            win = "Unentschieden!";
            status = 3;
        }
    }
    break;
```

Insgesamt gibt es 3 Cases, obig abgebildet ist Case 0.

Case 0 enthält die Auswertung des Siegers für den Fall das Spieler 1 Schere gewählt hat.

Haben Spieler 1 Schere und Spieler 2 Papier gewählt, so wird als Ergebnis dem String win der Inhalt "Du hast gewonnen!" übergeben und als Status wird 1 hinterlegt.

Haben Spieler 1 Schere uns Spieler 2 Stein gewählt, so wird in win der Inhalt "Du hast verloren!" und als Status 2 gespeichert.

Der letzmögliche Fall, wenn Spieler 1 und 2 Schere gewählt haben, ist das Unentschieden.

Ist ein Unentschieden der Fall wird als Status eine 3 hinterlegt und der String win erhält den Inhalt "Unentschieden!".

Der gleiche Ablauf findet in Case 1 und 2 statt, für die Fälle das Spieler 1 Stein beziehungsweise Papier gewählt hat.

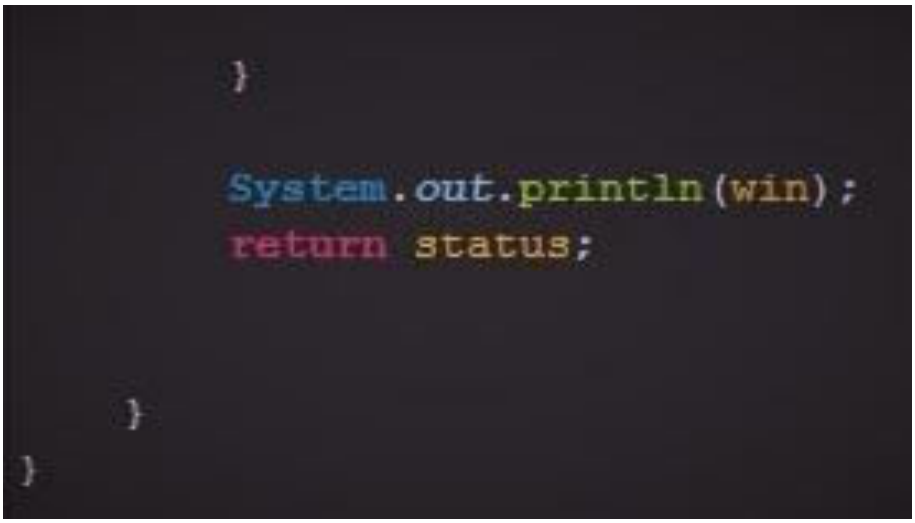

Abb.:14

Im abschließenden Teil des Codes werden die vorher im String win angehängten Inhalte ausgegeben.

VI RMI

Remote Method Invocation (RMI, deutsch etwa „Aufruf entfernter Methoden"), gelegentlich auch als Methodenfernaufruf bezeichnet, ist der Aufruf einer Methode eines entfernten Java-Objekts und realisiert die Java-eigene Art des Remote Procedure Call. „Entfernt" bedeutet dabei, dass sich das Objekt in einer anderen Java Virtual Machine befinden kann, die ihrerseits auf einem entfernten Rechner oder auf dem lokalen Rechner laufen kann. Dabei sieht der Aufruf für das aufrufende Objekt genauso aus wie ein lokaler Aufruf, es müssen jedoch besondere Ausnahmen (Exceptions) abgefangen werden, die zum Beispiel einen Verbindungsabbruch signalisieren können.

Auf der Client-Seite kümmert sich der sogenannte Stub um den Netzwerktransport. Der Stub muss entweder lokal oder über das Netz für den Client verfügbar sein. Das Erstellen des Stubs wird von der Java Virtual Machine übernommen.

Entfernte Objekte können zwar auch von einem bereits im Programm bekannten entfernten Objekt bereitgestellt werden, für die erste Verbindungsaufnahme werden aber die Adresse des Servers und ein Bezeichner benötigt. Für den Bezeichner liefert ein Namensdienst auf dem Server eine Referenz auf das entfernte Objekt zurück. Damit dies funktioniert, muss sich das entfernte Objekt im Server zuvor unter diesem Namen beim Namensdienst registriert haben. Der RMI-Namensdienst wird über statische Methoden der Klasse java.rmi.Naming angesprochen. Der Namensdienst ist als eigenständiges Programm implementiert und wird RMI Registry genannt.

Aus zeitlichen Gründen, hat unsere Gruppe es leider nicht geschafft eine funktionierende RMI zu programmieren.

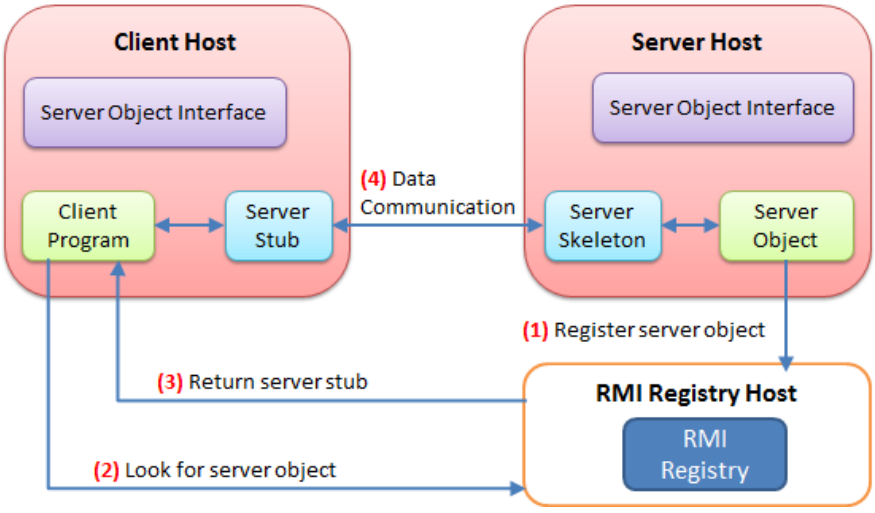

Abb.:15

VII Zusammenfassung

Während unserer Projektarbeit haben sich Herr Betlejewski und Herr Brinkötter um die Spiel.java-Datei gekümmert. Herr Beyer hat an den JLabels und an dem JFrame gearbeitet, während Frau Tekle die JButtons angelegt hat. Frau Rempel hat sich derzeit an der RMI versucht, wie bereits gesagt, konnten wir diese aus zeitlichen Gründen nicht fertigstellen. Folglich haben wir nur einige Ansätze der RMI, welche wir Ihnen bereits am 30. Juni in unserer Gruppenpräsentation vorgestellt haben.

Weiterhin lässt sich sagen, dass wir während des Projekts einige Komplikationen hatten, welche wir gemeinsam in der Gruppe gelöst haben. Eine Voraussetzung für ein solches Projekt sind selbstverständlich Programmier-Kenntnisse und das Verstehen und Lösen von Problem. Da dies nicht immer alleine funktioniert, ist es wichtig in einer Gruppe zu arbeiten, in der ein angenehmes Klima herrscht, und man gemeinsam an Lösungen arbeiten kann.

VIII Quellenverzeichnis

https://de.wikipedia.org/wiki/Remote_Method_Invocation

https://de.wikipedia.org/wiki/Schere,_Stein,_Papier

Bildverzeichnis

Abb. 1: http://www.wallpapermonkey.com/hd-wallpapers/wood-hd-wallpaper-hd/atta

Abb. 2: eigene Abbildung

Abb. 3: http://www.fotocommunity.de/photo/brausetablette-thorsten-thiel/127358

Abb. 4: eigene Abbildung

Abb. 5: eigene Abbildung

Abb. 6: eigene Abbildung

Abb. 7: eigene Abbildung

Abb. 8: eigene Abbildung

Abb. 9: eigene Abbildung

Abb. 10: eigene Abbildung

Abb. 11: eigene Abbildung

Abb. 12: eigene Abbildung

Abb. 13: eigene Abbildung

Abb. 14: eigene Abbildung

Abb. 15: https://github.com/harry1357931/Remote_Method_Invocation-RMI