Schiffe Versenken 2.0

Lara Sophie Haumersen
Laura Sophia Hämmerl
Annabell Borgelt
Alessia Caterisano

Agenda

1. Einleitung
2. Aufbauplan
3. Regelwerk
4. Main
5. Schiffe
6. Checkboxen
7. Spielfelder
8. Spielfeld
9. Actionlistener

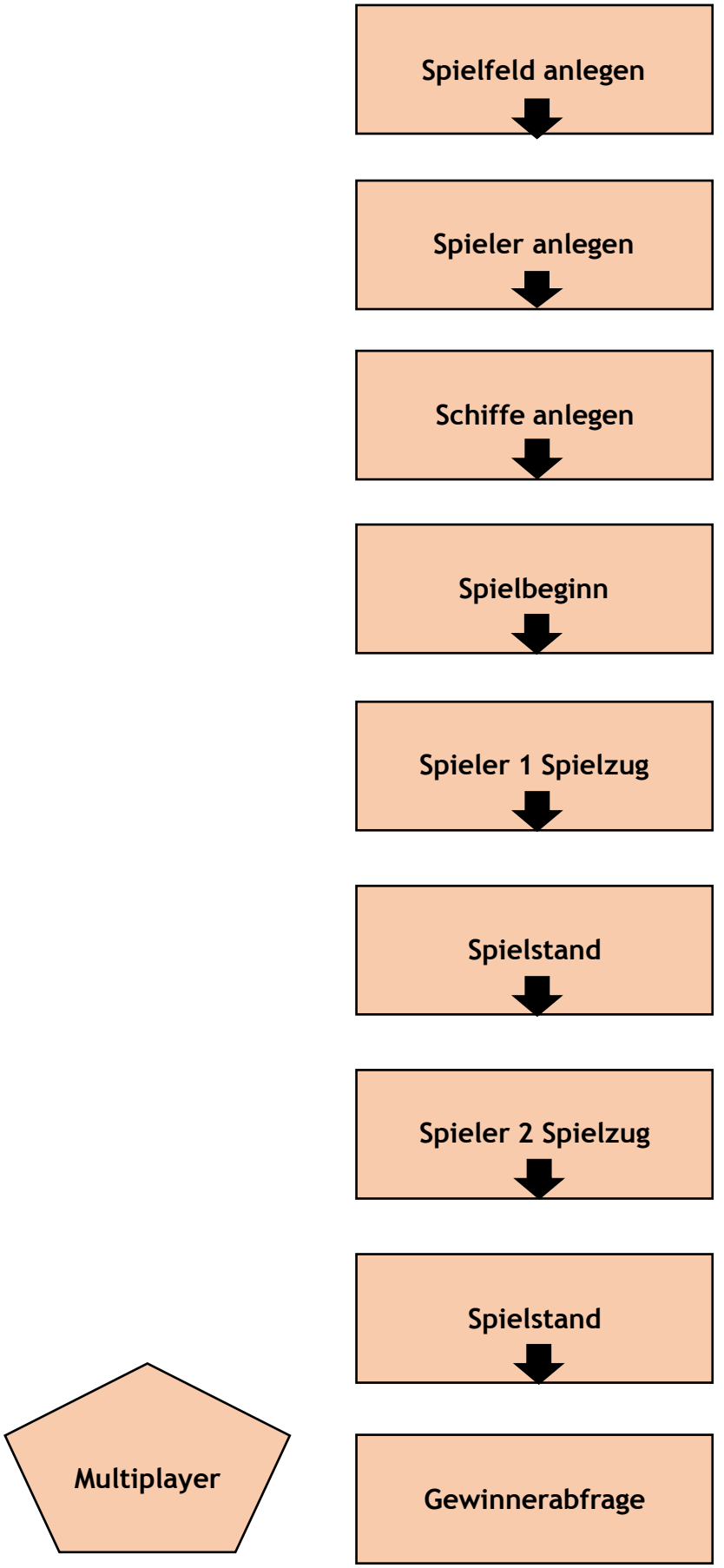
1. Einleitung

Ziel des Semesterprojektes des Praktikums Programmieren II ist es ein Mehrpersonenspiel mit grafischer Oberfläche zu entwickeln. Das Spiel soll an unterschiedlichen Rechnern gegeneinander gespielt werden können. Der Code soll in Java geschrieben und ausgeführt werden.

Das programmierte Spiel richtet sich nach dem bekannten Zweispielerspiel „Schiffe versenken“. Da es keine exakte Kopie des Spiels ist, sondern sich durch einige Regeln von dem Originalspiel unterscheidet, wurde ein eigener Name (Schiffe versenken 2.0) gewählt und ein eigenes Regelwerk erstellt. Nachdem die Grundidee feststand, haben die Projektpartner gemeinsam einen Spielablaufplan entwickelt. Anhand dieses Ablaufplans orientierte sich das gesamte Programm, beginnend mit der Initialisierung des Spielfeldes, über die Erstellung der Schiffe und der 49 Spielfeld-Boxen (Checkboxen), hin zu der grafischen Gestaltung des Spielfeldes. Auf besagte Punkte wird im Laufe dieser Verschriftlichung ausführlich eingegangen.

Ob gleich das Programm bezüglich der Präsentation unter den Projektpartnern aufgeteilt wurde, wurde das gesamte Programm gemeinsam ausgearbeitet und erstellt.

2.Ablaufplan



3. Regelwerk

• 15 Schiffe sind zu versenken

Ziel des Spiels ist es 15 zufallsgenerierte Schiffe unter den 49 möglichen Feldern zu versenken.

• Maximal 49 Spielzüge pro Spieler möglich

Während des Spiels ist es möglich pro Spieler höchstens 49 Felder auszuwählen. Das heißt jeder Spieler hat höchstens 49 Züge frei.

• Spieler mit den wenigsten Spielzügen gewinnt

Da jeder Spieler genauso viele Spielzüge hat, wie es Spielfelder gibt, gewinnt der Spieler, der die wenigsten Spielzüge braucht, um alle Schiffe zu versenken.

• Ein Spielwechsel pro Spieler

Eine komplette Spielrunde besteht aus jeweils einem Spielerwechsel. Das bedeutet, dass Spieler 1 beginnt und mit maximal 49 Spielzügen die 15 Schiffe versenkt. Dann erfolgt der Spielerwechsel, nach dem Spieler 2 ebenfalls versucht alle 15 Schiffe in weniger Zügen zu versenken, als Spieler 1.

4. Main

Zu Beginn des Mainblocks wird das Package main aufgerufen. Darauf folgend wird RemoteException und die grafische Oberfläche des Spielfeldes importiert.

```
package main;
import java.rmi.RemoteException;

import gui.Spielfeld;

public class Main {
	public static void main(String[] args) throws
RemoteException {
		Spielfeld spielfeld = new Spielfeld();
		spielfeld.setSize(800, 600);
		spielfeld.setVisible(true);
	}
}
```

Abb.1

Nachdem Öffnen der Main Klasse, wird auch die Main Methode geöffnet. Throws RemoteException ist im Exception-Teil des Methodenkopfes enthalten, um die Robustheit gegenüber jeglichen Kommunikationsproblemen sicherzustellen. Nachfolgend wird das Spielfeld initialisiert. Auch wurde festgelegt welche Größe (8000x6000 Pixel) das Spielfeld haben sollte. Zuletzt wurde der Ausdruck .setVisible mit dem Boolean true gesetzt, um das Spielfeld beim Ausführen des Programms auch tatsächlich anzeigen zu lassen. (Abb1)

5. Schiffe

In der Klasse „Schiffe“ befindet sich eine ArrayList, in der Integer (Zahlen) gespeichert werden. Unser Spiel hat 15 verschiedene Schiffe, die durch 15 Zufallszahlen bestimmt werden. Die Zufallszahlen werden von 0 – 48 zufällig ausgewählt, da es 49 mögliche Felder gibt.

Int zahl = (int) Math.round(Math.random() * 48);

Die Schiffe können nicht doppelt gesetzt werden. Falls die Zufallszahl beispielsweise doppelt 5 ist, wird k um eins subtrahiert. Somit wäre die Zufallszahl dann die 4. (Abb. 2)

```
package main;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;

public class Schiffe extends UnicastRemoteObject implements SchiffeInterface{
	ArrayList<Integer> schiffe = new ArrayList<Integer>();

	public ArrayList<Integer> getSchiffe() {
		return schiffe;
	}

	public void setSchiffe(ArrayList<Integer> schiffe) {
		this.schiffe = schiffe;
	}

	public Schiffe() throws RemoteException {
		for (int k = 0; k <= 14; k++) {
			int zahl = (int) Math.round(Math.random() * 48);
			if (schiffe.contains(zahl)) {
				k--;
			} else {
				schiffe.add(zahl);
			}
		}
	}

	@Override
	public boolean contains(int i) throws RemoteException {
		// TODO Auto-generated method stub
		return this.getSchiffe().contains(i);
	}
```

Abb.2

6. Checkboxen

Die Checkboxen bilden das Spielfeld. Sie sind dafür da, um die Felder anzuklicken und auszuwählen. Es gibt 49 Checkboxen, die von dem Spieler angeklickt werden können. Auf Grund der 49 Checkboxen, ist pro Spieler 49 Züge möglich.

public CheckBoxen () { for(inti= 0;i<49,i++){

Die Checkboxen wurden in einer ArrayList angelegt.

Public class CheckBoxen {
ArrayList <JCheckBox> boxen = newArrayList<JCheckBox> ();

7. Spielfelder

Es befindet sich ein JPanel im GridLayout. Der Panel wird dem Layout zugewiesen. Das GridLayout hat die Größe 49

GridLayout layout = new GridLayout (7,7); layout.setVgap (1);
layout.setHgap (2);

Mit dem setVgap wird die Vertikale angegeben und mit dem setHgap die Horizontale. Somit haben wir ein GridLayout mit 7 x 7.

8.Spielfeld

Im JFrame, welches das Standart-Fenster für grafische Oberflächenprogrammierung ist und alle Container und Bedienelemente aufnimmt und zu einer Oberfläche vereint, wird das Spielfeld dargestellt. Durch das Erzeugen des JPanel mit BorderLayout wird die Anordnung und teilweise auch das Verhalten der Checkboxen bestimmt. Der JPanel bestimmt in welchem Bereich des JFrames das Objekt abgelegt wird. Die Möglichkeiten bestehen es im, South,

Abb.3

```
(...)

public class Spielfeld extends JFrame {
private static final long serialVersionUID =
3571603337396435035L;

SchiffeInterface schiffel;

private CheckBoxen checkBoxen = new
CheckBoxen(); //private Schiffe schiffe;// = new
Schiffe();
JTextField zug = new JTextField();
JTextField stand = new JTextField();

Integer[] spielstand = new Integer[2]; int spieler = 0;
```

North, East and West Bereich zu platzieren. Gibt man keine genaue Platzierung an wird es im Center (mittig) Bereich dargestellt. Die 49 CheckBoxen der Spielfläche werden durch den HauptPanel im Center des JFrame platziert. Der Abstand vom Fensterinhalt zum Rand (10, 10, 10, 10) ist durch , BorderFactory.createEmptyBorder, definiert worden. Das Programm wird über einen Start Button/Schaltfläche, am rechten Rand (East) gesteuert. Zwei Textfelder, JTextField zug und JTextField stand, welche den Stand des Spieles bzw den Zwischenstand und die Anzahl der Spielzüge ausgibt befindet sich ebenfalls am rechten Rand. Es besteht nicht die Möglichkeit das die Spieler selber in dem Textfeld schreiben können (false gesetzt). Dem HauptPanel wird ein weiterer Panel zugeordnet der den Start Button und das Textfeld im East Part platziert. (Abb. 3 und 4)

```
public Spielfeld(SchiffeInterface si) throws RemoteException { super("Schiffe
versenken");

this.schiffel = si;
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel hauptPanel = new JPanel(new BorderLayout());
hauptPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

Box buttonBox = new Box(BoxLayout.Y_AXIS);

JButton start = new JButton("Starten"); start.addActionListener(new Mein-
StartListener()); buttonBox.add(start);
//schiffe = new Schiffe();

zug.setText("Spieler 1 ist dran"); zug.setEditable(false); buttonBox.add(zug);
stand.setEditable(false); buttonBox.add(stand);

hauptPanel.add(BorderLayout.EAST, buttonBox); hauptPanel.add(new Spielfel-
der(checkBoxen)); getContentPane().add(hauptPanel);

}

public void setSchiffeInterface
```

Abb.4

9. Actionlistener

Im Actionlistener werden die Anzahl der Schüsse und Treffer angegeben. Welche zu Anfang 0 ergeben, wenn keine Aktion durch das „Start“ gemacht wurden. Würde beispielsweise in 5 Zügen 2 Treffer gemacht werden, so wird diese dadurch ausgegeben, durch „anzahlSchuesse“ und „anzahlTreffer“. Durch die for each Schleife wird überprüft, dass wenn eine Checkbox ausgewählt wird die Checkbox überprüft wird ob diese belegt ist. Wenn die Checkbox belegt ist, wird die Anzahl der Schüsse erhöhen. (Abb. 5)

```
int anzahlSchuesse = 0;
int anzahlTreffer = 0;
int i = 0;
for (JCheckBox checkBox : checkBoxen.getBoxen()) { if
(checkBox.isSelected()) {
                        anzahlSchuesse++;
//if (schiffe.getSchiffe().contains(i)) { // anzahlTreffer++;
//}
try {
if (schiffe1.contains(i)) {
                                anzahlTreffer++;
                        }
} catch (RemoteException e) {
// TODO Auto-generated catch block
                                e.printStackTrace();
                        }
}
i++;}
                stand.setText("Schüsse: " + anzahlSchuesse + " Treffer: "
                        + anzahlTreffer);
```

Abb. 5

```
if (anzahlTreffer == 15) {
 spielstand[spieler] = anzahlSchuesse;
 for (JCheckBox checkBox : checkBoxen.getBoxen()) {
 checkBox.setSelected(false);
 }
 if (spieler == 0) {
 zug.setText("Spieler 2 ist dran");
 spieler = 1;
} else {
 String message = "Das Spiel ist vorbei.\n";
 if (spielstand[0] < spielstand[1]) {
 message += "Gewonnen hat Spieler 1. Mit "
 + spielstand[0] + "Sch„ssen.";
 } else if (spielstand[0] > spielstand[1]) {
 message += "Gewonnen hat Spieler 2. Mit "
 + spielstand[1] + "Sch„ssen.";
 } else {
 message += "Das Spiel ist unentschieden";
 }
JOptionPane.showMessageDialog(null, message);
```

Abb. 6

Außerdem hat jede Checkbox einen Zustand, sei es true oder false.
Durch die zweite for each Schleife wird angegeben, dass wenn alle 15 Treffer gemacht worden sind alle Checkboxen auf „selected“ gestellt werden und somit Spieler 2 mit dem Spielzug dran ist. Dabei wird auch der Schriftzug „Spieler 1 ist dran“ in „Spieler 2 ist dran“ geändert. Hierbei wird durch die if else Bedingung nach den 15 Treffern gewechselt. Für den Spielstand wird geschaut, dass wenn der Spielstand eines Spielers höher ist als der des anderen ist, wird ein Sieger ausgegeben. Hierbei ist auch ein Unentschieden möglich. Durch „Message“ wird eine Nachricht ausgegeben, dass das Spiel vorbei ist und durch das JOptionPanel wird ein kleines Fenster ausgegeben, welches eine Nachricht beinhaltet, dass das Spiel vorbei ist. (Abb. 6)