

System-Spezifikation

Hochschule Luzern Technik & Architektur

Programmieren 2 - FS12

Gruppe 10 Redzepi Iljasa | Reichmuth Marco | Rey Philipp | Rohrer Felix

Eine interdisziplinäre Projektarbeit der Studiengänge Elektrotechnik und Informatik.

Horw, 31.05.2012

Autoren

Redzepi Iljasa		Reichmuth Marco	
Studiengang	Informatiker (Berufsbegleitend)	Studiengang	Elektroniker (Berufsbegleitend)
Adresse		Adresse	
Telefon		Telefon	
E-Mail	iljasa.redzepi@stud.hslu.ch	E-Mail	marco.reichmuth@stud.hslu.ch
Rey Philipp		Rohrer Felix	
Studiengang	Elektroniker (Vollzeit)	Studiengang	Informatiker (Berufsbegleitend)
Adresse		Adresse	
Telefon		Telefon	
E-Mail	philipp.rey@stud.hslu.ch	E-Mail	felix.rohrer@stud.hslu.ch

Änderungskontrolle

Version	Datum	Autor	Beschreibung
1.0	22.04.2012	Felix Rohrer	Vorlage erstellen, erste Texte schreiben
1.1	10.05.2012	Iljasa Redzepi	Kapitel Architektur beschrieben
1.2	10.05.2012	Felix Rohrer	Klassendiagramm, Design-Entscheide
1.3	10.05.2012	Iljasa Redzepi	Sequenzdiagramm erstellt
1.4	10.05.2012	Rey Philipp	Softwareanforderungen und Design-Entscheide
1.5	11.05.2012	Felix Rohrer	CLI Client dokumentiert
-	11.05.2012	Felix Rohrer	Version 1.5 freigeben
1.6	30.05.2012	Felix Rohrer	Klassendiagramm auf die neuste Version angepasst
			GUI – Client dokumentieren
-	31.05.2012	Felix Rohrer	Version 1.6 freigeben

Inhalt

1 E	ıführung1		
2 A	rchitektur-Beschreibung	L	
2.1 2.2 2.3	Funktionale Sicht Entwicklungssicht Verteilungs- und Betriebssicht	25	
3 S	oftwareanforderungen und Design-Entscheide	5	
3.1 3.2	Softwareanforderungen6 Design-Entscheide	57	
4 E	nvironment-Anforderungen	7	
4.1 4.2 4.3	Hardware Software Java Virtual Machine	7 7 7	
5 C	CLI – Client	3	
5.1 5.2	Unterstützte CLI Parameter	3 Э	
6 G	GUI – Client	L	
6.1 6.2 6.3 6.4 6.5 6.6	Spielen im GUI12Settings12Cheat / Hilfe12Solver13Design14Help / About15	L 2 3 1 5	
Abbilo	dungsverzeichnis16	5	

1 Einführung

Informationen über das System "Mastermind" werden im Dokument "Kundenanforderungen" spezifiziert.

2 Architektur-Beschreibung

2.1 Funktionale Sicht



Abb. 1 Mastermind: Funktionale Sicht (Sequenzdiagramm)

2.2 Entwicklungssicht

2.2.1 Packages Übersicht



Abb. 2 Mastermind: Package Übersicht

2.2.2 Mastermind Klassenübersicht



Abb. 3 Mastermind: Klassen Übersicht

2.2.3 Package Model

Settings
Attributes
public int ANZ_COLOR_MIN = 4
public int ANZ_COLOR_MAX = 8
public int ANZ_PEG_MIN = 4
public int ANZ_PEG_MAX = 10
public int ANZ_GUESS_MIN = 6
public int ANZ_GUESS_MAX = 12
public int ANZ_DESIGN_MAX = 6
private int numberOfColors
private int numberOfPegs
private boolean allowDuplicateColor
private int numberOfRows
private int idDesignUsed
Operations
public Settings()
public void setSettings(int numberOfColors, boolean allowDuplicateColor, int numberOfPegs, int numberOfRows)
public int getNumberOfColors()
public void setNumberOfColors(int numberOfColors)
public int getNumberOfPegs()
public void setNumberOfPegs(int numberOfPegs)
public boolean getAllowDuplicateColor()
public void setAllowDuplicateColor(boolean allowDuplicateColor)
public int_getNumberOfRows(_)
public void setNumberOfRows(int numberOfRows)
public int getDesign()
public void setDesign(int idDesign)

📃 Guess
Attributes
private int pegs[0*]
private int score
Operations
public Guess()
public Guess(int pegs[0*])
public Guess(int pegs[0*], int score)
public void setPegs(int pegs[0*])
public int[0*] getPegs()
public void setPeg(int pos, int color)
public int getPeg(int pos)
public int getScore()
public String toString()
public boolean equals(Object otherGuess)
public Result compareGuess(Guess otherGuess)
public int hashCode()

📃 Result	
Attributes	
private int anzPegColorPosRight = 0	
private int anzPegColorRight = 0	
Operations	
public Result()	
public Result(int anzPegColorPosRight, int anzPegColorRight)
public int_getAnzPegColorPosRight()	
public void setAnzPegColorPosRight(int anzPegColorPosRight	t)
public int getAnzPegColorRight()	
public void setAnzPegColorRight(int anzPegColorRight)	
public void incAnzPegColorPosRight()	
public void incAnzPegColorRight()	
public int[0*] getResult()	
public String getResultAsString()	
public String toString()	
public boolean equals(Object otherResult)	
public int hashCode()	

Abb. 4 Package Model: Klassenübersicht

2.2.4 Package Logic



GuessHandler
Attributes
Operations
public GuessHandler()
public boolean validateGuess(int pegs[0*])
<u>public boolean_validateGuess(Guess guess)</u>

	📃 Serializer
	Attributes
priva	ate ObjectOutput output
priva	ate FileOutputStream fileOuputStream
priva	ate FileInputStream fileInputStream
	Operations
<u>pub</u>	lic void Serialize(File file, Object obj.)
pub	lic Object DeSerialize(File file)

📃 RandomGuess
Attributes
Operations
public Guess_genRandomGuess(_)

ArgParser
Attributes
<pre>private String params[0*] = new Vector<string>()</string></pre>
private int paramIndex = 0
private int numberOfParams = 0
Operations
public ArgParser(String args[0*])
public boolean hasOption(String opt)
public String getOption(String opt)
public String nextParam()
public int_numberOfParams(_)

Abb. 5 Package Logic: Klassenübersicht

2.2.5 Package Client



Abb. 6 Package Client: Klassenübersicht

2.2.6 Package Solver



Abb. 7 Package Solver: Klassenübersicht

2.2.7 Package Engine



Abb. 8 Package Engine: Klassenübersicht

2.3 Verteilungs- und Betriebssicht

In unserem Projekt wird aktuell die Anwendung nicht auf verschiedene Systeme verteilt. In einem späteren Zeitpunkt, wenn die Option Netzwerk implementiert wird, wird die Anwendung auf verschieden System verteilt.

3 Softwareanforderungen und Design-Entscheide

3.1 Softwareanforderungen

3.1.1 Nichtfunktionale Anforderungen

Anforderung	Beschreibung
Zuverlässigkeit	Es sollen keine Abstürze des Spiels vorkommen
Benutzerfreundlichkeit	Intuitive und verständliche Oberfläche (GUI)
Effizienz	Systemressourcen werden nicht unnötig ausgelastet
Performance	Anweisungen des Benutzers sollen ohne Verzögerung ausgeführt werden
Wartbarkeit	Nachträgliche und Änderungen sind ohne grosse Umstände möglich
Erweiterbarkeit	Nachträgliche Ergänzungen sind ohne grosse Umstände möglich
Korrektheit	Auf die Ergebnisse/Ausgaben des Spiels ist Verlass

3.1.2 Funktionale Anforderungen

Anforderung	Beschreibung
Spiel Speichern	Aktuellen Spielstand abspeichern
Spiel laden	Gespeicherten Spielstand wieder abrufen
Hilfe	Hilfestellung anzeigen
About-Dialog	Informationen über Spiel anzeigen
Optionen	Rolle und Peg-Anzahl wählen

3.2 Design-Entscheide

Nach Möglichkeit wird das Model-View-Controller (MVC) Architekturmuster angewendet.

Das Mastermind wird so aufgebaut, dass der "Client" unabhängig von der Engine ist. Die definierten Schnittstellen ermöglichen eine möglichst kleine Abhängigkeit zwischen diesen zwei Logischen Einheiten. Dadurch ist es in einem späteren Entwicklungsschritt kein Problem einen weiteren Client zu implementieren.

3.2.1 Singleton Settings

Für die Einstellungen (Settings) wird das Singleton Entwurfsmuster verwendet. Zusammen mit dem SettingsHandler ist sichergestellt das es jeweils nur eine Instanz von diesem Objekt gibt.

Jeder Softwareteil ist selber bemüht eine entsprechende Referenz anzufordern und die entsprechenden Settings zu verwenden.

3.2.2 Interface zwischen Engine und Client

Das Interface zwischen Mastermind Engine und Client ist so implementiert, dass entsprechende Objekte übergeben werden.

Der Client übergibt der Engine ein "Guess" Objekt, darin enthalten ist eine Zeile, resp. Rateversuch.

Als Rückgabewert liefert die Engine ein "Result" Objekt, darin ist die Anzahl Schwarzer- resp. Weisser-Stifte (richtige Farbe am Richten Ort, resp. richtige Farbe) enthalten.

Durch weitere Methoden, welche von der Engine zur Verfügung gestellt werden, kann z.B. direkt Abgefragt werden ob der Spieler gewonnen hat oder nicht.

4 Environment-Anforderungen

4.1 Hardware

Das Mastermind-Spiel läuft auf fast jeder Hardware. Grundsätzlich muss Java lauffähig sein, an die Ressourcen werden vom Spiel keine speziellen Anforderungen gestellt.

Für den Solver, auf einem 8 Farben mit 8 Position Spielfeld, ist es von Vorteil wenn man ein Multi-Core System mit ausreichend Ram besitzt.

4.2 Software

Das Mastermind-Spiel läuft auf allen Betriebssystemen für welche es Java gibt.

4.3 Java Virtual Machine

Damit das Mastermind-Spiel läuft, muss die JVM installiert sein.

5 CLI – Client



Abb. 9: CLI – Hilfe / About

5.1 Unterstützte CLI Parameter

5.1.1 -? | Hilfe

Hilfe / Usage anzeigen

5.1.2 –c | Anzahl Farben

Anzahl Farben definieren, z.B. für 4 Farben: -c=4

5.1.3 -d | Doppelte Farben (Ja/Nein)

Doppelte Farben erlaubt, z.B. für "Ja": -d=j

5.1.4 –p | Anzahl Steckplätze

Anzahl Steckplätze definieren, z.B. für 6 Positionen: -p=6

5.1.5 –a | Anzahl Rateversuche

Anzahl Versuche, z.B. für 10 Versuche: -a=10

5.1.6 –s | Secret Code setzen

Das zu erratende Muster definieren, z.B. für 4132: -s=4132

5.1.7 -n | Nochmals spielen (Ja/Nein)

Abfrage ob nochmals spielen, z.B. für "Nein": -n

5.2 CLI-Client Bedienung

5.2.1 Optionen definieren

Falls die Optionen nicht via Parameter definiert wurden, werden diese Abgefragt. In den Eckigen-Klammern ist der Default-Wert vorgegeben, welcher mittels Return übernommen werden kann.

Sobald alle Optionen definiert sind, wird der "Secret Code" erzeugt.

```
Anzahl Farben (4-8) [4]:
Doppelte Farben (j/n) [n]:
Anzahl Steckplaetze (4-6) [4]:
Anzahl Versuche (1-30) [6]:
Secret Code wird erzeugt...
```

Abb. 10: CLI – Optionen definieren

Wird eine ungültige Option eingegeben, wird dies abgefangen und solange wiederholt bist es eine gültige Eingabe war.

	-
Anzahl Farben (4-8) 4 :	2
	— · · · · · · · · · · · · · · · · · · ·
> Bitte queitige Anzani	Farben eingeben!
An = ah = an han (1 0) [1]	F
Anzani Farben (4-6) [4]:	- 5
> Bitto qualtiga Anzahl	Earbon oingohonl
> bille guerlige Anzann	raiben eingeben:
Anzahl Earbon (1-8) [1]	2
	a
Nitte queltige Anzahl	Farhen eingehenl
> Diete guereige Anzani	rurben enigeben.
Δ nzahl Earben (4-8) [4].	6
	U

Abb. 11: CLI – Ungültige Optionen abfangen

Die Optionen müssen "spielbar" sein, z.B. ist die Kombination mit 4 Farben, ohne Doppelte Farben und 6 Steckplätze nicht möglich. Dies wird solange wiederholt bis eine spielbare Konfiguration eingegeben wurde.

```
Anzahl Farben (4-8) [4]: 4
Doppelte Farben (j/n) [n]: n
Anzahl Steckplaetze (4-6) [4]: 6
Anzahl Versuche (1-30) [6]: 8
```

Diese Optionen sind nicht gueltig! Bitte neue eingeben...

Abb. 12: CLI – Ungültige Kombination abfangen

5.2.2 Spielen

Die Eingabe der Kombination erfolgt als Zahlen, pro Position ist es eine Ziffer.

Für die Kombination 1-2-3-4 wird "1234" eingegeben. Nach der Eingabe erfolgt das Resultat in Anzahl Schwarzer, resp. Anzahl Weisser Stiften.

```
Versuch 1/6: 1234
> Schwarz: 1, Weiss: 3
Versuch 2/6: 2134
> Schwarz: 2, Weiss: 2
Versuch 3/6: 2314
> Schwarz: 1, Weiss: 3
Versuch 4/6: 2143
> Schwarz: 1, Weiss: 3
Versuch 5/6: 1324
> Schwarz: 2, Weiss: 2
Versuch 6/6: 3214
Du hast leider nicht gewonnen. Die gesuchte Kombination war: [3, 1, 2, 4]
```

Abb. 13: CLI – Rateversuche

Falsche und ungültige Eingaben werden hier ebenfalls abgefangen.

```
Versuch 1/6: 123456
> Bitte eine gueltige Kombination eingeben!
Versuch 1/6: asdf
> Bitte eine gueltige Kombination eingeben!
Versuch 1/6: a s d f
> Bitte eine gueltige Kombination eingeben!
Versuch 1/6: 1 2 3 4
> Bitte eine gueltige Kombination eingeben!
```

Abb. 14: CLI – Ungültige Rateversuche abfangen

6 GUI – Client

Das GUI wird primär mit der Maus bedient. In der obersten Zeile ist das zu erratende Muster. In den darunter liegenden Zeilen können die Rateversuche eingegeben werden. Rechts davon ist der Knopf für die Überprüfung sowie das Ergebnis (Schwarze / Weisse Pins).

Um eine Farbe zu setzen kann die Maustaste oder aber das Mausrad verwendet werden.

Mastermind	
Game Cheat Help	
0000	
0000	0
0000	•••
0000	••
0000	••
0000	••
0000	•••
Tipp: Use Mousewheel to c	hange Peg

Abb. 15: GUI – Startscreen

6.1 Spielen im GUI

Eine Überprüfung des Rateversuches ist erst möglich nachdem eine gültige Kombination eingegeben wurde. Zuunterst wird jeweils angezeigt ob man Gewonnen oder Verloren hat.



Abb. 16: GUI – Ungültige Eingabe



Abb. 17: GUI – Gewonnen



Abb. 18: GUI – Verloren

6.2 Settings

In den Settings können die verschiedenen Optionen definiert werden:

- Anzahl Farben
- Doppelte Farben
- Anzahl Positionen
- Anzahl Rateversuche
- Design

Es können nur "gültige" (spielbare) Optionen gespeichert werden.



Abb. 19: GUI – Settings Dialog

Settings		X
Color Settings Number of Colors Allow duplicate Colors Number of Pegs Number of Attempts	4 ▼ No ▼ 8 ▼ 6 ▼	Layout Icon Smiley Please select playable options Cancel Save

Abb. 20: GUI – Settings Dialog: Ungültige Eingabe

6.3 Cheat / Hilfe

Unter dem Menü punkt "Cheat" kann Wahlweise ausgewählt werden ob nur eine einzelne Position oder die gesamte Lösung angezeigt werden soll.

Mastermind	
Game Cheat Help	
$\bigcirc \textcircled{0} \textcircled{0} \textcircled{0} \textcircled{0} \textcircled{0} \textcircled{0} \textcircled{0} \textcircled{0}$	
0000	0
0000	•••
0000	•••
0000	•••
0000	•••
0000	•••
Tipp: Use Mousewheel to c	hange Peg

Abb. 21: GUI – Cheat: One Peg

Mastermind	X
Game Cheat Help	
0000	(
0000	••
0000	•••
0000	•••
0000	••
0000	•••
Tipp: Use Mousewheel to c	change Peg

Abb. 22: GUI – Cheat: Ganze Lösung anzeigen

6.4 Solver

Für den Solver muss zuerst das zu erratende Muster definiert werden. Entweder gibt mal selber eines ein, oder aber lässt eines per Zufall erzeugt.



Abb. 23: GUI – Solver: Set Hidden Pattern



Abb. 24: GUI - Solver: Pattern definieren

Mit der aktuellen Implementation des Solvers ist die komplexeste Version von 8 Farben bei 8 Positionen. Dies ergibt 16'777'216 Möglichkeiten!



Abb. 25: GUI – Solver: Lösung gefunden



Abb. 26: GUI - Solver: Multi-Thread Solver

6.5 Design

In der aktuellen Version stehen 6 Verschieden Designs zur Auswahl: Color, Smiley, Cube, Owl, Car, World und Tux.

👩 Mastermind

- - X



Abb. 27: GUI – Design: Color



Abb. 30: GUI – Design: Owl



Abb. 28: GUI – Design: Smiley

B Mastermind Game Cheat Help in the section of the $\bullet \bullet \bullet$ 🍘 🍩 🍩 🍻 -٠. $\bullet \bullet \bullet$. 🙈 🚕 🙈 $\bullet \bullet$ $\bigcirc \bullet \bullet \bigcirc$. 🚙 🚙 🕋) 💿 ... a 🕋 õõ $\bullet \bullet \bullet$ \odot Congratulations!

Mastermind Game Cheat Help 向向 000 00 **\$** ŶŶ ----**ù ù** 000 ěě. Congratulations!

Abb. 29: GUI – Design: Cube



Abb. 31: GUI – Design: Car

Abb. 32: GUI – Design: World



Abb. 33: GUI – Design: Tux

6.6 Help / About

B Help		About	×
Rules		(i)	Mastermind v1.5
The goal of Mastermind is to discover a combination of coloured pegs selected by the code	maker at random.		© 2012 Iliasa Redzeni Marco Reichmuth
At each turn, the code breaker chooses a combination of coloured pegs and the code make each peg on the right position and a white dot for each peg with the right colour at the wrong breaker chooses a different combination, adding pegs of other colours and switching the or or both.	er returns a black dot for g position. Then the code rder of the coloured pegs,		Felix Rohrer, Philipp Rey
The game ends until the combination is found or the maximum number of attempts is react	hed.	Abb.	35: GUI – About Dialog
Settings			
- Allow duplicate colors: Yes/No			
- Number of pegs 4-10 - Number of attempts: 6-12			
- Icon style: Color, Smiley, Cube, Owl, Car or World			
Functions			
# Game:			
 New: Start new Game New Solver: Act as code maker by setting your own pattern or generate it by random. 			
- Open: Open a saved game			
- Save: Save the current game			
- Exit: Close the Game			
#Cheat:			
- Show one hidden pegs: Displays one reg by random - Show all hidden pegs: Display all Pegs			
ОК			
bb 34: GIII – Help Dialog			

Abbildungsverzeichnis

Abb. 1 Mastermind: Funktionale Sicht (Sequenzdiagramm) 1
Abb. 2 Mastermind: Package Übersicht 2
Abb. 3 Mastermind: Klassen Übersicht 2
Abb. 4 Package Model: Klassenübersicht
Abb. 5 Package Logic: Klassenübersicht
Abb. 6 Package Client: Klassenübersicht 4
Abb. 7 Package Solver: Klassenübersicht 4
Abb. 8 Package Engine: Klassenübersicht 5
Abb. 9: CLI – Hilfe / About
Abb. 10: CLI – Optionen definieren
Abb. 11: CLI – Ungültige Optionen abfangen 9
Abb. 12: CLI – Ungültige Kombination abfangen9
Abb. 13: CLI – Rateversuche
Abb. 14: CLI – Ungültige Rateversuche abfangen 10
Abb. 15: GUI – Startscreen 11
Abb. 16: GUI –Ungültige Eingabe 11
Abb. 17: GUI – Gewonnen
Abb. 18: GUI – Verloren 11
Abb. 19: GUI – Settings Dialog 12
Abb. 20: GUI – Settings Dialog: Ungültige Eingabe 12
Abb. 21: GUI – Cheat: One Peg 12
Abb. 22: GUI – Cheat: Ganze Lösung anzeigen 12
Abb. 23: GUI – Solver: Set Hidden Pattern
Abb. 24: GUI – Solver: Pattern definieren 13
Abb. 25: GUI – Solver: Lösung gefunden 13
Abb. 26: GUI – Solver: Multi-Thread Solver
Abb. 27: GUI – Design: Color
Abb. 28: GUI – Design: Smiley
Abb. 29: GUI – Design: Cube
Abb. 30: GUI – Design: Owl
Abb. 31: GUI – Design: Car
Abb. 32: GUI – Design: World 14
Abb. 33: GUI – Design: Tux
Abb. 34: GUI – Help Dialog
Abb. 35: GUI – About Dialog 15