Vorgehensweise und Entscheidungsfindung

class MainApp (main-Methode)

Diese Klasse setzt das Look&Feel auf den des Systems. Danach wird ein neues Objekt der Klasse SudokuIO erstellt und im Anschluss ein weiteres Objekt der Klasse SudokuFrame, welches das SudokuIO Objekt übergeben bekommt. Anschließend wird die Methode buildFrame() aufgerufen, die den Frame aufbaut.

class SudokuFrame

Die Klasse SudokuFrame besteht aus einem JFrame mit einem Gridlayout(4,3). Die erste Zeile mit den drei Zellen ist für die Knöpfe zum Laden und Speichern, sowie für ein JLabel mit der Aufschrift Sudoku. Die übrigen neun Zellen sind für die einzelnen 3*3 SudokuTable. Die neun 3*3SudokuJTable haben wir in ein zweidimensionales SudokuTable Array gepackt, so haben wir immer ein Objekt, aus dem wir alle weiteren herausholen können. Die Klasse SudokuTable ist eine Erweiterung der JTable Klasse. In der Methode buildFrame() wird zunächst der Frame aufgebaut und mit den entsprechenden Parametern und Knöpfen eingerichtet.
Nun wird ein neues Objekt der Klasse SudokuLogik angelegt, dem auch gleich das JTable Array “tables” übergeben wird, damit wir keine weiteren Abhängigkeiten spatter in der SudokuLogik Klasse haben, genauso wie anfangs in der MainApp, wo wir ein SudokuIO Objekt an diese Klasse übergeben.
In der folgenden verschachtelten for-Schleife werden die einzelnen Zellen des tables-Array mit SudokuTable Objekten befüllt. Mit dem Aufruf der Methode buildTable() wird ein neuer SudokuTable erstellt, mit ein paar Eigenschaften bedacht und anschließend von der Methode returniert. Dieses Objekt wird anschließend in den Frame gepackt.
Nachdem alle neun Table erzeugt wurden, werden diese Mittels des SudokuLogik Objekts mit den Zufallszahlen befüllt. Über frame.pack(); wird die Framegröße gesetzt, zum Anzeigen aller hinzugefügten Komponenten.
Die Methode just4help(); ist nur zum Ausgeben und debuggen des tables-Objekts, also für das Programm nicht von Bedeutung.
Der ActionListener “listenerSaveLoadButton” überwacht den speichern und laden Knopf, je nachdem, welcher Knopf betätigt wurde, lost das ensprechende event aus.
Beim Speichern wird entsprechend die saveSudoku Methode aus SudokuIO gecallt und beim Laden die loadSudoku Methode. Nach dem Laden werden dann die TableModels durch die geladenen Models aus der gespeicherten Datei überschrieben (for-Schleife).

class SudokuLogik

In der Methode fillTables() wird zu Beginn festgelegt, wieviele Zufallszahlen generiert werden sollen. Bei unserem Beispiel liegt der Wert zwischen 20 und 40.
In der anschließenden for-Schleife werden zwei Zufallszahlen zwischen 0 und 2 generiert. Diese beiden Werte holen einen zufälligen Table aus dem SudokuTable Array.
Nun muss geprüft werden, ob die Zahl gültig ist. Das geschieht mittels der Methode “isNumberValidIn”. Nachdem eine gültige Zahl gefunden wurde, wird die entsprechende Zelle auch gleich als nicht editierbar markiert mittels der “setRandomValueAt” Methode aus der Klasse SudokuTable. Die do-while Schleife wird solange durchlaufen, bis ein gültiger Wert gefunden wird.
Die Methode “isNumberValidIn” besteht aus drei Prüfstrukturen. Zunächst wird geprüft, ob die Zahl doppelt im Table enthalten ist, wenn nicht, wird geprüft, ob die Zahl in einem der

anderen Tables in der selben Reihe enthalten ist. Falls nicht, wird noch geprüft, ob die gleiche Zahl in einem anderen Table in der gleichen Spalte steht. Besteht die Zahl alle Prüfungen, wird true zurückgegeben, schlägt eine der Prüfungen fehl, wird false zurückgegeben und die weiteren Tests werden übersprungen.

class SudokuTable

Die Klasse SudokuTable ist eine Erweiterung der JTable Klasse. Neu sind ein Bool-Array, um sich zu merken, welche Zellen im Table editierbar sein sollen und welche nicht. Weiterhin neu ist die Methode setRandomValueAt, die im Prinzip das gleiche tut, wie die JTable eigene Methode setValueAt, lediglich erweitert um die Funktion, das hier die mit Zufallszahlen belegten Zellen im Bool-Array auf true gesetzt werden.
Die im JTable enthaltene Methode isCellEditable wurde überschrieben, diese überprüft nun den entsprechenden Zelleninhalt im Table, enthält diese ein true, befindet sich also eine Zufallszahl darin, gibt die Methode !true, also false zurück, da diese Zellen ja nicht mehr editiert werden dürfen.
Diesen Umweg sind wir gegangen, da wir ursprünglich das Problem hatten, dass wenn man nur prüft, ob ein Feld eine Zahl enthält, man die ursprünglich leeren Felder nicht mehr ändern kann, sobald man eine Zahl eingetragen hat. Der Anwender könnte also seine Eingabe nicht verbessern. So mussten wir einen Weg finden, uns die anfänglich mit Zufallszahlen belegten Felder irgendwie zu merken. Daher die zusätzlichen Methoden und das Bool-Array.
Um ein erstelltes Spiel zu laden, haben wir in dieser Klasse das Interface Serializable implementiert.

class SudokuIO

In dieser Klasse befinden sich lediglich Methoden, um ein generiertes Spiel zu speichern ode r ein bereits gespeichertes Spiel wieder zu laden. Serialisiert werden lediglich die TableModels der SudokuTables. Nach dem Laden einer gespeicherten Datei und dem Setten der geladenen Models aktualisiert sich die Oberfläche selbstständig.

Sonstiges

Auf dem Aufgabenblatt gibt es zwei Tabellen für unterschiedliche Schwierigkeitsstufen. Man könnte die Klasse SudokuLogik dahingehend erweitern, dass man verschiedene Zahlenbereiche für die Schwierigkeitstufen angibt. Also z.B. mittels random Werte zwischen 17 und 20 für schwer, zwischen 21 und 25 für mittel und 26 und 30 für leicht. Man könnte dafür z.B. ein Pulldown Menü nehmen oder drei Knöpfe, eine JMenuBar, … da gäbe es verschiedene Möglichkeiten.

Martin Slowikowski
Matrikelnummer: 199 91 66

Jan-Tristan Rudat
Matrikelnummer: 200 78 52

Teamname: Bernie und Ert

Quellenangaben

Class UIManager
Sun Microsystems, Inc.; Class UIManager
URL: http://download.oracle.com/javase/6/docs/api/javax/swing/UIManager.html
(abgerufen am 03.10.2010)