Sudoku

Samurai um 1860 [1]

Delphi-Programmierpraktikum
Erstellt von Max Rohde

Matr. Nr.: 2848
Fachbereich: Bachelor Wirtschaftsinformatik
E-Mail: winf2848
Fachsemester: 3tes Fachsemester
Verwaltungssemester: 3tes Verwaltungssemester

Anlagen:
CD mit ausführbarer Programmdatei und Quelltexten für Borland Delphi 7

Grundlage für die Funktionalität des Programms:
http://www.fh-wedel.de/~ps2/aufgaben/abschl_all/ss2006/SudokuAufgabe.html#loesen

Richtlinien für Dokumentation:
http://www.fh-wedel.de/~ps2/regelungen/dokurichtlinie.html
und http://www.fh-wedel.de/cis/semesterinfo/richtlinien/doku-richtl.html

Richtlinien für die Object Pascal Quelltexte:
http://www.fh-wedel.de/~ps2/programmierstil/pascal_progstil.html
http://www.fh-wedel.de/~ps2/programmierstil/delphi_progstil.html
http://www.fh-wedel.de/~ps2/programmierstil/inline_doku.html

Inhaltsverzeichnis

A. Benutzerhandbuch

Bei Sudoku handelt es sich um ein Programm, mit dessen Hilfe man Sudokus, beispielsweise aus Zeitungen oder Rätselheften auf den Computer übertragen kann und sie dort speichern, bearbeitend und selbstständig oder mit Hilfe des Programms lösen kann.
Es gibt auch eine Möglichkeit, sich Sudokus generieren zu lassen und diese dann zu lösen.

	5		13				9	1		11			7		
1		9	15	5	3		14	2	7	8			10		12
2	6						15					5	8		9
	7	10	11	6	8			5		12		4	2	1	14
	1	11	8	2				6	3						
	9			3	5	16		7				12	14		1
5	2				1				10		11		4	3	6
	3	16		8	4	15	6		1			10		2	
			3		12				16		6	14		10	
9	14	4	6			1		11				13			
10	16	2	5		14		4	12	13	15	9		1	6	8
	15		7	13								2	9		
13	8	3	4	12	9				11						7
14		5				4					7	11	13		2
15		6	2	16		7		14	12	1	4	8	3	9	5
				14			1		2						10

Ein nicht ganz einfaches Sudoku ...

1. Ablaufbedingungen

Betriebssystem:	Windows 98, ME, 2000, XP
Hardware:	700 Mhz Pentium oder kompatibel 256 MB RAM Grafikkarte (16 Mio. Farben empfohlen)

2. Programminstallation/-start

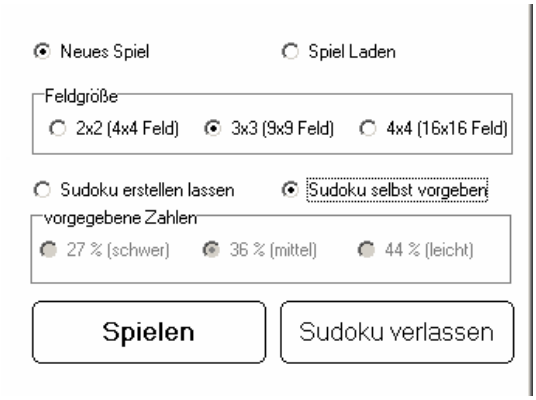
Die Applikation benötigt keine Installation. Die Programmdatei kann ohne Vorbedingungen unter allen unter 1. aufgeführten Systemen ausgeführt werden.

3. Bedienungsanleitung

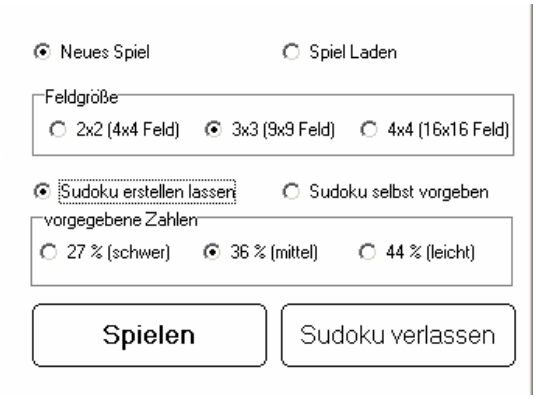
a. Start Dialog

Nach dem Start von Sudoku gelangen Sie zunächst zum Start Dialog. Hier können Sie auswählen, ob Sie ein vom Computer generiertes Sudoku, ein selbst vorgegebenes oder geladenes Sudoku spielen möchten.

Ein Sudoku mit Hilfe des Computer lösen lassen
Wenn Sie ein Sudoku, beispielsweise aus einer Zeitschrift am Computer lösen wollen, wählen Sie beim Start Dialog zunächst „Neues Spiel“ dazu die betreffende Feldgröße (diese ist in den meisten Fällen 9x9) und dann die Option „Sudoku selbst vorgeben“. Um das Spiel zu beginnen klicken Sie auf „Spielen“.



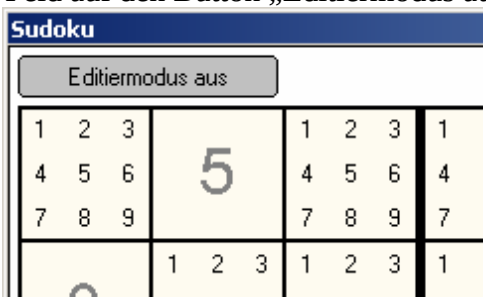
Einstellungen für das Vorgeben eines Sudokus
Auch wenn Sie sich ein zufälliges Sudoku vorgeben lassen wollen und wählen Sie zuerst „Neues Spiel“ anschließend die Feldgröße. Danach klicken Sie auf die Option „Sudoku erstellen lassen“. Danach kann der Schwierigkeitsgrad angegeben werden. Dieser bezieht sich allein auf die Anzahl der vorgegebenen Zahlen (im Prozent aller Zahlen). So werden bei der Schwierigkeit leicht und einer Feldgröße von 81 Feldern 36 Zahlen vorgegeben.
Um das Spiel zu beginnen klicken Sie auf „Spielen“.



Ein vorher gespeichertes Sudoku laden
Um ein altes Spiel zu laden, wählen Sie zunächst „Spiel laden“. Um eine Datei zu öffnen klicken Sie auf den „Laden“-Button. Die Datei kann über einen windows-typischen Dialog ausgewählt werden.
Um mit dem Spielen zu beginnen klicken Sie auf den Button „Spielen“.

b. Sudokus editieren

Wenn Sie mit der Option „Sudoku selbst vorgeben“ gestartet haben, oder auf den Button „Editiermodus an“ geklickt haben, befinden Sie sich im Editiermodus. Sie können die Zahlen beim 4x4 und 9x9 Feld durch Klicken auf die kleinen Ziffern in den Feldern setzten, durch Klicken auf die Zahlen diese wieder entfernen. Beim 16x16 Feld öffnet sich beim klicken auf ein leeres Feld ein kleiner Hilfsdialog, auf dem Sie per Mausklick die gewünschte Ziffer auswählen können.
Alle Zahlen, die Sie im Editiermodus setzten, werden grau angezeigt.
Wenn Sie mit dem Spielen beginnen oder fortfahren wollen, klicken Sie am links oben über dem Feld auf den Button „Editiermodus aus“.

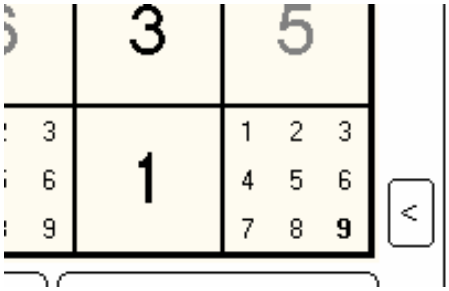


Durch Klicken auf den Button „Editiermodus an“ können Sie jederzeit in den Editiermodus zurückkehren.

c. Sudoku spielen

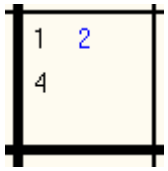
Zahlen setzten/ löschen
Die Zahlen setzten/löschen Sie wie beim Editiermodus beschrieben.

Erste/ letzte Zeile lösen lassen
Um die erste bzw. letzte Zeile lösen zu lassen klicken Sie auf die kleinen Buttons „<“ die sich rechts neben dem Feld neben der untersten und obersten Zeile befinden.



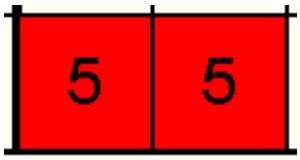
Button letzte Zeile lösen

Sudoku Lösungshilfen anzeigen
Um Hilfe bei der Lösung des Sudokus zu erhalten, klicken Sie auf den Button „Hilfe ein“. Diese Funktion ist nur für 4x4 und 9x9 Sudokus verfügbar.
Die Hilfe markiert Zahlen rot, wenn diese die einzige Möglichkeit in ihrer Zeile, Spalte oder ihrem Block sind.
Blau werden die jeweils letzten zwei Möglichkeiten für Zahlen in einem Block eingefärbt.



Zelle mit Lösungshilfe

Sudoku lösen/ prüfen lassen
Wenn Sie ein Sudoku gelöst haben oder ein Sudoku vom Computer lösen lassen wollen, klicken Sie auf den Button „Lösen/ Prüfen“. Es werden die fehlenden Ziffern ergänzt, oder eine Fehlermeldung ausgegeben, wenn das Sudoku falsch oder unlösbar ist.
Wurden Zahlen eingetragen, die eine Regelverletzung verursachen, so werden diese rot hervorgehoben.

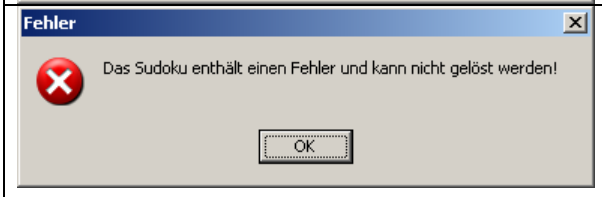
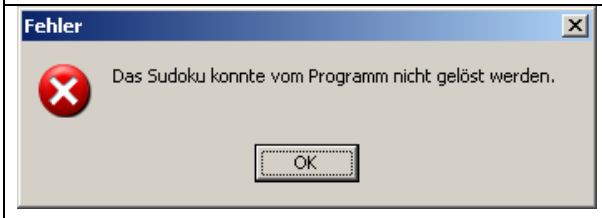
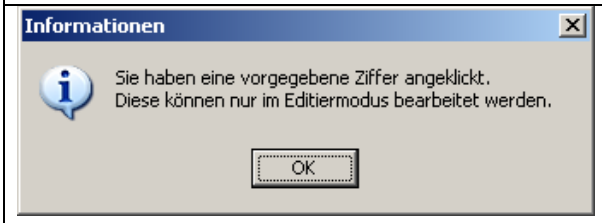


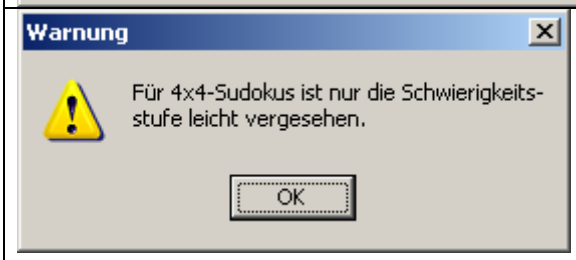
Zahlen, die als nicht regelkonform markiert wurden

Sudoku speichern
Um ein Sudoku zu speichern klicken Sie auf den Button „Speichern“. Hier können Sie in einem windows-typischen Dialog den Speicherort und Namen für Ihr Sudoku angeben.

Spiel beenden
Wenn Sie das Spiel beenden möchten, um zum Start Dialog zurückzukehren, klicken Sie auf den Button „Spiel beenden“.

4. Fehlermeldungen

Fehlermeldung	**Fehlerursache**	**Behebungsmaßnahme**
	Die Datei wurde zwischenzeitlich gelöscht, oder ist von einem anderen Programm geöffnet	Wählen Sie eine andere Datei aus, oder beenden Sie alle Programme, die auf diese Datei zugreifen könnten.
	Die ausgewählte Datei hat kein gültiges Format.	Wählen Sie eine gültige Datei aus.
	Die ausgewählte Datei ist von einem anderen Programm geäffnet.	Beenden Sie alle Programme, die auf diese Datei zugreifen könnten.
	s. Meldung	Entfernen Sie gegebenenfalls schon gesetzte Ziffern und versuchen Sie das Lösen erneut.
	Sie haben ein Sudoku mit sehr ungünstiger Verteilung eingegeben, dass aufgrund der Vielzahl an Lösungswegen nicht gelöst werden konnte.	Geben Sie ein anderes Sudoku ein.
	Sie haben auf eine „graue“ Ziffer geklickt.	Klicken Sie auf den Button „Editiermodus an“, um auch vorgegebene Zahlen zu bearbeiten.

	Sie haben auf „Spielen“ geklickt, aber noch keine Datei ausgewählt.	Klicken Sie auf den Button „Laden“ und wählen Sie eine gültige Datei aus.
	Sie wollten die Schwierigkeitsstufe „schwer“ oder „mittel“ bei gleichzeitiger Anwahl eines 4x4-Feldes auswählen.	Wählen Sie die Schwierigkeitsstufe leicht.

5. Wiederlaufbedinungen

Kommt es während der Programmausführung zu einem Systemabsturz, kann SUDOKU danach wieder normal gestartet werden. Das zur Zeit des Absturzes gerade bearbeitete Sudoku ist allerdings verloren.

B. Programmierhandbuch

1. Entwicklungskonfiguration

Sudoku wurde auf folgendem System entwickelt:

System:
Microsoft Windows XP
Professional
Version 2002
Service Pack 1

Computer:
AMD Athlon(TM) XP 2400+
2.00 GHz
512 MB RAM

Dabei kam folgende Software zum Einsatz:
Borland Delphi

Version 7.0 (Build 4.453)
Copyright © 1983-2002
Borland Software Corporation
http://www.borland.com

2. Problemanalyse und Realisation

a. Problemanalyse

Es sei zwischen vier Problemfeldern unterschieden: Datenstruktur, Darstellung, Bedienung und Lösung eines Sudokus.

Datenstruktur
Die maximale Größe eines Sudokus beträgt laut Aufgabenstellung 16x16 Felder, in denen jeweils eine Zahl zwischen 1 bis maximal 16 bzw. der Zustand leer abgespeichert werden muss.
Dazu muss hinterlegt sein, ob eine Zahl vom Sudoku vorgegeben wurde, oder ob sie der Benutzer eingetragen hat.
Neben den Zahlen muss natürlich die Größe des Feldes vermerkt werden.

Darstellung
Es müssen Sudoku Felder mit einer Blockgröße von 2x2, 3x3 und 4x4 dargestellt werden können (Feldgröße 8,9,16). Des Weiteren müssen im 3x3 Sudoku Hilfszahlen optional eingeblendet werden können.

Bedienung
Die Zielgruppe des Programms ist eher der Durchschnitts-Anwender, da die Funktionalität nicht auf einen eingeschränkten Benutzerkreis schließen lässt. Die wichtigen Funktionen müssen folglich einfach bedienbar und erreichbar sein und dürfen nicht in Untermenüs oder komplizierten Dialogen versteckt sein.

Lösen/ Buchführung
Es sollen 2x2, 3x3 und 4x4 Sudokus bearbeitet werden können. Diese müssen auch gelöst werden. Die Aufgabenstellung gibt vor, dass ein Backtracking-Algorithmus mit vorherigem Ausschluss von logisch ermittelbaren Möglichkeiten benutzt werden soll.

b. Realisationsanalyse

Datenstruktur
Es gibt die Möglichkeit, das Feld in einer linearen Liste (oder ähnlichen Datenstruktur) mit Hilfe von Pointer zu realisieren oder alle Daten in einem Feld fester Größe abzulegen.
Die Umsetzung mit Pointer ist aufwendiger und damit fehleranfälliger. Ein Feld fester Größe zu benutzen, bringt den Nachteil mit sich, dass keine größeren Sudokus als 16x16 Felder angeboten werden können.
Die Aufgabe fordert nur 16x16 große Sudokus, daher wird mit Feldern fester Größe gearbeitet.
Für die Struktur dieser gibt es auch mehrere Möglichkeiten: Auf ein Sudoku gibt es aus seiner Natur heraus immer drei relevante Sichten: Die einzelnen Spalten, die einzelnen Zeilen und die einzelnen Blocks. Der Zugriff auf die Zeilen und Spalten ist am einfachsten über ein zweidimensionales Feld, der Zugriff auf die einzelnen Blocks über eine Aufteilung des Gesamtfeldes in Blocks, in denen dann als zweidimensionales Feld dann die einzelnen Zahlen stehen.

Des weiteren gibt es die Möglichkeiten, die Daten in einer eigenen Unit zu kapseln (d.h., die Daten liegen in der Unit als unit-globale Variable – andere Units greifen über die Funktionen der Daten-Unit auf die Daten zu), was allerdings den Nachteil mit sich bringt, dass es nahezu unmöglich ist Kopien der Daten anzufertigen. Der Vorteil, dass die Datenhaltung bei der Kapselung ohne Folgen für das restliche Programm geändert werden kann, überwiegt meiner Ansicht nach erst beim Einsatz objektorientierter Programmierung. Die naheliegenste Lösung ist das Speichern der Daten in einer Variable, die durch Call by Reference an die Verarbeitungsroutinen übergeben wird und in den Instanzen der Bearbeitungs-Formulare gespeichert wird. Durch eine einfache Zuweisungsoperation kann man die Daten so bei Bedarf kopieren (bspw. zum Berechnen der Lösung).

Darstellung
Es gibt einmal die Möglichkeit über ein TCanvas-Objekt oder die Windows-GDI das Feld und die Zahlen manuell ausgeben zu lassen, und die Möglichkeit, das Feld mit Delphi VCL-Komponenten zu realisieren.
Erstere Möglichkeit bieten den Vorteil einer wesentlich besseren Performance sowie der größeren Möglichkeiten der Darstellung, allerdings den Nachteil einer umständlichen Umsetzung.
VCL-Komponenten zu benutzen bietet den überwiegenden Vorteil, dass Darstellung (Paint) und Interaktivität (Click) in einer Komponente gebündelt sind, was die Umsetzung erleichtert.

Bedienung
Vor dem eigentlichen Spielen sind einige Parameter anzugeben. Zum Beispiel, ob man ein neues Spiel erstellen möchte, oder ein bereits vorhandenes laden möchte. Man könnte das Spielfeld sperren, solange der Benutzer diese Eingaben noch nicht gemacht hat. Diese zusätzlichen Möglichkeiten „Spiel laden“, „Spiel neu erstellen“ sind jedoch nicht für den eigentlichen Spielablauf von Bedeutung, weshalb sie beim Spielen auch nicht zur Verfügung stehen sollten. Hier ist die beste Alternative einen Start-Dialog vorzulagern, in dem der Benutzer Spiele erstellen und laden kann und zu dem er aus dem laufenden Spiel zurückkehren kann.

Lösen/ Buchführung
Wie beschrieben ist ein Backtracking-Algorithmus vorgegeben. Es ist jedoch die Frage zu entscheiden, wie aufwendig die Berechnung des logischen Ermittelns von Lösungen im Verhältnis zum einfachen Ausprobieren durch Rekursion und Backtracking (hier bietet sich eine rekursive Lösung an, da die maximale Rekursionstiefe auf 256 Aufrufe (im 16x16 Feld) beschränkt werden kann.) ausfallen soll.
Das logische Ausprobieren kann gegebenenfalls mehr Zeit verbrauchen als das vorgehen mit „Brute Force“. Welcher Weg allgemein der beste ist, kann nicht ermittelt werden, da die Eignung einer Lösung stark von der Struktur des zu lösenden Sudokus abhängig ist.
Die einzelnen Schritte im Backtracking sollten jedoch möglichst wenig Zeit verbrauchen.

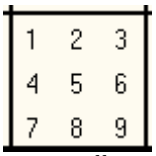
c. Realisationsbeschreibung

Datenstruktur

Zuerst sei die Terminologie der Typen erläutert:
Unter einem Spiel sind Daten zusammengefasst, mit denen der Benutzer während der Durchführung eines Sudoku-Spiels interagiert, oder die zur Durchführung der Spielfunktionen benötigt werden.
Ein Feld besteht aus mehreren Blöcken in denen sich wiederum die Zellen befinden. Ein 3x3-Sudoku-Feld besteht zum Beispiel aus 3 mal 3 Blöcken mit jeweils 3 mal 3 Zellen. In jeder Zelle gibt es die Zahl oder den Wert, der die Zahlen-Eingabe des Benutzers oder die vorgegebenen Zahlen beschreibt.
Nicht nur für die Anzeige der Hilfen, sondern auch der Lösungsalgorithmus macht eine weitere Differenzierung der leeren Zellen sinnvoll: sie können theoretisch mit NxN (N = Feldgröße) unterschiedlichen Ziffern gefüllt werden.
Mit den Ziffern angefangen wird nun die Realisierung in Delphi von Unten nach Oben beschrieben. Eine genaue Beschreibung zu jedem Typ ist auch in der Inline-Dokumentation zu finden:

```
TSZiffer = record
  Status: TZifferStatus;
  MyLabel: TLabel;
  Alternativen: TSWert;
end;
```

Der Status einer Ziffer gibt an, wie Sie auf dem Spielfeld dargestellt wird: gar nicht, normal, rot oder blau. Die Eigenschaft MyLabel ist nur Spielfelder aktiv, die gerade visuell dargestellt werden. Die Prozedur AddFeld() (uDarstellung) legt hier für jede Ziffer den Zeiger zum passenden Label. So kann später einfach, wenn zum Beispiel der Status der Ziffer geändert wird, die richtige Komponente gefunden werden und in ihrem Erscheinungsbild den Daten angepasst werden. Dies geschieht zum Beispiel in der Prozedur SetZifferStatus() (uDarstellung).
Der Wert Alternativen wird zur Lösung des Sudokus benötigt. Er gibt an, wie viele Alternativen es minimal in dem Block/ der Spalte/ der Zeile in der die Ziffer steht, gibt. Gibt es zum Beispiel eine Alternative (Alternativen=1) so muss die Ziffer für ihre Zelle gesetzt werden (Sie wird bei eingeschalteter Hilfe-Funktion rot angezeigt.).
Der Status wird unabhängig von den Alternativen benötigt, da die Aufgabenstellung vorsieht, dass nur die letzten zwei Möglichkeiten in Blocks blau eingefärbt werden sollen, nicht aber die letzten zwei Möglichkeiten einer Zeile oder eines Blocks.



Neun Ziffern

```
TSLinIndex = 1..cMaxFeldGroesse;
TSZiffern = array [TSLinIndex] of TSZiffer;
```

Der Ziffern-Array stellt alle Möglichkeiten von Zahlen in einer Zelle als einfache Liste dar.

Eine Zelle mit Wert

```
TSZelle = record
  Wert: TSWert;
  Ziffern: TSZiffern;
  Vorgegeben: Boolean;
  MyLabel: TLabel;
  MyShape: TShape;
end;
```

Die Eigenschaften der Zelle unterteilen sich einmal in den Wert, der eine Zahl zwischen Null und der Feldgröße annehmen kann (0 steht dabei für Zelle nicht gesetzt), in die eben beschriebenen Ziffern, den Boolean-Wert Vorgegeben, der angibt, ob der Wert der Zelle zu den vorgegebenen gehört, sowie den Pointern MyLabel und MyShape, die analog zur Eigenschaft MyLabel der Ziffer verwendet werden.

```
PSZelle = ^TSZelle;
TSZellenZeile = array[TSLinIndex] of PSZelle;
TSZellen = array[TSLinIndex] of TSZellenZeile;
```

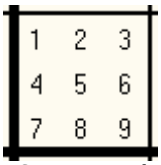
Der Zellen-Array wird dynamisch in der Prozedur FeldInitialisieren (uFeld) mit Pointern, die auf die einzelnen Zellen zeigen, initialisiert. Er ermöglicht den Zugriff auf das Feld in der Zeilen/Spalten-Sicht.

```
TSBlockZeile = array[TSIndex] of TSZelle;
TSBlock = array[TSIndex] of TSBlockZeile;
TSFeldZeile = array[TSIndex] of TSBlock;
TSFeld = array[TSIndex] of TSFeldZeile;
```

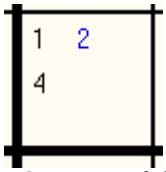
Der Block-Array enthält Blockgröße * Blockgröße Zellen. Das Feld-Array enthält wiederum Blockgröße * Blockgröße viele Blöcke und stellt damit das gesamte Sudoku-Feld dar. Das Feld-Array erlaubt den Zugriff auf das Feld aus Block-Sicht.

```
TSSpiel = record
  SFeld: TSFeld;
  SZellen: TSZellen;
  BlockGroesse: TBlockGroesse;
  FeldGroesse: TFeldGroesse;
  Schwierigkeit: TSchwierigkeit;
  ZeigeStatus: Boolean;
  BenutzeZiffern: Boolean;
  EditierModus: Boolean;
  AllesNichtErlaubt: TIndexSet;
  NichtErlaubtSpalte: array[TSLinIndex] of TIndexSet;
  NichtErlaubtZeile: array[TSLinIndex] of TIndexSet;
  NichtErlaubtBlock: array[TSIndex, TSIndex] of TIndexSet;
end;
```

Der Record Spiel enthält alle für die Durchführung eines Sudoku-Spiels relevanten Daten. Die eben erwähnten Strukturen Feld und Zellen, die den Zugriff auf die Zellen aus unterschiedlichen Sichten ermöglichen, daneben aber noch weitere Angaben: Blockgröße muss bei der Erstellung eines Sudukus angegeben werden. Die Feldgröße wird daraus in der Prozedur FeldInitialisieren (uFeld) berechnet.
Die Schwierigkeit wird bei der Erstellung eines Feldes benötigt, und gibt an, wie viele Felder vorgegeben werden sollen.
Die Eigenschaft ZeigeStatus gibt an, ob die Hilfen (Ziffern verschwinden, oder werden rot oder blau dargestellt) angezeigt werden sollen.

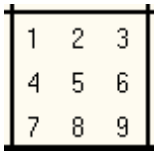


ZeigeStatus ausgeschaltet



ZeigeStatus eingeschaltet

BenutzeZiffern gibt an, ob die kleinen Ziffern in den Zellen angezeigt werden sollen; beim 4x4 Sudokus wird dieser Wert zum Beispiel deaktiviert, da das Feld durch die vielen Ziffern unübersichtlich werden würde.



BenutzeZiffern eingeschaltet


BenutzeZiffern ausgeschaltet

Editiermodus ist True, wenn graue Zahlen gesetzt und gelöscht werden können.
Die folgenden Eigenschaften sind aus Performance-Gründen in das Spiel-Rekord aufgenommen. AllesNichtErlaubt ist die Menge von Ziffern, die der maximalen Menge von Ziffern in einer Zelle bei der aktuell gewählten Blockgröße entspricht. Der Wert wird in der Prozedur FeldInitialisieren gesetzt.
NichtErlaubtSpalte, NichtErlaubtZeile, NichtErlaubtBlock sind ebenfalls Mengen von Ziffern, die diejenigen Ziffern eines bestimmten Blocks, einer Zeile, einer Spalte entsprechen, die nicht mehr gesetzt werden dürfen. Die Arrays werden in der Prozedur SetZifferStatus (uLogik) mit ersten Ziffern gefüllt und im Backtracking-Teil des Sudokus-Lösens ständig entsprechend den gesetzten Ziffern aktualisiert. So kann sicher gegangen werden, dass durch Setzten einer Ziffer kein Regelverstoß verursacht wird.

Die Entscheidung fiel hier, wie gesagt, für einen zusammenfassenden Record und gegen eine Daten-Unit. Die Variablen von TSSpiel werden in den Formularen bei Bedarf im Private-Teil gespeichert und dann bei Bedarf per Call by Reference an die Verarbeitungs-Routinen übergeben. Es ist sehr wichtig, dass Call by Reference ständig einzuhalten, nicht nur aus Gründen der Performance, sondern auch, weil die Zeiger im Zellen-Array nach einer Call by Value-Übergabe auf die falschen Zellen zeigen.

Ein Sudoku Spiel wird gespeichert, in dem in eine typisierte Datei genau ein Datensatz des Typs TSSpiel geschrieben wird (uDatei). Dabei wird natürlich überflüssige Information mitgespeichert (das Zellen-Pointerfeld, die Zeiger

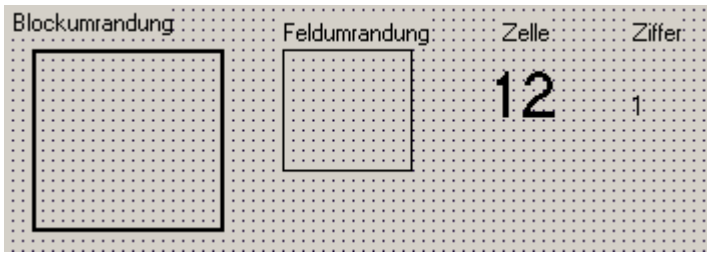
auf die Komponenten), da die Größe der Datei jedoch nur bei ca. 50 kb liegt und das Schreiben und Lesen von Sudoku spielen nur einen verschwinden geringen Anteil an der Laufzeit ausmacht, würde hier zusätzlicher Entwicklungsaufwand (z.B. Speichern als Textdatei/ XML/ ...) keinen im Verhältnis stehenden Nutzen erbringen.

Um die Verarbeitung von Sudokus zu vereinfachen wurde zusätzlich der Typ der Zellen-Liste eingeführt:

```
TZellenPipeItem = record
  V_w, V_h: TSLinIndex;
  A: TSWert;
  Z: TSLinIndex;
end;

TZellenNo = 0..cMaxFeldGroesse*cMaxFeldGroesse;

TZellenPipe = array[TZellenNo] of TZellenPipeItem;

TShortZellenPipe = array[TSWert] of TZellenPipeItem;
```

Jedes Element dieser Liste verweist über die Eigenschaften V_w (Spalte) und V_h (Zeile) auf eine Zelle auf dem Sudoku-Feld. Die Eigenschaften A und Z charakterisieren für die Liste, die von der Loesen-Prozedur verwendet werden, diejenige Ziffer einer Zelle, die die wenigsten Alternativen aufweist. Nach diesen Eigenschaften kann eine Zellen-Liste auch sortiert werden (SortZellenPipe Unit uFeld).
Es gibt zwei unterschiedliche Formen von Listen die ZellenPipe und die ShortZellenPipe. Erstere wird verwendet, wenn die Zellen eines ganzen Feldes gespeichert werden sollen, letztere, wenn nur eine Spalte, Zeile oder ein Block gespeichert werden soll. Die ShortZellenPipe erlaubt für logische Regeln, die auf Zeilen und Spalten identisch angewendet werden sollen, die gleiche Funktion zu benutzen. Ein Beispiel dafür ist die Funktion BerechneLockedCandidates2 (Unit uLogik).

Darstellung
Die wichtigsten Darstellungsroutinen sind in die Unit uDarstellung ausgelagert. Auf dem Formular frmDarstellung befinden sich einige Vorlagen, die zur Generierung des Spielfeldes verwendet werden.



Vorlagen auf frmDarstellung

Die Realisierung der Darstellung folgt der Datenstruktur. Jede Zahl wird durch ein Label beschrieben, jede darunter liegende Ziffer ebenso. Wenn eine Zahl zugewiesen wird, so wird das Label dieser Zahl angezeigt, wenn keine Zahl zugewiesen wurde, werden die Ziffern-Label angezeigt. Schaltet der Benutzer die Hilfe-Funktion ein, so wird über die Font-Eigenschaft der Labels die

geforderte Farbe zugewiesen. Beim Starten eines Spiels wird die Prozedur AddFeld des Formular frmDarstellung aufgerufen (hierfür wird zunächst eine Instanz des Formular erstellt), die das Feld in Form von TShape-Komponenten (für das Gitter) und den oben beschrieben Labels auf einem Panel auf der Instanz des frmSpiel-Formulars erstellt. Die hinzugefügten Komponenten werden durch das Freigeben des frmSpiel-Formulars automatisch freigegeben. Des Weiteren wird durch die AddFeld-Prozedur im Feld-Array zu jeder Zahl und zu jeder Ziffer ein Verweis zu der betreffenden Komponente hinterlegt, dass beim Ändern der Stati auch einfach die Darstellung der Daten geändert werden kann.

Bedienung

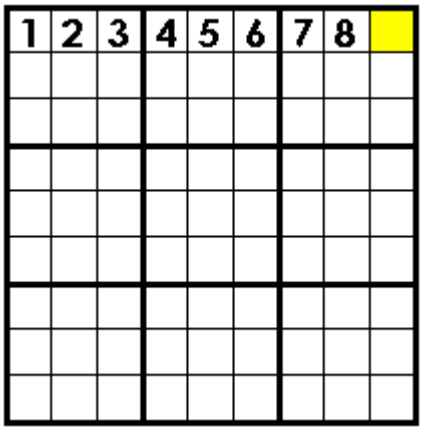
Im Start-Dialog stellt der Benutzer ein, ob er ein Spiel Neu erstellen oder laden möchte. Durch Auswahl des entsprechenden Radio-Buttons wird die jeweils passende Seite des darunter liegenden Notepads geöffnet. Durch Klicken auf den Button „Spielen“, wird ein Spiel-Rekord erstellt, in den alle Spiel-relevanten Daten gemäß den vorher eingetragenen Benutzer-Eingaben gespeichert werden.
In dem darauf folgenden, modalen Spiel-Dialog können wird durch Klicken auf eine Zahl die ZelleClick Ereignisbehandlungsroutine aufgerufen, in der zunächst das entsprechende Daten-Feld im Feld ermittelt wird, für das dann der Wert des Inhalts (Value) wieder auf Null gesetzt wird. Durch Aufrufen der Prozedur SetZelle, die sich in der Hilfs-Unit uFeld befindet, wird die Zelle in das Feld geschrieben und durch den Aufruf von RefreshFeld aus der Unit uDarstellung der veränderte Status der Labels gesetzt (es wird zwar für alle Labels der Status neu gesetzt jedoch nur die geänderten werden neu gezeichnet – das erledigt die VCL automatisch).
Des weiteren sind die wichtigsten Funktionen durch große Buttons dargestellt. Die eher weniger verwendeten Funktionen „erste/ letzte Zeile lösen“ und „Editiermodus ein“ sind eher kleiner gehalten.
Für das Lösen und Prüfen eines Sudokus wurde der gleiche Button verwendet, da sich der Button der gewünschten Funktion leicht anpassen kann, in dem geprüft wird, ob das Feld vollständig gefüllt ist, oder nicht.

Lösen/ Buchführung

Zuerst sei die Buchführung beschrieben, da diese auch Grundlage für das Lösen des Sudokus ist.
Die Buchführung wird in der Prozedur SetZifferStatus (uLogik) berechnet.

Singeltons

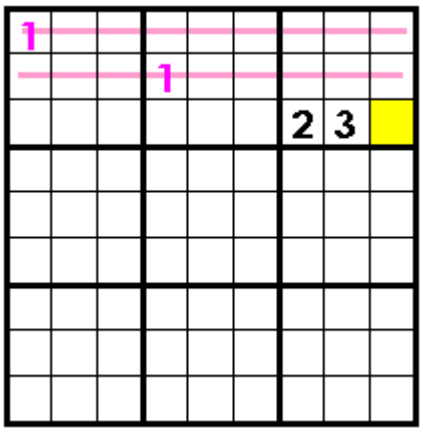
Zuerst werden alle Ziffern entfernt, die beim Setzen sofort einen Regelverstoß verursachen würden (SetZifferSichtbar uLogik). Dabei wird einfach überprüft, welche Zahlen sich im Block, in der Spalte oder Zeile schon befinden und diese dann aus den Möglichkeiten der restlichen Ziffern entfernt.
Dies entspricht der Regel „Singeltons“. Das in der folgenden Darstellung gelb markierte Feld, enthält im Programm eine rote Ziffer (=eindeutige Möglichkeit).

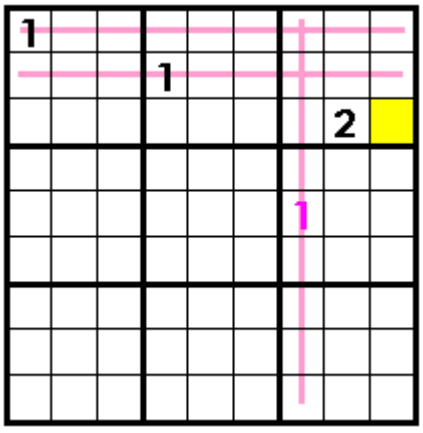


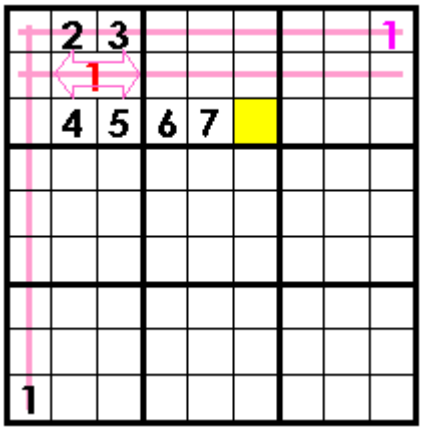
[2]

Parallele Linien/ Kreuzende Linien

In der nächsten Stufe, wird in der Funktion BerechneZifferRotBlau (uLogik) für jede Ziffer in jeder Zelle ermittelt, wie viele alternative Setz-Möglichkeiten es für sie noch in ihrem Block/ ihrer Zeile/ ihrer Spalte gibt. Gibt es nur eine Alternative, so muss diese eindeutig gewählt werden. Folgend sind einige Beispiele aufgeführt, in denen die gelben Zellen jeweils eine rote Ziffer enthalten.





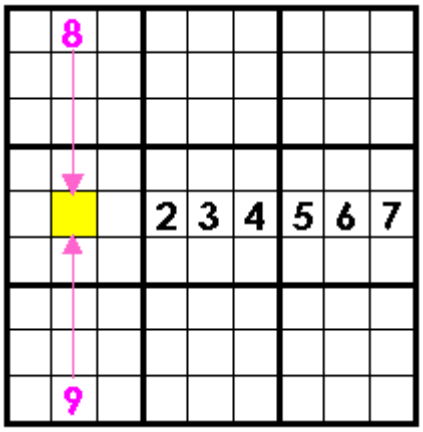


[2]

Die jeweiligen Alternativen werden für die Ziffern gespeichert. Diese Information wird vom Lösungs-Algorithmus verwendet, in dem er immer zuerst die Ziffer mit den wenigsten Alternativen zuerst ausprobiert (im Backtracking-Teil).
Des weiteren wird in der Prozedur BerechneZifferRotBlau, den letzten Möglichkeiten der Status rot und den vorletzten eines Block der Status blau zugeordnet.

Blockade 2

Als letztes wird, wenn eine Ziffer die letzte übergebliebene in ihrer Zelle ist, ihr auch die Alternativenzahl eins und der Status rot zugeordnet. Das entspricht folgendem Bild:

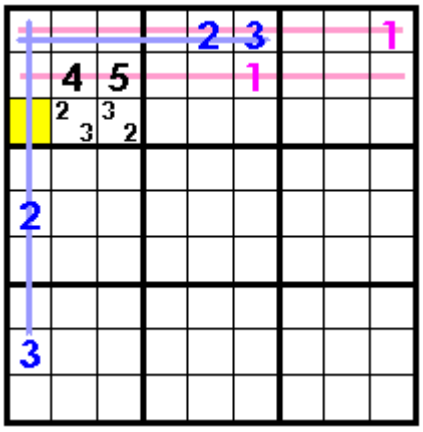


[2]

Beim Aufruf der Prozedur Loesen (uLogik) wird zunächst eine Kopie des aktuellen Spiels angelegt.
Darauf werden ein paar weitere logische Ausschlussregeln angewandt:

Blockade 3

Wenn es eine Teilmenge von Ziffern in einer Spalte/ einem Block/ einer Zeile gibt, die genau so häufig auftritt wie sie mächtig ist, so werden alle Alternativen, die in den gleichen Zellen wie die betreffende Teilmenge stehen, aber nicht zur Teilmenge gehören, aus den Alternativen entfernt.

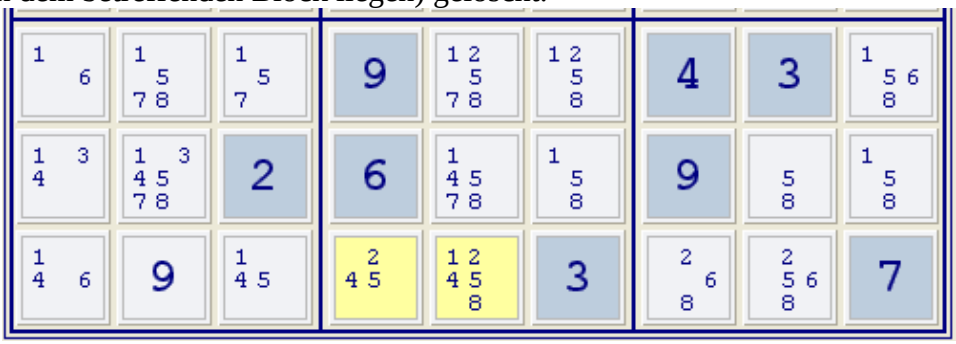


[2]

Blockade 3: Die 2 und die 3 können aus der gelb markierten Zelle entfernt werden

Locked Candidates

Tritt eine Ziffer in dem Abschnitt einer Zeile/ einer Spalte, die in einem bestimmten Block liegt, genau so oft auf wie in dem bestimmten Block, so werden alle anderen Möglichkeiten dieser Ziffer in der Zeile/Spalte (die nicht in dem betreffenden Block liegen) gelöscht.



[3]

Die 2en in den gelb markierten Felder können gelöscht werden

Dann wird versucht durch wiederholtes Setzen der roten Ziffern (die Ziffern, die letzte Alternative sind) und anschließendem Neuberechnen der Alternativen durch BerechneZifferStatus, sowie den vorgestellten logischen Ausschlussregeln, die Anzahl der zu lösenden Felder zu reduzieren (in der Funktion Loesen uLogik). Dies wird so lange durchgeführt, wie die logischen Ausschlussregeln Zahlen zum Setzen finden.
Wurde hierbei kein Regelverstoß verursacht, wird versucht, die übrig gebliebenen Zellen per Backtracking zu füllen.
Die Zellen werden durch die Prozeduren AddToPipe, AddToPipeReverse und AddToPipeSchlange (alle uFeld) einer Zellen-Liste hinzugefügt. Es ergeben sich insgesamt fünf unterschiedliche Reihenfolgen. Die nacheinander ausprobiert werden, wenn eine Reihenfolge zu keinem Ergebnis in angemessener Zeit geführt hat. Dies ist notwendig, da oftmals in einer anderen Herangehensweise, das Sudoku einfacher gelöst werden kann. Damit nicht zu viel Rechenzeit für einen Weg aufgebracht wird, wird bei jedem Funktionsaufruf der Parameter VersucheUebrig übergeben; der am Anfang mit dem Wert cMaxSteps aus der Unit uKonstanten initialisiert und bei jedem Aufruf um eins dekrementiert wird, bis, wenn der Wert eins erreicht ist, die Lösung unterbrochen wird.
Für jede der Reihenfolgen wird die Funktion LoesePipe (uLogik) aufgerufen.

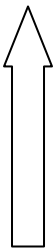
Die Vorgehensweise von LoesePipe sei hier kurz vereinfacht in Pseudo-Code wiedergegeben:

```
Ist das Ende der Liste erreicht?
  Ja: Gebe Ergebnis Ja zurück
  Nein:
    Kann eine Ziffer gesetzt werden, die keinen
    Regelverstoß verursacht?
      Ja: Gebe das Ergebnis zurück, dass das Lösen der
          übrig gebliebenen Liste zurückliefert
      Nein: Gebe das Ergebnis Nein zurück
```

Dabei wird jeweils als erstes versucht, die Ziffer zu setzten, die sich bei vorherigem Anwenden der Logik-Regeln als am wahrscheinlichsten herausgestellt hat (ihr Alternativen-Wert ist der kleinste der Zelle).
Am Anfang jedes Durchlaufes der Funktion wird, soweit der Modus „Intelligent“ gewählt wurde (dieser ist nur sinnvoll, wenn eine gewisse Anzahl von Zellen vorgegeben ist), versucht, durch Logik Möglichkeiten zu setzten. Wurden welche gefunden, so werden sie gesetzt und in eine Liste geschrieben. Führt die spätere Anwendung des Backtrackings in eine Sackgasse, so werden die Zellen, die sich in dieser Liste befinden wieder auf leer gesetzt.

3. Programmorganisationsplan

Aus Übersichtlichkeits-Gründen wird hier nicht jede Unit einzeln mit ihren Verknüpfungen aufgezeigt, sondern nur eine Unterteilung in Ebenen vorgenommen. Die Units binden dabei Units einer höheren Ebene (die höchste ist Utility) ein, eine Unit eine übergeordneten Ebene bindet jedoch keine untergeordneten Units ein. Units die sich auf einer Ebene befinden binden sich nicht gegenseitig ein.

Einbindung	Ebene	Units		
	Utility	uUtils		
	Konstanten	uKonstanten	Options.inc	
	Typen	uTypen		
	Hilfe	uDarstellung	uDatei	uFeld
	Berechnung	uLogik		
	Hilfs-Formulare	u16Ziffern	uLogger	uProgress
	Formulare	uSpiel		
	Hauptformular	uMain		

4. Programmtest

Die Testfälle sollen (soweit nicht anders angegeben) jeweils für Sudokus der Blockgröße 2x2, 3x3 und 4x4 durchgeführt werden.

Testfall	**Erwartetes Ergebnis**	**Erzieltes Ergebnis**
Ein Sudoku zum Zahlen vorgeben öffnen	Leeres Sudoku mit aktiviertem Editiermodus wird geöffnet.	Ok
Ein Sudoku generieren lassen	Es kommt ein Sudoku mit zufälligen Zahlen, die grau sind	Ok
Sudoku speichern	Die Datei wird an der ausgewählten Stelle gespeichert,	Ok
Korrektes Sudoku lösen lassen	Alle nicht gefüllten Felder werden gefüllt	Ok
Fehlerhaftes Sudokus lösen lassen	Es erscheint eine Meldung, dass Sudoku unlösbar sei. Zahlen, die einen Regelverstoß verursachen werden rot markiert.	Ok
Korrektes, komplett ausgefülltes Sudoku überprüfen	Es erscheint die Meldung, Sudoku sei korrekt.	Ok
Fehlerhaftes, komplett ausgefülltes Sudoku überprüfen	Es erscheint eine Meldung, Sudoku sei fehlerhaft.	Ok
Hilfe Einschalten (nur 2x2 und 3x3)	Es werden unmögliche Alternativen ausgeblendet, die letzten zwei Möglichkeiten (an Ziffern) im Block blau gefärbt und die letzten Möglichkeiten in Zeile/Spalte/Block rot eingefärbt	Ok
Hilfe Ausschalten (nur 2x2 und 3x3)	Es werden wieder alle Ziffern angezeigt	Ok
Erste/ Letzte Zeile lösen lassen bei korrekt gefülltem Sudoku	Die erste/ letzte Zeile wird gelöst	Ok
Erste/ Letzte Zeile lösen lassen bei fehlerhaften Sudoku	Es erscheint eine entsprechende Fehlermeldung	Ok

C. Quellenangaben

[1] Japanese samurai in armour, 1860s. Photograph by Felice Beato. Public Domain
[2] http://www.janko.at/Raetsel/Sudoku/Regeln.htm
[3] http://www.angusj.com/sudoku/hints.php