Projekte

Einführung in die Programmierung mit *Go*

11. März 2015

Zum Bestehen der Veranstaltung müssen Sie eines der auf den folgenden Seiten beschriebenen Projekte umsetzen. Senden Sie dazu bis zum Donnerstag, 18:00, eine Mail an den Veranstalter mit Namen und Matrikelnummern der beiden Teilnehmer Ihrer Gruppe und einer Liste der bevorzugten Projektnummern in absteigender Reihenfolge Ihrer Präferenz.

Mailadresse: `reichenb@em.uni-frankfurt.de`

Falls keines der von Ihnen gewünschten Projekte verfügbar ist, wird Ihnen eines der verbleibenden Projekte zufällig zugeteilt.

Anforderungen an das Projekt sind:

- Sie müssen die geforderte Lösung vollständig implementieren und den Quellcode vor dem 27. März beim Veranstalter einsenden oder auf einem externen Revisionskontrollsystem (Github, Bitbucket, Sourceforge, ...) deponieren.
- Der Quellcode muß vollständig und selbstständig ausführbar sein. Sie dürfen externe Bibliotheken für Komponenten verwenden, sofern die Aufgabenstellung nicht explizit um eine 'eigene Implementierung' bittet.
- Verwenden Sie Bibliotheken der Standardbibliothek (`http://golang.org/pkg/`) sofern möglich, um Zeit und Aufwand zu sparen.
- Ihr Quellcode muß der Go-Modulstruktur genügen (wie im 2. Arbeitsblatt betrachtet).
- Die Module müssen Unit Tests beinhalten. Mindestens zwei Unit-Test-Dateien für jeweils unterschiedliche Operationen müssen vorliegen, mit sichtbarem Bemühen, die Korrektheit der Implementierungen zu prüfen. Wenn dies nicht möglich oder angemessen ist, muß eine schriftliche Begründung mit eingesendet werden.
- Am 27. stellen Sie Ihre Lösung in einem Vortrag von 15 Minuten vor. Wenn nicht anders angegeben, können Sie die Struktur des Vortrages beliebig wählen; eine Demonstration ist ebenso möglich wie ein Vortrag mit Folien oder eine Besprechung des Quellcodes.

Ihre Teilnahme an der Veranstaltung ist erfolgreich, wenn Sie die obigen Anforderungen erfüllen. Wenn Ihre Implementierung unvollständig ist, kann der Veranstalter sie dennoch akzeptieren, sofern (a) ernsthaftes Bemühen und eine Fertigstellung des Großteils des Projektes ersichtlich sind, und (b) verbleibende Probleme und Einschränkungen explizit in Ihrem Vortrag benannt werden.

Nützliche Ressourcen:

- Speichern und Laden von Daten per json: `http://blog.golang.org/json-and-go`
- Um Text von der Eingabe einzulesen, können Sie (nach Importieren der betreffenden Module `"os"` und `"bufio"`) folgenden Code verwenden:

```
reader := bufio.NewReader(os.Stdin)
text, _ := reader.ReadString('\n')
```

1 Go-Refactoring

Ein *Refactoring* ist eine Transformation eines Programmes, die dessen Verhalten nicht ändert.

Ein typisches Refactoring ist die Umbenennung einer Variablen.

1. Implementieren Sie ein Programm, das als Parameter eine `go`-Datei und drei Parameter nimmt:

   - Programmzeile
   - Programmspalte
   - Name

   Das Programm soll den Bezeichner (`go/ast.Ident`) identifizieren, der an der betreffenden Programmzeile und Programmspalte im Programm liegt und in den neuen Namen umbenennen.

2. Sie dürfen vereinfachend annehmen, daß der angegebene Bezeichner Teil eines `ValueSpec`-Knotens ist.

3. Die Umbenennung soll *konsistent* durchgeführt werden, das heißt, sie soll andere Vorkommnisse des gleichen Bezeichners in der gleichen Datei ebenfalls umbenennen.

Folgende Probleme müssen Sie dabei korrekt behandeln:

```
fun f() int {
  var x int = 2
  return x + 1
}
var x int // darf nicht verändert werden
```

Wenn die Variable `x` aus der Zeile `var x int = 2` umbenannt werden soll, müssen Sie folgendes sicherstellen:

- Die globale Variable mit gleichem Namen darf nicht verändert werden
- Das zweite Vorkommen der Variablen in der Zeile `return x + 1` muß ebenfalls umbenannt werden.

Ihr Refactoring muß eigenständig implementiert sein. Sie können aber das Programm `gofmt` als Basis verwenden:

- Quellcode: `https://golang.org/src/cmd/gofmt/gofmt.go`
- Weiterer relevanter Quellcode für das Werkzeug ist im Verzeichnis `https://golang.org/src/cmd/gofmt/`
- Beachten Sie insbesondere die Funktion `simplify` in `https://golang.org/src/cmd/gofmt/simplify.go`, die den abstrakten Syntaxbaum (`ast`) bearbeitet und mit der Funktion `ast.Walk` über diesen Baum wandert.
- Weitere Informationen finden Sie in der Dokumentation der Pakete `"go"` `"go/ast"` usw.

2 Term-Umschreibung lösen

Schreiben Sie ein Go-Programm, das Term-Umschreibungsregeln der Form

$$A(x, y) \longrightarrow B(C(x))$$

umsetzen kann.

- Ein Term t ist entweder:
  - Eine Variable x, y, $\ldots$, oder
  - Ein Konstruktor-Term $C(t_1, \ldots, t_n)$ für beliebige C und beliebig vielen Subtermen t_i. Insbesondere ist auch $n = 0$ möglich.
- Ein *geschlossener Term* ist ein Term, der keine Variablen beinhaltet.
- Zwei geschlossene Terme sind gleich, wenn sie syntaktisch übereinstimmen.
- Eine Umschreiberegel ist ein Paar $t_l \to t_r$, so daß alle Variablen, die in t_r vorkommen, auch in t_l vorkommen.
- Eine *Substitution* σ ist eine Abbildung von Variablen auf geschlossene Terme. Mit $\sigma(t)$ beschreiben wir das rekursive Ersetzen von Variablen in t durch diese geschlossenen Terme.
- Ein Term t wird durch eine Regel $t_l \to t_r$ zu einem Term t' umgeschrieben gdw ein σ existiert, so daß $t = \sigma(t_l)$, und $t' = \sigma(t_r)$.
- Ihr System nimmt eine Menge von Umschreiberegeln und einen geschlossenen Eingabeterm t als Eingabe. Es schreibt t gemäß den Umschreiberegeln so lange um, bis keine weitere Umschreibung mehr möglich ist.

Sie müssen die Terme nicht aus einer Datei einlesen; es ist ausreichend, wenn diese in einem *Go*-Programm als *Go*-Werte kodiert werden können.

Testen Sie unter Anderem mit folgende Umschreiberegelmenge:

$$\begin{array}{lcl}
A(S(x), y) & \longrightarrow & S(A(x, y)) \\
A(Z(), y) & \longrightarrow & y \\
T(x) & \longrightarrow & S(S(S(x))) \\
M(Z(), y) & \longrightarrow & Z() \\
M(S(Z()), y) & \longrightarrow & y \\
M(S(S(x))), y) & \longrightarrow & A(y, M(S(x), y)) \\
E(x, x) & \longrightarrow & TRUE()
\end{array}$$

Überprüfen Sie, daß das Ergebnis von $E(A(Z(), S(Z())), A(S(Z()), Z()))$ und $M(T(S(Z)), T(T(Z)))$ wie erwartet ist.

3 Arithmetische Bibliothek

Schreiben Sie eine Bibliothek, die arithmetische Terme mit den folgenden Konstrukten unterstützt:

- Addition, Subtraktion, Multiplikation, Division, Potenzierung
- Sinus und Kosinus
- Natürlicher Logarithmus und Potenzierung mit der Eulerschen Zahl e
- Fließkommakonstanten
- Variablen

Ihre Bibliothek soll folgende Operationen über die Terme erlauben:

- Menschenlesbare Darstellung als String
- Auswertung, falls eine Map von Variablennamen auf Variableninhalte angegeben wird
- Ableitung und Integrierung
- Typische Vereinfachungen (Multiplikation mit 1 oder 0, Addition von 0, Addition mehrerer Konstanten)

Demonstrieren Sie dies mit geeigneten Beispielen.

4 Multi-User Dungeon

Entwickeln Sie mit Hilfe von Goroutinen und der Go-Netzwerkbibliothek ein 'Multi-User-Dungeon' (MUD). Mitspieler können zusammen oder alleine durch diese virtuelle, rein durch Text beschriebenen Welt wandern.

Ein solches System ist Client-Server basiert; Sie können aber das existierende Programm `telnet` als Client verwenden, so daß sich Ihre Implementierung auf den Server beschränken kann.

- Jeder Mitspieler muß sich mit Paßwort anmelden. Sie können die Mitspieler fest einkodieren oder die Anmeldung neuer Mitspieler erlauben.
- Mitspieler interagieren mit dem Spiel, indem Sie Zeichenketten an den Server schicken. Sie können die Funktion `strings.Fields` verwenden, um diese Zeichenketten auseinanderzunehmen.
- Die für die Mitspieler verfügbaren Befehle sollten mindestens Befehle sein, um folgendes zu tun:
  1. Nach Norden, Süden, Osten oder Westen zu gehen
  2. Einen explizit benannten Gegenstand zu nehmen oder abzulegen
  3. Einen explizit benannten Gegenstand an einen Mitspieler zu geben
  4. Einen explizit benannten Gegenstand zu benutzen
  5. Alle vom Spieler selbst gehaltenen Gegenstände aufzuführen
  6. Zu sprechen
- Alle Aktionen, die für einen Mitspieler im gleichen Raum sichtbar sein sollten (Spechen; Aufnehmen, Ablegen, Übergeben von Gegenständen; Betreten oder Verlassen des Raumes) werden diesen mitgeteilt.
- Das MUD besteht aus mehreren 'Räumen', die miteinander verknüpft sind. Jeder Raum hat einen Namen und eine Beschreibung, die den Spielern angezeigt werden, wenn der Raum betreten wird.
- Bauen Sie eine kleine Welt mit verschiedenen miteinander verbundenen Räumen und Gegenständen.

5 Lokaler Spiegel-Proxy

Bauen Sie einen Webserver, der HTTP-Anfragen an einen anderen Webserver weiterleitet. Ihr Server soll die besuchten Webseiten lokal spiegeln:

- Der Webserver kann im Online-Modus oder im Offline-Modus gestartet werden. Als Parameter erhält er den Namen des zu spiegelnden externen Webservers (z.B. `de.wikipedia.org`).
- Im Online-Modus werden Anfragen an einen bestimmten Pfad an den gleichen Pfad des externen Webservers weitergeleitet. Die zurückgelieferten Informationen werden zum Einen an den anfragenden Webbrowser weitergeleitet, zum Anderen aber auch lokal im Dateisystem gespeichert.
- Im Offline-Modus werden die Anfragen aus den lokal gespeicherten Dateien bedient.
- Wenn im Online-Modus auf eine bereits vorher geladene URL zugegriffen wird, soll diese Anfrage ebenfalls aus den zwischengespeicherten Informationen bedient werden.
- Es soll über einen Kommandozeilenparameter möglich sein, daß der Online-Modus HTML-Dateien, die vom externen Server geladen werden, automatisch auf URLs prüft und diese URLs in einer separaten Goroutine im Hintergrund vorläd. Zur Vereinfachung nehmen wir an, daß alles, was in der Datei auftaucht und die Form `href="`s`"` hat, eine gültige URL ist (wobei s ein beliebiger String ist, der nicht mit # beginnt). Die Funktion `strings.Index` kann Ihnen bei der Suche nach solchen Zeichenketten helfen.

6 Datenstrukturen und Performanz

Implementieren Sie die folgenden beiden Datentypen in drei Programmiersprachen— Go, Python, und einer dritten Sprache Ihrer Wahl (z.B. C oder Java). Messen Sie, wie effizient die Implementierungen sind.

- Implementieren Sie für die drei Sprachen die *doppelt verkettete Liste*. Ihre Datenstruktur sollte Listenknoten als explizite Objekte erlauben, und mindestens folgende Operationen bieten:
  - Neuen Knoten vor anderen Knoten einfügen
  - Neuen Knoten hinter anderen Knoten einfügen
  - Knoten vor anderem Knoten entfernen
  - Knoten hinter anderem Knoten entfernen
  - Vorgängerknoten finden
  - Nachfolgerknoten finden
  - Knoten auslesen (`int`-Zahl)
- Implementieren Sie für die drei Sprachen den *Splay-Baum*[1] (Spreizbaum), mit mindestens folgenden Operationen:
  - Neuen, leeren Baum erzeugen
  - Element (`int`) einfügen
  - Prüfen, ob ein bestimmter Wert als Element enthalten ist, und Anwendung der Splay-Operation, wenn der Wert gefunden wurde.
- Stellen Sie sicher, daß die Implementierungen in den verschiedenen Sprachen sich in ihrer Struktur möglichst ähnlich sind.
- Entwerfen Sie mindestens zwei Performanztests für jeden der Datentypen, und verwenden Sie die gleichen Operationsaufrufe zum Testen in allen Implementierungen.
- Führen Sie 13 Zeitmessungen für jeden Performanztest durch. Verwerfen Sie die ersten drei Messungen, um Aufwärmeffekte zu minimieren. Berechnen Sie Mittelwert, Median und Standardabweichung der Messungen; alternativ können Sie einen Kasten- oder Violinengraphen erstellen. Wenn die Standardabweichung größer als 20% der Messung ist, überprüfen Sie Ihr Meßverfahren, um sicherzustellen, daß Ihre Messungen nicht durch starke Zufallseffekte beeinflußt wird.
- Messen Sie auch die Länge des geschriebenen Programmcodes.

Für diese Aufgabe müssen Sie eine Präsentation vorbereiten, um Ihre Messungen in Form von Graphen zu illustrieren.

[1] `http://de.wikipedia.org/wiki/Splay-Baum`

7 A*-Suche

Implementieren Sie den A*-Algorithmus[2] in Go und einer zweiten Sprache Ihrer Wahl (z.B. Python, C, oder Java). Messen Sie, wie effizient die Implementierungen relativ zu einander sind.

- Bauen Sie eine kleine Bibliothek zur Repräsentierung von Graphen. Ihre Bibliothek soll Graphknoten und Kanten repräsentieren.
- Jeder Graphknoten hat eine Position (gegeben durch x, y-Koordinaten).
- Jede Kante zwischen zwei Knoten hat eine Länge. Diese Länge darf nicht kürzer als die Luftlinie zwischen den beiden Knoten sein.
- Erzeugen Sie zufällig drei Testgraphen. Verwenden Sie diese Graphen in beiden Programmiersprachen. Sie können diesen Schritt vereinfachen, wenn Sie die erzeugten Testgraphen in der Form von Funktions- oder Methodenaufrufen ausgeben, die zur Konstruktion des Graphen führen.
- Implementieren Sie A*-Suche über Graphen in Ihrer Graph-Bibliothek.
- Messen Sie die Performanz in Go und der anderen von Ihnen gewählten Programmiersprache.

Beachten Sie die Anmerkungen zur Meß-Methodologie aus der vorherigen Aufgabe.
Für diese Aufgabe müssen Sie eine Präsentation vorbereiten, um Ihre Messungen zu zeigen.

[2] `http://de.wikipedia.org/wiki/A*-Algorithmus`

8 Zeitmanagement im Browser

Implementieren Sie ein Programm zur Verwaltung von möglicherweise terminierten TODO-Listen in einem Webserver.

- Bauen Sie einen Webserver, der bei ohne weitere Angaben bei einem Zugriff mehrere Dinge anzeigt:
  - Eine Liste aller TODO-Einträge, die für den heutigen Tag oder in der Vergangenheit terminiert sind
  - Eine Liste aller TODO-Einträge, die auf 'Warten' gestellt sind
  - Eine Liste aller nicht terminierten TODO-Einträge
  - Eine Liste mit weiteren Optionen
- Jeder TODO-Eintrag mit Ausnahme der Warteliste soll mit einem Link versehen sein. Ein Zugriff darauf markiert den Eintrag als 'erledigt' und legt ihn in die Ablage.
- Neben jedem TODO-Eintrag (außer Einträgen auf der 'Warten'-Liste') soll zusätzlich ein Link sein, um diesen Eintrag auf die 'Warte'-Liste zu stellen. Damit ist gemeint, daß zur Weiterbearbeitung auf Rückantwort gewartet werden muß.
- Die URL für Einträge auf der 'Warten'-Liste macht die betreffenden TODO-Einträge wieder aktiv und stellt sie an die Spitze der TODO-Liste.
- Die weiteren Optionen sollen folgende Operationen erlauben:
  - Neuer TODO-Eintrag: Hier kann ein neuer Eintrag eingeführt werden. Optional kann auch ein Erledigungsdatum angegeben werden (in einer dem Programm genehmen Form).
  - Periodischer TODO-Eintrag: Analog zu 'Neuer TODO-Eintrag'; der Eintrag wiederholt sich wöchentlich an einem bestimmten Tag oder täglich.
  - Ablage betrachten: Dies führt alle erledigten TODO-Einträge auf. Diese können wiederbelebt werden.
- Die TODO-Listeneinträge sollen in einer Datei gesichert werden. Bei Start des Servers werden sie geladen und nach jeder Änderung auf der Festplatte aktualisiert.

9 3D-Fraktale

Berechnen Sie die Mandelbrot-Menge und illustrieren Sie diese dreidimensional. Verwenden Sie dazu folgenden vereinfachten Voxel-Ansatz:

- Zeichnen Sie für jeden berechneten Bildpunkt nicht nur den Punkt selbst, sondern einen dünnen (einen Pixel breiten) Balken von der Position des Pixels nach oben. Die Höhe des Balkens soll gleich der gemessenen Iterationszahl k sein. Wir bezeichnen die ursprüngliche Position des Pixels nun als die 'Startposition' des Balkens.

- Verwenden Sie nun die Entfernung des Bildpunktes, um die Entfernung des Balkens zum Betrachter zu simulieren. Dazu nehmen Sie die y-Koordinate der Startposition, um diese Entfernung zu simulieren– desto größer y (desto näher am unteren Rand des Bildes), desto näher ist der Balken am Betrachter. Die Entfernung d sollte also linear zur y-Koordinate der Startposition des Balkens sein.

  Wählen Sie die tatsächliche y-Koordinate der Startposition des Balkens nun anders: setzen Sie sie als $\frac{c}{d}$, wobei c eine Konstante ist. Experimentieren Sie mit geeigneten Werten. Dividieren Sie zudem die Höhe des Balkens durch d. Sie müssen ggf. die Höhe des Balkens noch mit einer Konstanten multiplizieren, damit der Balken sichtbar bleibt.

- Experimentieren Sie mit möglichen Farbzuweisungen und weiteren Verfeinerungen in der Bilddarstellung.

- Ihr Programm nimmt x- und y-Position sowie Vergrößerungsfaktor und *Blickwinkel* des Betrachters als Kommandozeilenparameter.

- Das Ergebnis wird in eine `png`-Datei geschrieben.

- Identifizieren Sie interessante Bilder und stellen Sie diese als Teil Ihres Vortrages vor.

10 Webserver zur Dokumentenablage und Suche

Implementieren Sie einen Webserver, der zur Dokumentenablage verwendet werden kann. Ihr Webserver sollte die folgenden Operationen zur Verfügung stellen:

- Hochladen von einzelnen Dateien
- Hochladen von `.zip`- und `.tar.gz`-Dateien, die automatisch entpackt werden
- Herunterladen von Dateien, wobei zuvor hochgeladene `.zip` bzw. `.tar.gz`-Dateien sowohl verpackt als auch unverpackt (in einzelnen Dateien) herunterladbar sein sollen.
- Sichern der Dateien auf der Festplatte, falls der Server gestoppt und neu gestartet werden muß.
- Löschen von Dateien.
- Textsuche nach Suchbegriffen in den Dateien.

11 Backup-Manager: Dateien Sichern

Implementieren Sie einen Backup-Manager und ein Wiederherstellungsprogramm. Ihr Backup-Programm soll Dateien von einer Stelle im Dateisystem an eine andere Stelle kopieren.

- Konfiguration: Erlauben sie dem Benutzer die Angabe von
  - Zu sichernden Verzeichnissen
  - Verzeichnissen innerhalb der zu sichernden Verzeichnisse, die NICHT gesichert werden sollen
  - Dateiendungen, die nicht gesichert werden sollen
- Bei der ersten Verwendung kopieren Sie rekursiv in allen Verzeichnnissen alle Dateien von der Quellposition zur Zielposition, sofern die Dateien den obigen Anforderungen des Benutzers entsprechen.
- Merken Sie sich in einer Protokolldatei alle gesicherten Dateien zusammen mit der Dateigröße und der CRC32-Checksumme des Inahlts der Datei.
- Bei späteren Verwendungen prüfen Sie die aktuellen Inhalte des Dateisystems gegen die Protokolldatei. Kopieren Sie NUR Dateien, die neu sind oder deren Größe oder CRC32-Checksumme sich geändert hat. Wenn eine Datei gelöscht wurde oder die Größe 0 hat, vermerken Sie dies in der Protokolldatei, löschen das Backup aber nicht.
- Erlauben Sie die Wiederherstellung von verlorengegangenen Dateien. Ihr Programm restauriert alle gelöschten oder auf Größe 0 gesetzten Dateien vom Backup-Pfad.

Beachten Sie, daß dies keine vollständige Backup-Lösung ist, da Sie immer nur eine Version des Backups haben. Sie können dieses Projekt mit einer dritten Person bearbeiten, um die Lösung zu vervollständigen. In diesem Fall sind folgende zusätzliche Aktivitäten nötig:

- Sichern der Protokolldatei selbst.
- Speichern von Datum/Uhrzeit des Programmstarts des Backup-Programmes.
- Jede gesicherte Datei kann mehrfach gespeichert sein; wenn die Datei sich ändert, wird die vorherige Version also nicht überschrieben.
- Bei der Wiederherstellung von Dateien kann Datum/Uhrzeit angegeben werden, um ältere Versionen wiederherzustellen.

12 Kryptographisches Chat-System mit Dateienaustausch

Go verfügt über eine Bibliothek von Verschlüsslungsverfahren. Verwenden Sie diese, um kryptographisch gesicherte Kommunikation zu ermöglichen.

- Ermöglichen Sie die Chat-Kommunikation zwischen beliebig vielen Teilnehmern, wie in den folgenden Schritten beschrieben.
- Jegliche Kommunikation findet verschlüsselt statt, gemäß der kryptographischen API, die Go zur Verfügung stellt[3].
- Jeder Kommunikationsteilnehmer hat ein Paar von privaten und einen öffentlichen Schlüsseln. Der öffentliche Schlüssel wird weitergegeben und von anderen Kommunkationsteilnehmern zum Verschlüsseln verwendet. Der private Schlüssel wird nicht weitergegeben; er dient zum Entschlüsseln.
- Jegliche Kommunikation zwischen Teilnehmern an dem verschlüsselten Chat ist mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Die einzige Ausnahme ist das Versenden von öffentlichen Schlüsseln.
- Entwerfen Sie ein einfaches Kommunikationsprotokoll mit den obigen Rahmenbedingungen.
- Schreiben Sie ein Chatprogramm (Client und Server).
- Der Server dient dazu, Datein zwischen den Clients hin und her zu leiten, und den Clients mitzuteilen, welche Chat-Teilnehmer angemeldet sind.
- Erlauben Sie das Versenden bzw. Empfangen von Dateien über Ihr Kommunikationsprotokoll.

[3] http://golang.org/pkg/crypto/

13 Sudoku-Löser

Dieses Programm liest ein Sudoku[4]-Problem aus einer Textdatei ein, z.B. in der Form

```
. . .  . . .  . 7 5
. . .  . . .  . . .
3 . .  . 2 .  . . .

. 7 5  . . .  . . .
. . 4  . 6 .  1 . .
. . .  . . 1  2 . .

. . .  5 4 6  . . .
8 9 .  . . .  . . .
. . .  . 7 .  . . .
```

wobei Punkte freie Zellen sind, und druckt eine Lösung aus.

- Implementieren Sie ein Leseprogramm für die gegebenen Sudoku-Testdateien.
- Implementieren Sie einen 'brute force'-Lösungsalgorithmus, der alle Möglichkeiten rekursiv durchprobiert.
- Verfeinern Sie Ihren Algorithmus: bestimmen Sie zunächst, wieviele mögliche Zahlen es für jedes Feld in der aktuellen Auswahl gibt, die die im aktuellen Spielfeld sichtbaren Beschränkungen nicht verletzen. Wählen Sie zufällig eines der Felder mit den *wenigsten* möglichen Zahlen aus. Iterieren Sie durch alle Möglichkeiten für dieses Feld und arbeiten Sie rekursiv weiter.
- Testen Sie die beiden Verfahren anhand von existierenden Sudoku-Rätseln. Welches Verfahren ist schneller?

[4] `http://de.wikipedia.org/wiki/Sudoku`

14 Zeichenprogramm per Kommandozeile

Schreiben Sie ein Programm, das `png`-Dateien zeichnet. Das Programm soll einen Dateinamen als Parameter akzeptieren, in der in jeder Zeile einer der folgenden Befehle steht:

- '`farbe f`' setzt die aktuelle Zeichenfarbe. Sie können sich hier für ein beliebiges nichttriviales Format zum Ausdrücken von Farben *f* entscheiden.
- '`punkt x y`' zeichnet einen einzelnen Punkt mit der gegebenen Farbe an der gegebenen Koordinate.
- '`linie x1 y1 - x2 y2`' zeichnet eine Linie zwischen den gegebenen Koordinaten. Dabei soll es möglich sein, die Linie noch weiter zu verlängern, also noch `- x3 y3` usw. anzugeben.
- '`polygon x1 y1 - x2 y2 ...`': Analog zu `linie`; der einzige Unterschied ist, daß noch eine abschließende Linie vom letzten bis zum ersten Punkt gezogen wird.
- '`kreis x1 y1 r`': Zeichnet einen Kreis um den Punkt $\langle x1, y1 \rangle$ mit Radius *r*.
- '`fuelle x1 y1`' füllt das Bild mit der gegebenen Farbe; dazu können Sie den *flood fill*-Algorithmus [5] verwenden.

Implementieren Sie zum Zeichnen von Linien und Kreisen den Bresenham-Algorithmus [6]. Nutzen Sie dabei die Möglichkeiten zur Spiegelung, so daß Sie den betreffenden Algorithmus nur für einen Oktante implementierne müssen.

[5] http://de.wikipedia.org/wiki/Floodfill

[6] `http://de.wikipedia.org/wiki/Bresenham-Algorithmus`

15 Blackjack

Implementieren Sie das Spiel Blackjack[7] mit einem menschlichen Spieler gegen den Computer.

- Das Spiel folgt den üblichen Regeln. Es wird offen gespielt; jeder sieht also alle Karten.
- Betrachten Sie die 52 Karten als konkrete Werte und den Kartenstapel als Slice von Karten.
- Der Kartenstapel wird zu Beginn gemischt. In nachfolgenden Spielen wird immer vom gleichen Stapel gezogen.
- Wenn der Kartenstapel leer ist, aber eine neue Karte benötigt wird, werden alle Karten aus bereits abgeschlossenen Spielen neu zusammengemischt.
- Zeigen Sie in jeder Runde die Karten des Computers, die Karten des Menschen, und die Höhe des Kartenstapels an.
- Der Computer hat nur die Optionen 'Karte' oder 'Keine Karte'.
- Der Mensch hat die Option 'Karte', 'Keine Karte', oder 'Verdoppeln' (Einsatz verdoppeln, noch genau eine Karte ziehen).
- Nach jedem Spiel wird angezeigt, wie oft der menschliche Spieler gewonnen oder verloren hat. Gewinne mit 'Verdoppeln' zählen dabei doppelt.
- Der Computerspieler entscheidet, ob er eine weitere Karte zieht, indem er eine Simulation durchführt:
  - Der Stapel der noch nicht gezogenen Karten wird kopiert, die Kopie erneut durchmischt (um zu vermeiden, daß der Computerspieler 'in die Zukunft sehen' kann) und die nächste Karte gezogen, dann geprüft, ob der gezogene Gesamtwert 21 nicht überschreitet, der Zug also 'sicher' war.
  - Die gleiche Simulation wird für die Karten des menschlichen Spielers durchgeführt.
  - Beide Simulationen werden mehrfach durchgeführt (z.B. 100 mal) und die Ergebnisse protokolliert.
  - Der Computerspieler entscheidet anhand der Ergebnisse, ob er eine weitere Karte zieht oder nicht. Entwerfen Sie dazu ein Entscheidungsverfahren.

Beachten Sie, daß der Computer um so besser wird, desto kleiner der Kartenstapel ist, da er die 'Karten zählt' und sich merkt, welche Karten nicht mehr auftreten können.

[7] `http://de.wikipedia.org/wiki/Black_Jack`