Projekt zur GdI 1 - WS 2008/09

Steven Arzt, Oren Avni, Johannes Reichard, Guido Rößling

24. Januar 2009

Version 1.0

Inhaltsverzeichnis

1 Einführung in das Projekt

Im Rahmen des Projekts implementieren die Studierenden in Gruppen von jeweils vier Personen eine Java-Version des alten Spieleklassikers „Sokoban“. Die Aufgabenstellung stellt zunächst das zugrundeliegende Spiel Sokoban vor.

Die Aufgabe kann in vier verschiedenen „Ausbaustufen“ bearbeitet werden, die jeweils eine unterschiedliche Punktzahl zur Gesamtnote beitragen. Die minimale Ausbaustufe muss zum Erreichen der Mindestpunktzahl **vollständig** implementiert werden.

Ab Ausbaustufe I können nicht erreichte Punkte der gegebenen Ausbaustufe durch Elemente höherer Ausbaustufen „ausgeglichen“ werden. Projektabgaben, die „zwischen“ Ausbaustufen liegen, bei denen also erwartete Inhalte fehlen oder zusätzliche Elemente eingebaut wurden, sind natürlich ebenfalls möglich; die Ausbaustufen geben nur eine grobe Orientierung vor. Beachten Sie dabei jedoch, dass die Ausbaustufen nach Schwierigkeitsgrad gruppiert sind, d.h. Aufgaben höherer Stufen sind in der Regel schwieriger zu lösen als Aufgabenteile niedrigerer Ausbaustufen.

1.1 Das Spiel „Sokoban“

Im Spiel Sokoban steuert der Spieler einen Lagerarbeiter—das ist auch die Bedeutung des japanischen Titels. Dieser Lagerarbeiter bewegt sich in einem Lager, dessen Außengrenzen mit Wänden abgetrennt sind. Im Lager befinden sich mehrere Kisten (crates), die die Waren enthalten. Damit das Lager schön ordentlich ist, müssen alle Kisten an die dafür speziell gekennzeichneten Abstellpositionen geschoben werden.

Da die Kisten recht schwer sind, kann der Lagerarbeiter immer nur eine Kiste auf einmal schieben. Zusätzlich kann er eine gegebene Kiste nur dann schieben, wenn das Feld vor der Kiste frei ist, sich dort also weder eine andere Kiste noch eine Wand befindet. Ein Ziehen von Kisten ist grundsätzlich nicht möglich.

Als Spieler steuern Sie den Lagerarbeiter und müssen das Lager aufräumen. Welche Kiste auf welche Abstellposition geschoben wird, ist dabei nicht relevant. Ein Sokoban Spiel besteht aus mehreren Leveln. Die Level werden von uns als Textdateien bereitgestellt, die in einem bestimmten Format gehalten sind, das in Abschnitt 4 auf Seite 12 definiert wird.

Abbildung 4.1 auf Seite 13 zeigt eine Beispielausgabe. Wir erwarten nicht, dass das im Rahmen des Praktikums erstellte Spiel der Ausgabe optisch ähnlich sieht; die Abbildung soll nur zum Verstehen des Spiels beitragen. Weitere Informationen zu dem Spiel finden Sie z.B. unter `http://de.wikipedia.org/wiki/Sokoban` sowie über Google.

2 Organisation des Projekts

Der offizielle Bearbeitungszeitraum des Projekts beginnt *Montag, den 23. 2. 2009* und endet *Dienstag, den 10. 3. 2009*. Zur Teilnahme am Projekt müssen *alle* Gruppenmitglieder die folgenden Bedingungen erfüllen:

- Die Studienleistung zur GdI 1 bzw. zur ICS 1 wurde im Wintersemester 2008/2009 oder bereits in früheren Semestern erworben. Die einzige Ausnahme von dieser Regelung sind

Studierende der Fächer *Wirtschaftsinformatik* oder *Lehramt an Gymnasien (PO 2005)*, die keine Studienleistung als Voraussetzung für die Teilnahme benötigen.

- Die Teilnahme an der Prüfung wurde fristgerecht im Zentralen Prüfungssekretariat angemeldet und nicht wieder zurückgezogen.
- Jeder Teilnehmer der Gruppe hat sich vor dem Projekt im Webreg-System für das Projekt angemeldet und wurde der Gruppe zugeteilt. Im Webreg-System ist auch die Anmeldung direkt als Gruppe möglich und wird auch angeraten, damit die Gruppen nicht zufällig aufgefüllt werden müssen.

Die Aufgabenstellung wird direkt nach Ablauf der Abmeldefrist von der Prüfung, d.h. am Montag, den **26.01.2009**, im Portal veröffentlicht. Ab diesem Zeitpunkt ist prinzipiell bereits die Bearbeitung der Aufgabe möglich. Da zu diesem Zeitpunkt die Gruppen im Webreg noch nicht endgültig gebildet werden konnten, raten wir in diesem Fall aber *dringend* zu einer Anmeldung direkt als Gruppe von vier Studierenden in Webreg.

Im Projekt bearbeitet die Gruppe die in den folgenden Abschnitten näher definierte Aufgabe. Dabei wird die Gruppe von den Veranstaltern wie folgt unterstützt:

- Das Portal und insbesondere das *Forum zum Projekt-Organisation* sowie das *Forum zu Projekt-Inhalten* kann für alle gruppenübergreifenden Fragen zum Verständnis der Aufgabe oder Unklarheiten bei der Nutzung der Vorlagen genutzt werden.
- Jede Projektgruppe kann im Portal eine *Projektgruppe anlegen*, indem sie den passenden Link nutzt (dieser wird erst zu Beginn des Bearbeitungszeitraums freigeschaltet). In dieser Projektgruppe können Fragen diskutiert oder Code-Fragmente ausgetauscht werden (Tipp: umgeben Sie Code im Portal immer mit `<code type="java">...</code>`, damit er besser lesbar ist). Wenn der Tutor der Gruppe ebenfalls zur Projektgruppe eingeladen wird, wird er besser in die Diskussionen eingebunden und kann leichter und schneller Feedback geben.

  Bitte beachten Sie, dass diese Projektgruppen im Portal von uns nur ein *Angebot* an Sie sind, das Sie nicht nutzen müssen. Wenn Sie beispielsweise alle Aufgaben direkt in der WG eines Studierenden erledigen, bringt eine im Portal erstellte Projektgruppe vermutlich mehr Aufwand als Nutzen.
- Eine Gruppe von Tutoren betreut das Projekt. Jedem Tutor wird dabei eine Anzahl Projektgruppen zugeteilt. Die Aufgabe des Tutors ist es, die Gruppe im Rahmen des Projekts bei offenen Fragen zu unterstützen, nicht aber bei der tatsächlichen Implementierung. Insbesondere hilft der Tutor nicht bei der Fehlersuche und gibt auch keine Lösungsvorschläge. Der Tutor steht der Gruppe auch nur zeitlich begrenzt zur Verfügung: pro Gruppe wurden bis zu 3 Stunden Betreuung sowie eine halbe Stunde für die Testierung angesetzt.
- Für den einfacheren Einstieg stellen wir einige Materialien bereit, die Sie im Portal herunterladen können. Diese Materialien inklusive vorgefertigten Klassen *können* Sie nutzen, müssen es aber nicht. Zu den Materialien zählen auch vorbereitete Testfälle. *Diese müssen unverändert auf Ihrer Implementierung funktionieren, da sie—zusammen mit „privaten" Tutorentests—als Basis für die Abnahme dienen.*

Die *Abnahme* oder *Testierung* des Projekts (beschrieben in Abschnitt 3) erfolgt durch den Tutor und erfordert eine *vorherige Terminabsprache*. Bitte bedenken Sie, dass auch unsere Tutoren Termine haben (etwa die Abnahme der anderen von ihnen betreuten Gruppen) und nicht „pauschal immer können“. In Ihrem eigenen Interesse sollten Sie daher versuchen, so früh wie möglich einen Termin für die Besprechungen und die Abnahme zu vereinbaren. Sie sollten auch einen Termin für die erste Besprechung mit dem Tutor vereinbaren. Vor diesem Termin sollten Sie schon dieses Dokument komplett durchgearbeitet haben, sich alle offenen Fragen notiert haben (und im Portal nach Antworten gesucht haben), *und* einen Entwurf vorbereitet haben, wie die Lösung Ihrer Gruppe aussehen soll. Dieser Entwurf kann in UML erfolgen, aber prinzipiell ist jede (für den Tutor) lesbare Form denkbar. Bitte bringen Sie diesen Entwurf zum ersten Treffen mit dem Tutor mit, damit er oder sie direkt Feedback geben kann, ob dieser Ansatz funktionieren kann. Auch hier kann die Nutzung des Portals mit den Projektgruppen helfen, den Tutor „früher zu erreichen“.

3 Ihre Aufgabe

Implementieren Sie eine lauffähige Java-Version von Sokoban, die mindestens der „minimalen Ausbaustufe“ entspricht.

Das fertige Spiel muss *als ZIP oder JAR-Datei mit allen Quelltexten, Bildern etc. sowie einer kurzen Dokumentation (etwa 2 DIN A4-Seiten) im Portal hochgeladen werden* und vom Tutor *vor Ende des Praktikums testiert werden*. Das Testat besteht aus drei Bestandteilen.

Live-Test Der Tutor startet das Spiel und testet, ob alles so funktioniert wie spezifiziert. Dazu wird er bestimmte vorgegebene Szenarien durchspielen, aber auch zufällig „herumklicken“.

Software-Test Der Tutor testet die Implementierung mit den für alle Teilnehmer bereitgestellten (öffentlichen) und nur für die Tutoren und Mitarbeiter verfügbaren (privaten) JUnit - Tests. Alle Tests müssen **ohne Benutzerinteraktion** abgeschlossen werden können.

Code-Review Der Tutor sieht sich den Quellcode und Ihre Dokumentation an und stellt Fragen dazu.

3.1 Ablauf des Code-Review

Im Hinblick auf das Code-Review sollten Sie auf gut verständlichen und dokumentierten Code sowie eine sinnvolle Klassenhierarchie achten, in der Regel auch mit Aufteilung der Klassen in Packages.

Der Tutor wird einzelne Gruppenmitglieder seiner Wahl zu Teilen des Quelltextes befragen. Daher sollte sich jedes Gruppenmitglied mit allen Codeteilen auskennen - der Tutor wählt aus, zu *welchem Thema* er fragt und er wählt auch aus, *wer* die Frage beantworten soll. Die Bewertung dieses Teils bezieht sich also auf die Aussage eines „zufällig ausgewählten“ Gruppenmitglieds, geht aber in die Gesamtpunktzahl der Gruppe ein.

Damit soll einerseits die „Trittbrettfahrerei“ reduziert werden („ich habe zwar nichts getan, will aber dennoch die Punkte haben“). Gleichzeitig fördert diese Regelung die Gruppenarbeit, da auch und gerade besonders „starke“ Mitglieder verstärkt Rücksicht auf „schwächere“ nehmen müssen -

sonst riskieren sie eine schlechtere Punktzahl, wenn „der Falsche“ gefragt wird. Durch eine entsprechend bessere Abstimmung in der Gruppe steigen die Lernmöglichkeiten **aller** Gruppenteilnehmer. Auch für (vermeintliche?) „Experten“ wird durch das Nachdenken über die Frage „wie erkläre ich das verständlich?“ das eigene Verständnis vertieft.

3.2 Dokumentation

Neben dem Quelltexten ist auch eine kurze Dokumentation abzugeben (etwa 2 DIN A4-Seiten). Diese sollte die *Klassenstruktur* ihrer Lösung in UML umfassen und kurz auf die in ihrer Gruppe *aufgetretenen Probleme* eingehen sowie *Feedback zur Aufgabenstellung* liefern. Nur die Klassenstruktur geht in die Bewertung ein; die anderen Elemente helfen uns aber dabei, das Praktikum in der Zukunft besser zu gestalten und sind daher für uns sehr wichtig. Sie können den Teil mit der (hoffentlich konstruktiven) Kritik am Projekt auch gerne separat auf Papier—auf Wunsch ohne Angabe des Gruppennamens—dem Tutor geben, wenn Sie das Feedback lieber anonym geben wollen.

Für die Erstellung der Klassendiagramme können Sie beispielsweise folgende Tools nutzen:

- *doxygen* (`http://www.stack.nl/~dimitri/doxygen/`) eine Alternative zu JavaDoc, die—bei Wahl der entsprechenden Optionen–auch Klassendiagramme erzeugt. Eine Dokumentation zu *doxygen* finden Sie auf der Projekt-Homepage.
- *Fujaba* (`http://wwwcs.uni-paderborn.de/cs/fujaba/`)
- *BlueJ* (`http://bluej.org/`)

Dies sind nur unsere Empfehlungen. Es steht Ihnen selbstverändlich frei, andere Tools zu nutzen, etwa OpenOffice Draw, Microsoft Word etc.

3.3 Hinweise

Sie sollten in Ihrem Code strikt zwischen Logik (Code) und Darstellung (Design) unterscheiden, wie dies auch in der Praxis gängig ist. Ihre Tutoren werden bewerten, wie gut diese Trennung bei Ihnen gelungen ist.

Denken Sie bitte daran, dass **Testfälle keine Interaktion mit einem Spieler erfordern dürfen**.

3.4 Minimale Ausbaustufe

Um das Praktikum bestehen zu können, also mindestens 50 aus 100 Punkten zu erhalten, müssen **alle** nachfolgend genannten Leistungen erbracht werden:

Pünktliche Abnahme - 0 Punkte Der Quelltext mit Dokumentation wird rechtzeitig in das Portal hochgeladen (als JAR oder ZIP mit allen für Übersetzung und Ausführung erforderlichen *.java* Dateien, Bildern etc.) und wird vom Tutor nach vorheriger Terminabsprache rechtzeitig vor dem Ablaufen der Abnahmefrist testiert. **Sie** sind für die Einhaltung der Termine verantwortlich.

Compilierbares Java - 0 Punkte Der Quelltext ist komplett in Java implementiert und kann separat ohne Fehlermeldung neu compiliert werden.

Nur eigener Code - 0 Punkte Der Quelltext enthält **keine** oder **nur genehmigte** fremde Codeteile, insbesondere keinen Code von anderen Praktikumsgruppen oder aus dem Internet. Die Nutzung von fremdem Code kann nur durch die Mitarbeiter (nicht die Tutoren!), individuell oder bei allgemeiner Nachfrage im Portal pauschal für konkrete Codeteile, genehmigt werden.

In Ihrem eigenen Interesse müssen Sie in der kurzen Ausarbeitung sowie beim Testat **explizit und unaufgefordert** auf fremde Codeteile (mit Angabe der Quelle und des Umfangs der Nutzung) hinweisen, um sich nicht dem Vorwurf des Plagiarismus oder gar des Betrugsversuchs auszusetzen. Wir behalten uns vor, Lösungen (auch automatisiert) miteinander zu vergleichen. Bitte beachten Sie, dass ab einem gewissen Umfang der Nutzung von fremden Codes wir Ihre Abgabe nur noch als gescheitert betrachten können, da der Eigenanteil zu gering wird.

Vorbereitung für automatisierte Tests - 5 Punkte Die öffentlichen Tests verwenden die Methoden des Interface `SokobanTest`. Um die öffentlichen Tests ablaufen zu lassen, müssen Sie die Adapter-Klasse `SokobanTestAdapter` implementieren. Diese Klasse soll *keine* Sokoban Funktionalität enthalten, sondern die einzelnen Methoden lediglich auf die entsprechenden Methoden in Ihrer Implementierung abbilden.

Nutzen Sie auch die Möglichkeit, benötigte Objekte in der Methode `init` anzulegen und „passend“ zu initialisieren.

Beispiel: Haben Sie sich entschieden, die Anzahl der Kisten in einem Level in dem statischen Feld `count` der Klasse `Crate` zu zählen, so wäre dies eine passende Implementierung der Methode `getCrateCount()` aus `SokobanTest`:

```
public int getCrateCount() {
  return Crate.count;
}
```

Haben Sie sich für einen anderen Entwurf entschieden, sähe die Implementierung anders aus.

Ihr Programm muss auch *ohne diese Klasse voll funktionsfähig sein!* In Eclipse können Sie das testen, indem Sie die Klasse vom Build ausschließen (Rechtsklick auf `SokobanTestAdapter`, Build path, exclude).

Öffentliche Tests sind erfolgreich - 0 Punkte Die öffentlichen Testfälle der Klasse `SokobanTestMinimal` werden fehlerfrei absolviert. Da Sie diese bereits während des Praktikums selbst testen können, sollte diese Anforderung für Sie kein Problem darstellen. Beachten Sie, dass Sie für den Ablauf der Tests die Klasse `SokobanTestAdapter` (s. o.) implementieren müssen! Generell sind die öffentlichen Tests eine Hilfe für Sie. Die Testfälle durch „Schummeln“ erfolgreich zu bestehen, bringt Sie nicht weiter.

Parsen von Leveln - 5 Punkte Das von der Gruppe erstellte Spiel kann dem Dateiformat in Abschnitt 4 auf Seite 12 entsprechende Level korrekt parsen. Das bedeutet, dass Level fehlerfrei aus einer Datei eingelesen und in die interne Darstellung umgewandelt werden können. Die Art der internen Datenrepräsentation (Datenstruktur) bleibt Ihnen überlassen.

Tipp: Erinnern Sie sich bitte an die Java-Hausübungen, insbesondere diejenigen die Spiele behandelt haben :-)

Check auf syntaktische Korrektheit - 5 Punkte Beim Einlesen eines Levels wird überprüft, ob der Level syntaktisch korrekt ist. Ein Level ist genau dann syntaktisch korrekt, wenn er nur gültige Zeichen gemäß der *Formatbeschreibung* in Abschnitt 4 auf Seite 12 enthält und die folgenden Bedingungen erfüllt sind:

- Es befindet sich genau ein Arbeiter auf dem Spielfeld.
- Es befindet sich mindestens eine Kiste auf dem Spielfeld.
- Es befinden sich genauso viele Kisten wie Zielpositionen (Goals) auf dem Spielfeld.

Es soll *keine* Lösbarkeitsbetrachtung durchgeführt werden.

Ist ein Level nicht syntaktisch korrekt, soll eine `Exception` ausgelöst werden. Erstellen Sie dazu eine eigene Exception-Klasse, die aber nicht von `RuntimeException` abgeleitet sein soll.

Korrekte *toString()*-Methode - 5 Punkte Jeder eingeladene Level kann als String ausgegeben werden. Der ausgegebene String soll dabei dem Dateiformat in Abschnitt 4 auf Seite 12 entsprechen. Es soll der aktuelle Spielzustand wiedergegeben werden, nicht der Originalzustand.

Erkennen gelöster Level - 5 Punkte Das Spiel erkennt von selbst, wenn ein Level fertig gespielt wurde. Das ist der Fall, wenn sich alle Kisten auf einer Abstellpositionen befinden. Sobald dies passiert, soll eine entsprechende Meldung auf der Konsole ausgegeben werden.

Grafische Benutzerschnittstelle - 5 Punkte Es gibt eine grafische Benutzerschnittstelle ("GUI"), mit der man Sokoban spielen kann.

Grafische Umsetzung der Objekte - 5 Punkte Für jedes Objekt im Spiel (Spieler, Kiste, Wand, Zielfeld, Freiraum, Spieler auf Zielfeld, Kiste auf Zielfeld) existiert eine passende grafische Umsetzung.

Korrekte Leveldarstellung - 5 Punkte Alle eingeladenen Leveldateien müssen korrekt dargestellt werden. Betrachten Sie bitte dazu Abbildung 4.1 auf Seite 13 als Beispiel für eine grafische Umsetzung.

Tastensteuerung - 5 Punkte Die Spielfigur kann über die Cursor-Tasten (Up, Down, Left, Right) gesteuert werden. Jeder Tastendruck entspricht einem Schritt in die entsprechende Richtung. Falls ein Schritt gemäß den Spielregeln nicht möglich ist, wird der Befehl ignoriert und damit auch *nicht* als Schritt gezählt. Der Spieler soll auf einen ungültigen Zug adäquat hingewiesen werden, etwa durch eine entsprechende Textausgabe auf der Konsole.

Hinweis: Der Code in den Testklassen **darf auf keinen Fall** zur Anzeige von Dialogfenstern führen—auch nicht bei inkorrekten Schritten—, da diese eine Eingabe erfordern, die in unseren JUnit-Tests nicht automatisiert erfolgen kann.

Die betreffende Aktion soll nach den Spielregeln auf dem Spielfeld ausgeführt werden, d.h. der Arbeiter wird bewegt und etwaige Kisten werden verschoben.

Neustart und Beenden - 5 Punkte Der Spieler soll über die Taste **n** den aktuellen Level neu starten können. Mit der Taste **q** soll das Programm beendet werden können.

3.5 Ausbaustufe I

Die Ausbaustufe I erweitert die minimale Ausbaustufe. **Alle Anforderungen der minimalen Ausbaustufe gelten weiterhin, sofern sie unten nicht explizit geändert oder erweitert werden.** Die erste Ausbaustufe erlaubt es mehrere Level hintereinander zu spielen, außerdem soll eine Highscore Liste geführt werden.

Bei Implementierung dieser Ausbaustufe können insgesamt 70 aus 100 möglichen Punkten erreicht werden. Die öffentlichen Testfälle für diese Ausbaustufe befinden sich in der Klasse `SokobanTestExtended1`.

Bewegung über eine Methode - 0 Punkte Der Arbeiter soll mit der Methode *public void moveWorker(char direction)* bewegt werden können, wobei die Richtung **direction** einen der folgenden Werte annehmen kann:

- 'R' für eine Bewegung nach rechts (*right*)
- 'L' für eine Bewegung nach links (*left*)
- 'U' für eine Bewegung nach oben (*up*)
- 'D' für eine Bewegung nach unten (*down*)

Züge, die gegen die Spielregeln verstoßen, sollen nicht ausgeführt werden. Diese Methode ist für die Ausbaustufe I **obligatorisch**.

Öffentliche Tests der Ausbaustufe 1 sind erfolgreich - 0 Punkte Die öffentlichen Testfälle der Klasse `SokobanTestExtended1` werden fehlerfrei absolviert.

Zählen gültiger Schritte - 2 Punkte Jeder *gültige* Schritt des Spielers innerhalb eines Levels soll gezählt werden. Am Ende eines Levels ist die Gesamtschrittanzahl anzuzeigen, beispielsweise auf der Konsole.

Starten des nächsten Levels - 2 Punkte Hat der Spieler einen Level gelöst, wird automatisch der nächsthöhere Level gestartet. Dazu müssen Sie mehrere Level aus einem Verzeichnis lesen können.

„Highscore-Liste“ - 8 Punkte Es soll eine „Highscore-Liste“ gespeichert und aktualisiert (und daher auch eingelesen) werden. Das Dateiformat, das für die „Highscore-Liste“ verwendet werden soll, wird in Abschnitt 4 auf Seite 12 beschrieben. In der „Highscore-Liste“ werden zu jedem Level die *besten zehn* Lösungen gespeichert. Zu jeder Lösung wird die Nummer des Levels, der Name des Spielers und die Zahl der benötigten Schritte gespeichert.

„Highscore-Liste“ GUI - 5 Punkte Die „Highscore-Liste“ soll in der GUI angezeigt werden. Dabei sollen die Spieler zuerst absteigend nach erreichtem Level und dann aufsteigend nach der für das höchste Level benötigte Schrittzahl aufgelistet werden. Die Liste soll sich über die Benutzerschnittstelle auch zurücksetzen lassen.

Sie müssen zu einem geeigneten Zeitpunkt den Namen des Spielers abfragen, um ihn in die Highscore-Liste eintragen zu können! Beachten Sie, dass alle öffentlichen Testfälle ohne Benutzerinteraktion durchlaufen müssen.

Vereinfachte Maussteuerung - 3 Punkte Erweitern Sie das Spiel um eine einfache Maussteuerung. Wird ein Spielfeld oder eine Kiste direkt neben der Spielfigur angeklickt (also das Feld ober- oder unterhalb sowie links oder rechts von der Spielfigur), dann soll die Spielfigur dorthin bewegt werden, als wäre das entsprechende Kommando über die Tastatur eingegeben worden. Wenn Sie das von uns zur Verfügung gestellte Framework verwenden, können Sie hierzu die Methode `entityClicked` verwenden (siehe Abschnitt 5.2 auf Seite 15).

3.6 Ausbaustufe II

Die Ausbaustufe II erweitert Ausbaustufe I und die minimale Ausbaustufe. **Alle Anforderungen dieser Ausbaustufen gelten weiterhin, sofern sie unten nicht explizit geändert oder erweitert werden.**

Bei Implementierung dieser Ausbaustufe können insgesamt 90 aus 100 möglichen Punkten erreicht werden. Die öffentlichen Testfälle für diese Ausbaustufe befinden sich in der Klasse `SokobanTestExtended2`.

Öffentliche Tests der Ausbaustufe 2 sind erfolgreich - 0 Punkte Die öffentlichen Testfälle der Klasse `SokobanTestExtended2` werden fehlerfrei absolviert.

Undo und Redo - 8 Punkte Der Benutzer soll eine beliebige Anzahl an Spielzügen rückgängig machen bzw. wiederherstellen können. Um diese Funktionalität wirklich nützlich zu machen, bauen Sie bitte Tastatursshortcuts ein: Backspace-Taste für Undo, Enter-Taste für Redo.

Achten Sie darauf, dass Ihre Anwendung auch nach beliebig vielen Bewegungen des Arbeiters auf dem Spielfeld funktionsfähig bleibt.

Ist kein Undo bzw. Redo möglich (z.B. direkt nach dem Spielstart vor der ersten Bewegung des Arbeiters), soll eine Exception ausgelöst werden. Erstellen Sie eine eigene Exception-Klasse, die aber nicht von `RuntimeException` abgeleitet sein darf.

Spielstand sichern - 3 Punkte Der aktuelle Spielstand, also der eventuell vorhandene *Spielername* (für die Highscore Liste), der Name des *aktuellen Levels* und die *bisher durchgeführten Schritte* („Schrittsequenz“) sollen sich speichern lassen. Überlegen Sie sich hierzu ein Dateiformat. Die einzelnen Spielstände sollen mit der Dateiendung „.sok“ im Level-Verzeichnis gespeichert werden. Die Schrittsequenz soll als String codiert werden, wobei jedes Zeichen des Strings einen Schritt angibt (Format {'R', 'L', 'U', 'D'}, wie oben definiert).

Spielstand laden - 3 Punkte Spielstände die wie oben gesichert wurden, sollen sich wieder laden lassen. Dabei wird der gespeicherte Zustand wieder hergestellt.

Beachten Sie, dass unmittelbar nach dem Laden des Spielstandes kein undo / redo möglich sein *muss*.

Skin wechseln - 2 Punkte Erweitern Sie Ihr Spiel um zusätzliche Skins, sowie eine Möglichkeit, aus der Beutzerschnittstelle heraus den aktuellen Skin zu wechseln. Dies muss inmitten eines Spiels möglich ein, ohne dass dieses unterbrochen oder gar das Programm neu gestartet werden muss.

Ein Skin ist ein kompletter Satz der verwendeten Bilder. Sie dürfen eine feste Menge von Skins vorgeben, die dem Benutzer zu Verfügung stehen (mindestens drei).

Highscore-Liste mit Zeitangabe - 3 Punkte Erweitern Sie die Highscore-Liste um eine Anzeige der zur Lösung des jeweiligen Levels benötigten Zeit. Die Zeit soll jedoch nicht in die Berechnung des Rankings eingehen. Beachten Sie, dass damit die benötigte Zeit auch zum Spielstand gehört und beim Sichern und Laden von Spielständen berücksichtigt werden muss.

Tipp: sehen Sie in die Dokumentation der Methode *currentTimeMillis()* der Klasse *java.lang.System*.

„About"-Fenster - 1 Punkt Implementieren Sie eine About-Box, d.h. ein Fenster, das die Namen der beteiligten Mitglieder Ihrer Praktikumsgruppe auflistet. Sie kennen diese Fenster sicher aus vielen bekannten Windowsprogrammen. Seien Sie kreativ :-)

3.7 Ausbaustufe III

Die Ausbaustufe III erweitert Ausbaustufe II, I und die minimale Ausbaustufe. **Alle Anforderungen dieser Ausbaustufen gelten weiterhin, sofern sie unten nicht explizit geändert oder erweitert werden.**

Bei Implementierung dieser Ausbaustufe können insgesamt 100 aus 100 möglichen Punkten erreicht werden. Die öffentlichen Testfälle für diese Ausbaustufe befinden sich in der Klasse `SokobanTestExtended3`.

Öffentliche Tests der Ausbaustufe 3 sind erfolgreich - 0 Punkte Die öffentlichen Testfälle der Klasse `SokobanTestExtended3` werden fehlerfrei absolviert.

Cheat-Code - 2 Punkte Der Benutzer soll während des normalen Spiels über die Tastatur den Befehl `osjx,y#` eingeben können, wobei *x* und *y* für eine beliebige x- bzw. y-Koordinate auf dem Spielfeld stehen. Sobald das letzte Zeichen des Cheats (#) eingelesen wurde, soll die Spielfigur an die angegebene Position gesetzt werden, sofern diese noch nicht belegt ist und innerhalb des Spielfelds liegt.

Deadlock-Erkennung - 3 Punkte Nach jeder Bewegung einer Kiste durch den Spieler soll überprüft werden, ob sich das Spiel in einem Deadlock-Zustand befindet. Ein Deadlock-Zustand bedeutet, dass vom aktuellen Zustand aus keine Lösung des Spiels mehr möglich ist. Dies allgemein zu lösen ist recht kompliziert, daher beschränken wir uns in diesem Praktikum nur auf die folgenden Situationen, die erkannt werden müssen:

- mindestens eine Kiste wurde in eine Ecke geschoben, d.h. die Kiste grenzt nun an drei Wänden an.
- vier Objekte bilden ein Quadrat (z.B. vier Kisten oder drei Kisten und eine Wand) an einer beliebigen Stelle auf dem Spielfeld.

Beispiele für eine Deadlock-Situation werden in Abbildung 1 auf der nächsten Seite visualisiert. Beachten Sie, dass dort auch Situationen gezeigt sind, die zwar Deadlocks sind, aber nicht zwingend von erkannt werden müssen. Das Spiel befindet sich natürlich nicht in einem Deadlock, wenn die Kisten bereits auf Zielfeldern stehen.

Ist eine Deadlock Situation eingetreten, soll der Spieler darauf aufmerksam gemacht werden, etwa durch eine Ausgabe auf der Konsole.

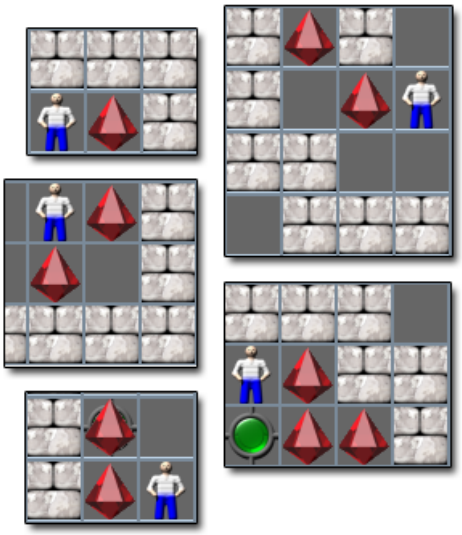

Abbildung 1: Beispiele für das Erkennen von Deadlock Situationen

Erweiterte Prüfung auf Level-Korrektheit - 5 Punkte Ein korrekter Sokoban-Level besitzt eine vollständige Umrandung durch Außenmauern. Dies bedeutet, dass der Arbeiter das Spielfeld an keiner Stelle durch regelgerechte Schritte verlassen kann. Implementieren Sie daher eine Methode *public boolean checkWallBounding()*, die überprüft, ob der Level korrekt an allen Seiten von Wänden umgeben ist, so dass sich die Spielfigur und alle weiteren Objekte (Kisten und Zielpositionen) innerhalb dieser Mauern befinden.

3.8 Bonuspunkte

Sie können auch mehr als 100 Punkte erreichen sowie fehlende Punkte aus anderen Ausbaustufen ausgleichen, indem Sie weitere Funktionen implementieren, die in den Ausbaustufen nicht spezifiziert wurden. Die Bewertung ist dabei Sache der Tutoren und der Veranstalter. Zur Orientierung stellen wir hier beispielhaft mögliche Erweiterungen mit Punktzahl vor, damit Sie wissen, was wir ungefähr erwarten.

Brute-Force-Solver - 6 Punkte Implementieren sie eine Methode *public String solve()*, die automatisch eine Lösung für den aktuellen Level bestimmt. Der Rückgabewert sind dabei die zurückgelegten Schritte im oben spezifizierten Format {'R', 'L', 'U', 'D'}.

Die Strategie ist dabei wie folgt: Führen Sie von der aktuellen Position alle möglichen Bewegungen aus und prüfen Sie, ob das Spiel dadurch gelöst wurde. Wenn nicht, setzen Sie den Algorithmus rekursiv für alle neuen Positionen aus. Wenn Sie eine Deadlock-Situation erkannt haben, brauchen Sie diesen Lösungsweg nicht mehr zu betrachten.

Der Benutzer soll eine maximale Suchzeit angeben können, nach der die Suche automatisch ergebnislos abgebrochen wird. Ihre Funktion erhält im Test maximal 25 Sekunden Zeit, um das erste Beispiellevel zu lösen.

Testen Sie den Algorithmus mit verschiedenen Levels. Welches Problem können Sie erkennen? (Wird mündlich beim Testat abgefragt, sofern die Aufgabe bearbeitet wurde).

Erweiterte Maussteuerung - 7 Punkte Erweitern Sie die Maussteuerung, so dass der Benutzer auf ein beliebiges erlaubtes Feld klicken kann und die Spielfigur dann mit einer geringstmöglichen Anzahl an Zügen dorthin bewegt wird. Die Bewegung soll sichtbar erfolgen, also ohne „Teleportationen".

Zur Vereinfachung sollen Kisten hierbei als Wände betrachtet werden. Der Arbeiter soll also um eine im Weg stehende Kiste mit dem kürzesten möglichen Weg herumlaufen, ohne Kisten zu verschieben. Ist die Bewegung nicht möglich, soll die Spielfigur am Startplatz verbleiben.

Nutzung von Audio-Clips - 2 Punkte Audio-Clips, die bei bestimmten Ereignissen abgespielt werden (z.B. gegen die Wand laufen oder eine Kisten auf ein Ziel stellen). Passende Soundclips werden von uns als MP3-Dateien auf der GdI1 Webseite bereitgestellt. Sie sollen die Soundausgabe zum korrekten Zeitpunkt implementieren. Nutzen Sie hierfür das von uns bereitgestellte Framework `javazoom.jar` und schauen Sie sich bitte dazu die Dokumentation (JavaDoc) an.

Überraschen Sie uns - 99 Punkte Sie können auch eigene Ideen zum Ausbau des Spiels einbringen. Die Bewertung mit Punkten ist dann Sache des Tutors und der Veranstalter.

4 Dateiformate

4.1 Leveldateien

Sokoban verwendet ein einfaches Textformat zur Definition von Spielleveln. Jeder Level steht in einer eigenen Datei, deren Name das Format `level_ij.txt` hat und für Level 1-9 eine führende 0 nutzt, also etwa `level_02.txt`.

Die Datei ist zeilenweise aufgebaut. Jede Zeile der Datei entspricht einer Zeile im Level. Alle Zeilen werden mit einem Zeilenvorschub „\n" abgeschlossen. Das Textformat kennt die in Tabelle 1 aufgeführten Zeichen, wobei jedes Zeichen genau einem Element der Ausgabe in der GUI entspricht.

Zeichen	Bedeutung
	(Leerzeichen): freies Feld
#	Wand
$	Kiste
.	Freies Zielfeld
*	Kiste (auf einem Zielfeld)
@	Spielfigur (auf einem freien Feld)
+	Spielfigur (auf einem Zielfeld)

Tabelle 1: Elemente des Sokoban-Formats

Abbildung 4.1 zeigt ein Beispiel für eine denkbare grafische Umsetzung eines gültigen Levels (konkret: Level 1 der von uns bereitgestellten Level) in ein grafisches System. Die Zielposition an

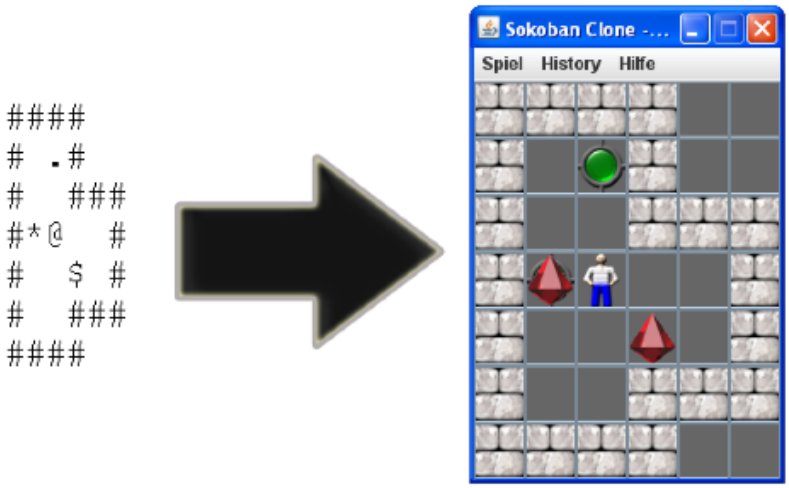

Abbildung 2: Beispielhafte Umsetzung der Formatdefinition in eine GUI

Position (1, 3) bei Zählung ab 0 besteht aus einer Kiste auf einer Zielposition. Die Zielposition wird hierbei allerdings durch die Kiste optisch fast vollständig verdeckt.

4.2 Highscoredateien

Die Highscoreliste soll in einer Datei namens *highscore.txt* gespeichert werden. Diese enthält einen Datensatz pro Zeile, die Reihenfolge der Zeilen in der Datei ist jedoch beliebig.

Eine Datensatz enthält mindestens drei Felder, die durch Tabstopps getrennt sind. Optional können weitere Felder (ebenfalls durch Tabstopps getrennt) folgen. Die ersten vier Datenfelder sind dabei fest vorgegeben (siehe Tabelle 2). Ihre Implementierung muss beliebige Dateien einlesen können, von Ihnen nicht verwendete Datenfelder müssen jedoch beim erneuten Abspeichern nicht erhalten bleiben.

Spalte	**Bedeutung**
1	Eindeutige Levelnummer
2	Spielername
3	Zum Lösen des Levels benötigte Anzahl Schritte
4	Zum Lösen des Levels benötigte Zeit in Sekunden

Tabelle 2: Datenfelder der Highscoredatei

Der folgende Datensatz beschreibt damit Spieler Peter, der zum Lösen von Level 1 33 Schritte und 55 Sekunden benötigt hat.

1 Peter 33 55 *optionaleWerte*

Falls Sie die Zeitangabe für die von Ihnen angestrebte Ausbaustufe noch nicht benötigen, können Sie diese wie weitere ignorierte Zusatzdaten behandeln.

5 Materialien

Auf der Veranstaltungsseite stellen wir Ihnen einige Dateien zur Verfügung, die Sie für Ihre Lösung verwenden können. Dabei handelt es sich um die ersten 25 Originallevel des Sokoban Spiels (aus dem Jahre 1982) als auch weitere (leichtere) 25 Levels zum "durchspielen" sowie die öffentlichen Testcases.

Wenn Sie möchten, können Sie unser bereitgestelltes Framework nutzen. Sie können aber auch vollständig eigene Lösungen implementieren. Vermeiden Sie es aber bitte, „das Rad neu zu erfinden“.

Hinweis: Ihre Lösung muss in jedem Fall mit den bereitgestellten Testklassen überprüfbar sein.
Hinweis: Die in den Tabellen *kursiv gesetzten* Methoden sollten Sie bei korrekter Implementierung niemals selbst aufrufen. Diese werden lediglich vom Framework verwendet, um Ihren Code über bestimmte Ereignisse zu informieren.

Wenn Sie bestimmte abstrakte Ereignismethoden des Frameworks nicht benötigen, lassen Sie deren Implementierung einfach leer. Für eine korrekte Lösung müssen nicht zwangsläufig alle Ereignisse behandelt werden.

Das Framework ist, soweit nicht anders angegeben, nicht threadsicher.

5.1 Klasse *gdi1sokoban.template.ui.GameWindow*

Diese Klasse bietet Ihnen ein Grundgerüst für ein Spielfenster. Um Ihre eigene Implementierung zu beginnen, leiten Sie eine Klasse hiervon ab. Grundsätzlich sind Sie bei der Ausgestaltung Ihrer Oberfläche frei, die Basisklasse soll lediglich eine Hilfe darstellen.

Überschreiben Sie die abstrakten Methoden, um beispielsweise auf Ereignisse wie Tastendrücke zu reagieren, die unter anderem den Spieler bewegen oder auch als bestimmte Kommandos fungieren sollen.

Neben der kurzen Beschreibung in diesem Dokument gibt Ihnen die JavaDoc-Dokumentation nützliche Hinweise zur Verwendung und den Parametern der vorhandenen Funktionen. Ein Blick in diese Quellen empfiehlt sich sehr—**am besten vor dem Gang zum Tutor**, um. . .

- sich selbst und auch dem Tutor wertvolle Zeit zu sparen, da sich so unter Umständen banale Fragen von selbst lösen.
- Nerven zu sparen, da das Warten auf die Antwort des Tutors Ihre Nerven beanspruchen wird :-)
- sich selbstständiges Arbeiten anzueignen.

5.2 Klasse *gdi1sokoban.template.ui.GamePanel*

Verwenden Sie diese Basisklasse (d.h. erben Sie von ihr), um das eigentliche Spielfeld darzustellen. Indem Sie die abstrakten Methoden überschreiben, können Sie die Felddarstellung steuern.

Das Framework wurde für Bilder gleicher Größe konzipiert. Sind Ihre Wände also z.B. 20 x 20 Pixel groß, sollten es Ihre Kisten auch sein, um unschöne Ränder bei der Darstellung zu vermeiden.

Die Bildgröße wird automatisch ermittelt. Das Spielfeld passt seine Größe ebenfalls automatisch daran an. Um das Feld passend in Ihrem Fenster zu platzieren, können Sie die aktuelle Größe über die Funktionen *getWidth()* und *getHeight()* abfragen.

Hinweis zur Threadsicherheit: Die Funktion *redraw()* ist threadsicher, sofern Ihre Implementierung von *setGamePanelContents()* dies ist.

5.3 Exceptions

Das Framework verwendet eine Reihe von Exceptionklassen, die Sie natürlich auch im Rahmen Ihrer eigenen Implementierung verwenden können. Hierbei handelt es sich um:

- *InternalFailureException*
  Diese Exception wird geworfen, wenn ein zur Laufzeit nicht korrigierbarer interner Fehler im Framework aufgetreten ist. Sollten Sie diese Exception jemals erhalten, wenden Sie sich bitte umgehend an einen Tutor. Die Exception speichert entweder eine Fehlermeldung oder eine innere Exception, die die ursprüngliche interne Fehlerursache angibt.
- *InvalidOperationException*
  Der aktuelle Zustand des Objekts lässt die gewünschte Operation nicht zu. Dies kann zum Beispiel auftreten, wenn Sie im GamePanel auf eine nicht registrierte Bild-ID zugreifen. Stellen Sie in einem solchen Fall sicher, dass die aufgerufene Operation im aktuellen Kontext sinnvoll ist und überprüfen Sie in die übergebenen Parameter. Die Exception speichert eine Fehlermeldung.
- *ParameterOutOfRangeException*
  Diese Exception wird geworfen, wenn mindestens einer der übergebenen Parameter außerhalb des zulässigen Wertebereichs liegt. Die Exception speichert den Namen des betreffenden Parameters, den Sie beim Auftreten der Exception überprüfen sollten.

5.4 Sounds mit JavaZoom

Zur Einbettung von Sounds stellen wir Ihnen im Verzeichnis *lib* die Bibliothek *javazoom* zur Verfügung, mit der Sie Audiodateien in verschiedenen Dateiformaten, darunter auch MP3, abspielen können. Sehen Sie sich hierzu insbesondere die Klasse *Sound* an. Der Konstruktor erhählt den Namen der einzulesenden Audiodatei. Mit der Methode *play()* können Sie den Abspielvorgang starten.

5.5 Die ersten Schritte

Ihre von *GameWindow* abgeleitete Klasse ist das Hauptfenster Ihres Spiels. Wenn diese Klasse *StudentWindow* heißt, können Sie folgenden Code zum Starten des GUI verwenden:

```
StudentWindow wnd = new StudentWindow();
wnd.setVisible(true);
```

Diesen Code sollten Sie in der *main* Methode Ihrer Hauptklasse platzieren, vorzugsweise also in *StudentWindow*.

Als Nächstes sollten Sie die Methode *createGamePanel* überschreiben und darin Ihr von *GamePanel* abgeleitetes Spielfeld setzen, das im Beispiel den Namen *StudentPanel* trägt:

```
StudentPanel panel = new StudentPanel(this);
add(panel);
return panel;
```

Das Beispiel fügt das Panel direkt in den Container des Fensters ein. Die konkrete Vorgehensweise hängt natürlich vom Layout Ihres Fensters sowie den weiteren darauf platzierten Komponenten ab.

Da die Klasse *GameWindow* (und somit auch alle von ihr abgeleiteten Klassen) von *JFrame* erbt, stehen Ihnen darin auch alle Funktionen eines JFrame zur Verfügung. Die Klasse *GamePanel* erbt von *JPanel*.

6 Zeitplanung

Wir empfehlen Ihnen, **vor Beginn der Bearbeitung** zuerst eine für Ihre Gruppe realistische Ausbaustufe auszuwählen und die Aufgabe dann in parallel bearbeitbare Teilaufgaben zu zerlegen, die Sie auf die einzelnen Gruppenmitglieder verteilen. Über zentrale Elemente wie die grundsätzlichen Klassennamen und die Vererbungshierarchie sollten Sie sich in der Gruppe einigen, um spätere Diskussionen zu vermeiden.

Erstellen Sie einen schriftlichen Zeitplan, wann Sie welche Aufgabe abgeschlossen haben möchten. Dabei ist es wichtig, den aktuellen Projektstand regelmäßig kritisch zu überprüfen. Ein solches geplantes Vorgehen vermeidet Stress (und damit unnötige Fehler) beim Abgabetermin.

Eine Zeitplanung für die minimale Ausbaustufe könnte etwa wie in Tabelle 6 auf Seite 21 gezeigt aussehen. Eine denkbare Gesamteinteilung für alle Ausbaustufen wird in Tabelle 6 auf Seite 21 gezeigt.

Bitte beachten Sie, dass **alle Zeiten stark von Ihrem Vorwissen und Ihren Fähigkeiten abhängen**, so dass die Zeiten nicht als „verbindlich“ betrachtet werden dürfen!

Tipp: Wenn Sie eine Ausbaustufe fertig implementiert haben, sollten Sie einen Gang zum Tutor nicht scheuen, ehe Sie sich gleich an die nächste Stufe heranwagen. Fragen Sie ihn oder sie nach eventuellen Unklarheiten, falls Sie z.B inhaltlich die Aufgabenstellung nicht verstanden haben. Es ist Aufgabe der Tutoren, Fragen zu beantworten, also nutzen Sie dieses Angebot.

Wichtig: „Sichern“ Sie stabile Zwischenergebnisse, etwa indem Sie ein Versionskontrollverfahren wie *CVS* oder *SVN* nutzen. Tipps dazu finden Sie im Portal. Die einfache Variante ist es, regelmäßig eine JAR-Datei mit den Quellen (!) anzulegen, die Sie jeweils je nach erreichtem Status „passend“ benennen. Dann haben Sie immer eine Rückfallmöglichkeit, falls etwa nach einem Code-Umbau nichts mehr funktioniert.

7 Errata

In diesem Kapitel werden wir etwaige nachträgliche Änderungen sammeln. Damit können Sie bei neuen Versionen der Aufgabenstellung direkt sehen, was sich geändert hat.

Derzeit sind keine Fehler bekannt.

8 Liste der Anforderungen

Anforderungen an das Projekt Seite

Prozedur	**Beschreibung**	**Was ist zu tun?**
createGamePanel()	Diese Methode liefert dem GameWindow die zu nutzende GamePanel-Instanz.	Überschreiben Sie diese Methode und fügen Sie folgende Aktionen ein: Erstellen Sie eine Instanz Ihrer von GamePanel abgeleiteten Klasse, platzieren Sie diese nach Wunsch auf Ihrem Fenster und geben Sie sie zurück.
keyLeftPressed()	Diese Methode wird aufgerufen, wenn die linke Pfeiltaste gedrückt wurde.	Überschreiben Sie diese Methode, um den Arbeiter nach dem Druck der Pfeiltaste zu bewegen.
keyRightPressed()	Diese Methode wird aufgerufen, wenn die rechte Pfeiltaste gedrückt wurde.	Überschreiben Sie diese Methode, um den Arbeiter nach dem Druck der Pfeiltaste zu bewegen.
keyUpPressed()	Diese Methode wird aufgerufen, wenn die Pfeiltaste nach oben gedrückt wurde.	Überschreiben Sie diese Methode, um den Arbeiter nach dem Druck der Pfeiltaste zu bewegen.
keyDownPressed()	Diese Methode wird aufgerufen, wenn die Pfeiltaste nach unten gedrückt wurde.	Überschreiben Sie diese Methode, um den Arbeiter nach dem Druck der Pfeiltaste zu bewegen.
keyNewGamePressed()	Diese Methode wird aufgerufen, wenn die Taste *n* gedrückt wurde.	Überschreiben Sie diese Methode, um den Level auf den Ursprungszustand zurückzusetzen.
keyUndoPressed()	Diese Methode wird aufgerufen, wenn die Rücklauftaste (Backspace) gedrückt wurde.	Überschreiben Sie diese Methode, um den letzten Spielzug rückgängig zu machen.
keyRedoPressed()	Diese Methode wird aufgerufen, wenn die Entertaste gedrückt wurde.	Überschreiben Sie diese Methode, um den letzten Spielzug wiederherzustellen.
keyOtherPressed()	Diese Methode wird aufgerufen, wenn eine nicht anderweitig behandelte Taste gedrückt wurde.	Überschreiben Sie diese Methode, um auf gesonderte Tastendrücke zu reagieren, falls Sie das Framework um andere Tastaturbefehle erweitern möchten.
notifyLevelLoaded(int, int)	Teilt dem Fenster mit, dass ein neuer Level geladen wurde.	Rufen Sie diese Methode nach dem Laden eines Levels auf. Die Parameter repräsentieren die Breite und Höhe des neuen Levels.

Tabelle 3: Ereignisbehandlungsmethoden in der Klasse *gdi1sokoban.template.ui.GameWindow*

Prozedur	Beschreibung	Was ist zu tun?
setGamePanelContents()	Diese Methode wird aufgerufen, um die Feldinhalte zu platzieren.	Überschreiben Sie diese Methode, um Ihre Feldinhalte (Arbeiter, Wände, Kisten) zu erzeugen.
entityClicked(int, int)	Diese Methode wird aufgerufen, wenn ein Element auf dem Spielfeld angeklickt wurde.	Überschreiben Sie diese Methode, um auf Klicks zu reagieren. Die Parameter geben die (x, y) Position des Klicks an.
panelResized()	Diese Methode wird aufgerufen, wenn sich die Größe des Spielfelds geändert hat.	Überschreiben Sie diese Methode, um Ihre Anzeige an die neue Feldgröße anzupassen.
hasEntities()	Diese Methode überprüft, ob aktuell Objekte auf dem Feld angezeigt werden.	Rufen Sie diese Methode auf, um zu testen, ob das Spielfeld leer ist.
isImageRegistered(String)	Diese Methode überprüft, ob ein bestimmtes Bild registriert wurde.	Rufen Sie diese Methode auf, um zu testen, ob ein Bild für eine Kiste, Wand usw. registriert wurde. Der Parameter ist die ID des Bildes.
registerImage(String, String)	Diese Methode registriert ein Bild zur Verwendung mit dem Framework.	Rufen Sie diese Methode auf, um eine Bilddatei zu laden, die Sie später verwenden möchten. Der erste Parameter ist die ID des Bildes, der zweite Parameter der Dateiname.
unregisterImage(String)	Diese Methode entfernt eine Bildregistrierung aus dem Framework.	Rufen Sie diese Methode auf, um eine bereits geladene Bilddatei wieder aus dem Framework zu entfernen. Dies ist z.B. nötig, wenn Sie Bilddateien zur Laufzeit austauschen möchten. Der Parameter ist die ID des Bildes.
placeEntity(String)	Diese Methode platziert das nächste Bild auf dem Spielfeld.	Rufen Sie diese Methoden auf, um das nächste Element (Arbeiter, Wand, Kiste usw.) auf dem Spielfeld zu platzieren. Das Feld wird dabei zeilenweise aufgefüllt. Der Parameter ist die ID des zu platzierenden Elements.
getWidth()	Diese Methode liefert die Breite des Spielfelds.	Rufen Sie diese Methoden auf, um die Breite des Spielfelds in Pixeln zu bestimmen.
getHeight()	Diese Methode liefert die Höhe des Spielfelds.	Rufen Sie diese Methoden auf, um die Höhe des Spielfelds in Pixeln zu bestimmen.

Tabelle 4: Wesentliche Methoden aus *gd1sokoban.template.ui.GamePanel*

Aspekt	**Geschätzter Zeitaufwand**
Einarbeitung in die Aufgabenstellung und Vorlagen	0.5 Tage
Einigung auf die grundsätzliche Klassen- und Packagehierarchie	0.5 Tage
Entwicklung der grafischen Benutzerschnittstelle	2 Tage
Umsetzung der grafischen Objekte	0.25 Tage
Steuerung der Figur mit Umsetzung in der GUI	0.25 Tage
Implementierung und Test des Parsers	3 Tage
Implementierung von *toString()*	0.25 Tage
Korrekte Leveldarstellung	2 Tage
Speichern und Laden von Spielzuständen	2 Tage
Tastensteuerung	0.5 Tage
Erkennen gelöster Level	0.5 Tage
Neustart und Beenden	0.25 Tage
Durchführen der öffentlichen Tests und Debugging	0.5 Tage

Tabelle 5: Zeitplanung für die Bearbeitung „nur“ der minimalen Ausbaustufe. Die Aufgaben werden in der Regel teilweise parallel von einzelnen oder mehreren Gruppenmitgliedern bearbeitet; dass die Summe mehr als 10 Tage umfasst, sollte Sie daher nicht stören.

Stufe	**Geschätzter Zeitumfang**
Minimale Ausbaustufe	2 - 3 Tage
Ausbaustufe I	3 - 4 Tage
Ausbaustufe II	2 - 4 Tage
Ausbaustufe III	2 - 4 Tage
Bonusaufgaben	Falls hier von Anfang an ein *starkes* Interesse bestehen sollte, sollten Sie sich am besten vom ersten Tag an Gedanken machen und möglichst zügig anfangen. . .

Tabelle 6: Zeitplanung für Bearbeitung aller Ausbaustufen