Projekt zur GdI 1 - Wintersemester 2009/2010

Guido Rößling, Jonas Marczona, Christian Merfels, Fabian Vogt

25. Januar 2010

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 Spiels *Plumber*. Die Aufgabenstellung stellt zunächst das zugrundeliegende Spiel 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 *Plumber*

Im Spiel Plumber (dt. Klempner) übernehmen Sie die Rolle eines (Notfall-)Klempners. An einer Stelle des Spielfelds wird nach einer bestimmten Zeit aus einem Rohr—der Quelle—Wasser austreten. Ihre Aufgabe ist es nun, die auf dem gesamten Spielfeld verteilten Rohre so anzuordnen, dass das Wasser von der Quelle zu einem im Boden verschwindenden Endstück—der Senke—laufen kann. Diese rettende Verbindung zur Vermeidung einer Überschwemmung müssen Sie dabei rechtzeitig fertigstellen, bevor das Wasser das letzte offene Rohrstück erreicht hat.

Dabei kann jedes Rohrstück *außer* der Quelle, der Senke oder bereits mit Wasser gefüllten Rohren durch ein Anklicken um 90 Grad rotiert werden. Das Wasser wird nach einer gewissen Zeit anfangen zu fließen und sich nach einer bestimmten Zeit immer um ein Spielfeld (Rohrstück) weiterbewegen.

Ein Level wird gewonnen, wenn das Wasser erfolgreich von der Quelle zur Senke fließen konnte. Wenn das Wasser wegen einer fehlenden Verbindung nicht weiter fließen kann, ist der Level verloren. Ein Rohrende am Spielfeldrand ist ebenfalls keine gültige Verbindung.

Abbildung 1 auf der nächsten Seite zeigt eine mögliche 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. Als Ergänzung zum Verständnis des Spielprinzips können Sie sich auch ein Beispiel unter `http://www.funny-games.biz/the-plumber.html` ansehen.

2 Organisation des Projekts

Der offizielle Bearbeitungszeitraum des Projekts beginnt *Montag, den 22. 02. 2010* und endet *Freitag, den 05. 03. 2010*. Zur Teilnahme am Projekt müssen sich *alle* Mitglieder einer gegebenen Projektgruppe im Portal im extra für das Projekt angelegten Kurs angemeldet haben und der gleichen Projektgruppe beigetreten sein. Das Portal unterstützt auch die Anmeldung direkt als Gruppe; wir raten **dringend dazu, diese Möglichkeit zu nutzen**! Bitte beachten Sie, dass bei der Eintragung

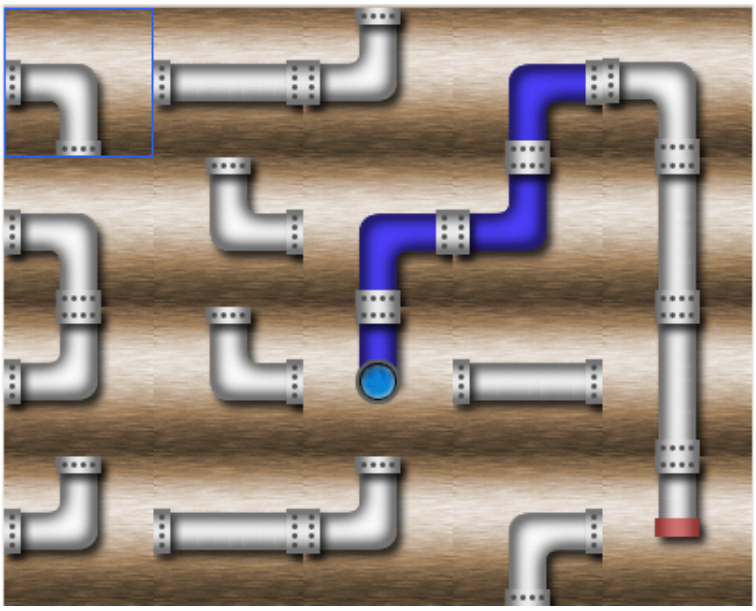

Abbildung 1: Beispiel einer grafischen Umsetzung von Plumber, bei der bereits Wasser in einer gültigen Verbindung fließt.

in die Projektgruppe nur Studierende für die eigene Projektgruppe ausgewählt werden, die bereits im Kurs zum Projekt *angemeldet* sind *und noch keiner Projektgruppe beigetreten sind*. Gruppen, die am Ende des Anmeldeintervalls noch nicht die Größe 4 haben, werden von uns mit zufällig gewählten anderen Studierenden aufgefüllt. Bitte versuchen Sie dies nach Möglichkeit zu vermeiden!

Die Aufgabenstellung wird vier Wochen vor Beginn der Projektphase, d.h. am *Montag, den 25. 01. 2010*, im Portal veröffentlicht. Ab diesem Zeitpunkt ist prinzipiell bereits die Bearbeitung der Aufgabe möglich. Da zu diesem Zeitpunkt die Gruppen im Lernportal noch nicht endgültig gebildet werden konnten, raten wir in diesem Fall aber *dringend* zu einer Anmeldung direkt als Gruppe von vier Studierenden in der entsprechenden Anmeldung im Lernportal.

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 der neu angelegte Kurs zum Projekt im Wintersemester 2009/2010 kann für alle gruppenübergreifenden Fragen zum Verständnis der Aufgabe oder Unklarheiten bei der Nutzung der Vorlagen genutzt werden.
- Jede Projektgruppe erhält im Portal eine eigene *Gruppe* sowie ein *Projektgruppenforum*. In diesem Projektgruppenforum können Fragen diskutiert oder Code-Fragmente ausgetauscht werden (Tipp: umgeben Sie Code im Portal immer mit `[code java] ... [/code]`, damit er besser lesbar ist). Da der Tutor der Gruppe ebenfalls der Projektgruppe angehören wird, ist er 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 Abstimmung über das im Portal erstellte Projektgruppenforum 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.* Dabei darf auch der Pfad zu den Testfällen *nicht* verändert werden.

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 abzusprechen.

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 dem Projektgruppenforum helfen, den Tutor „früher zu erreichen".

3 Ihre Aufgabe

Implementieren Sie eine lauffähige Java-Version des Spiels *Plumber*, die mindestens der „minimalen Ausbaustufe" entspricht.

Das fertige Spiel muss vom Tutor *vor Ende des Praktikums testiert werden*. Dazu müssen die Dokumentation (etwa 2 DIN A4-Seiten) sowie der Source-Code und alle zum Übersetzen notwendigen Bibliotheken und Dateien—außer den von uns im Portal bereitgestellten—rechtzeitig vor Ablauf der Einreichung **von einem Gruppenmitglied im Portal hochgeladen werden**.

Das Testat besteht aus den folgenden 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 die 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 Quelltexts 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 die folgenden Tools nutzen:

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

Dies sind nur unsere Empfehlungen. Es steht Ihnen selbstverständlich frei, andere Tools zu nutzen, etwa OpenOffice Draw, Microsoft Word etc. Bitte reichen Sie die Dokumentation und insbesondere das Klassendiagramm als **PDF-Datei** ein.

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 wir Ihre Abgabe ab einem gewissen Umfang der Nutzung von fremden Codes nur noch als gescheitert betrachten können, da der Eigenanteil zu gering wird.

Vorbereitung für automatisierte Tests - 0 Punkte Um die öffentlichen Tests korrekt ablaufen zu lassen, müssen Sie die Klasse `PlumberTestAdapterMinimal` korrekt implementieren. Diese Klasse soll *keine* Spiel-Funktionalität enthalten, sondern die einzelnen Methoden lediglich auf die entsprechenden Methoden in Ihrer Implementierung abbilden.

Objekte die Sie in diesem Adapter benötigen können Sie sich im Konstruktor erzeugen und „passend“ initialisieren.

Beispiel: Haben Sie sich entschieden, den Lösungsstand des aktuellen Levels in dem statischen Feld `isSolved` der Klasse `Level` zu speichern, so wäre dies eine passende Implementierung der Methode `isSolved()` der Klasse `PlumberTestAdapterMinimal`:

```
public boolean isSolved() {
  return Level.isSolved;
}
```

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 `PlumberTestAdapterMinimal`, Build path, exclude).

Öffentliche Tests sind erfolgreich - 0 Punkte Die öffentlichen Testfälle der Klasse `PlumberTestsuiteMinimal` 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 `PlumberTestAdapterMinimal` 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.

Diese Aufgabe umfasst nicht das Parsen der zusätzlichen Level-Informationen.

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 eingelesene Level syntaktisch korrekt ist.

Für unser Spiel Plumber bedeutet das: Ein Level ist genau dann syntaktisch korrekt, wenn er dem Dateiformat in Abschnitt 4 auf Seite 12 entspricht und es darüber hinaus genau eine Quelle und eine Senke gibt.

Es soll *keine* Lösbarkeitsbetrachtung durchgeführt werden.

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.

Grafische Benutzerschnittstelle - 5 Punkte Es gibt eine grafische Benutzerschnittstelle („GUI“), mit der man Plumber spielen kann.

Grafische Umsetzung der Objekte - 5 Punkte Für jedes Spielobjekt existiert eine passende grafische Umsetzung.

Wasserfluss - 5 Punkte Der Wasserfluss muss korrekt implementiert werden. Nach dem Start des Wassers soll dieses Schritt für Schritt fließen.

Bei jedem Schritt werden die nicht gefüllten Felder betrachtet, zu denen ein Rohr von einem schon mit Wasser gefüllten und benachbarten Feld zeigt. Besitzt das betrachtete Feld eine Rohrverbindung zu einem schon mit Wasser gefüllten Feld, so füllt es sich auch; ist jedoch keine Verbindung vorhanden, so ist das Spiel verloren.

Der Wasserfluss startet in der Quelle und stoppt in der Senke.

Zwischen den Schritten soll eine sichtbare Pause liegen, so dass man sehen kann, wie das Wasser fließt.

Achtung: Der Fluss innerhalb eines Feldes muss **nicht** animiert werden.

Erkennen gelöster Level - 5 Punkte Das Spiel erkennt von selbst, wenn ein Level erfolgreich gelöst wurde. Bei Plumber ist das der Fall, wenn zwischen Quelle und Senke eine korrekte Verbindung besteht, die mit Wasser gefüllt ist. Sobald dies passiert, soll eine entsprechende Meldung auf der Konsole ausgegeben werden.

Korrekte Leveldarstellung - 5 Punkte Alle eingeladenen Leveldateien müssen korrekt dargestellt werden. Betrachten Sie bitte dazu Abbildung 1 auf Seite 3 als Beispiel für eine grafische Umsetzung.

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.

Maussteuerung - 5 Punkte Das Spiel soll mit der Maus gesteuert werden. Ein Klick (im Folgenden mit der linken Maustaste) auf ein Feld bewirkt, dass dieses Feld im Uhrzeigersinn rotiert wird. Ein Klick auf eine Quelle oder Senke bleibt wirkungslos. Ebenso sollen Felder, die bereits mit Wasser gefüllt sind, nicht mehr rotierbar sein. Achten Sie jedoch darauf, keine Benutzereingabe—wie das Drücken einer Taste—zu erzwingen, da dies die automatisierten Tests nicht leisten.

Das Framework stellt Operationen bereit, die Sie nutzen können, um diese Funktionalität zu implementieren.

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.**

Bei Implementierung dieser Ausbaustufe können insgesamt 75 aus 100 möglichen Punkten erreicht werden. Die öffentlichen Testfälle für diese Ausbaustufe befinden sich in der Klasse `PlumberTestsuiteExtended1`.

Öffentliche Tests der Ausbaustufe 1 sind erfolgreich - 0 Punkte Die öffentlichen Testfälle der Klasse `PlumberTestsuiteExtended1` werden fehlerfrei absolviert. Beachten Sie, dass Sie für den Ablauf der Tests die Klasse `PlumberTestAdapterExtended1` implementieren müssen!

MVC - 7 Punkte Strukturieren Sie Ihr Programm nach dem Prinzip *MVC („Model View Controller“)* und führen Sie eine sinnvolle Einteilung in Pakete (*package hierarchy*) ein. Erstellen Sie ein Klassendiagramm in UML, welches die Struktur des Programms wiedergibt, und markieren Sie darin jeweils die Klassen der Bereiche Model, View und Controller. Ihr Tutor wird sich das Klassendiagramm ansehen und ggf. Fragen hierzu stellen. Insbesondere beim Code-Review wird auf die Umsetzung des MVC-Prinzips geachtet.

Highscore-Liste - 5 Punkte Für jedes Levelset (siehe Abschnitt 4.3 auf Seite 13) 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.

Die Highscore-Einträge sollen nach den folgenden Kriterien in der folgenden Reihenfolge (wichtigstes Sortierkriterium zuerst) sortiert werden:

1. Levelnummer innerhalb des Levelsets (absteigend)
2. Zeit bis Spielende (aufsteigend)
3. Rotationsschritte (aufsteigend)

Highscore-Einträge werden nur dann erstellt und gespeichert, wenn der jeweilige Level erfolgreich gelöst wurde.

Highscore-Liste GUI - 5 Punkte Die Highscore-Liste soll in der GUI angezeigt werden, wobei die Einträge entsprechend der Sortierung der Highscore-Liste angezeigt werden sollen. Sie müssen zu einem geeigneten Zeitpunkt den Namen des Spielers abfragen, um ihn in die Highscore-Liste eintragen zu können. Beachten Sie, dass alle Testfälle ohne Benutzerinteraktion durchlaufen müssen.

Spielstand sichern - 4 Punkte Der aktuelle Spielstand soll sich speichern lassen. Ein Spielstand umfasst den eventuell vorhandenen *Spielernamen* (für die Highscore-Liste), den *Namen des aktuellen Levelsets*, die *Nummer des aktuellen Levels im Levelset*, die *Anzahl der bisher durchgeführten Rotationsschritte*, die *bisher vergangene Zeit* und den *aktuellen Zustand des Spielfeldes*. Überlegen Sie sich hierzu ein geeignetes Dateiformat. Die einzelnen Spielstände sollen mit der Dateiendung „.sve“ im Level-Verzeichnis gespeichert werden.

Spielstand laden - 2 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 (siehe Abschnitt 3.6 auf der nächsten Seite möglich sein *muss*.

Wasserstart - 2 Punkte Stellen Sie eine Möglichkeit bereit, das Wasser starten zu lassen, bevor die Zeit bis zum automatischen Wasserstart abgelaufen ist. Dies kann z.B. über die GUI oder einen Hotkey geschehen.

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 `PlumberTestsuiteExtended2`.

Öffentliche Tests der Ausbaustufe 2 sind erfolgreich - 0 Punkte Die öffentlichen Testfälle der Klasse `PlumberTestsuiteExtended2` werden fehlerfrei absolviert. Beachten Sie, dass Sie für den Ablauf der Tests die Klasse `PlumberTestAdapterExtended2` implementieren müssen!

Undo und Redo - 5 Punkte Der Benutzer soll eine beliebige Anzahl an Spielzügen rückgängig machen bzw. wiederherstellen können. Um diese Funktionalität wirklich nützlich zu machen, bauen Sie bitte Tastaturshortcuts ein: Backspace-Taste für Undo, Enter-Taste für Redo. Ist im aktuellen Zustand kein Undo bzw. Redo möglich (z.B. kein Undo bei neu begonnenem Spiel), so bleiben die Funktionen wirkungslos.

Weitere Level-Informationen - 2 Punkte Das Levelformat lässt die Möglichkeit offen, weitere Zusatzinformationen zu einem Level anzugeben. Diese Informationen sollen beim Laden eines Levels beachtet werden. Weitere Informationen zu den Einstellmöglichkeiten finden Sie in Abschnitt 4 auf Seite 12.

Skin wechseln - 2 Punkte Erweitern Sie Ihr Spiel um zusätzliche Skins, sowie eine Möglichkeit, aus der Benutzerschnittstelle 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).

„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 Programmen. Seien Sie kreativ :-)

Tastensteuerung - 5 Punkte Erweitern sie das Spiel um eine Tastatursteuerung. Es soll über die Pfeiltasten gesteuert werden. Die Markierung ist erkennbar darzustellen, etwa durch eine farbige Umrandung des Spielfeldes. Jeder Tastendruck entspricht der Markierung des nächsten Spielfeldes in der entsprechenden Richtung. Die Markierung kann nicht über die Grenzen des Spielfeldes hinaus verschoben werden. Befindet sich auf dem aktuell markierten Spielfeld ein rotierbarers Spielobjekt, so wird es mit Druck auf die Leertaste im Uhrzeigersinn rotiert.

Zu Beginn muss das Spielfeld in der linken oberen Ecke markiert sein.

Sie können die im Framework vorhandenen Methoden zur Tastatursteuerung überschreiben und in Ihrer Implementierung verwenden.

Hinweis: Der Code in den Testklassen **darf auf keinen Fall** zur Anzeige von Dialogfenstern führen. Dies darf auch bei inkorrekten Schritten nicht geschehen, da sonst eine Eingabe erfordert würde, die in den JUnit-Tests nicht automatisiert erfolgen kann.

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 `PlumberTestsuiteExtended3`.

Öffentliche Tests der Ausbaustufe 3 sind erfolgreich - 0 Punkte Die öffentlichen Testfälle der Klasse `PlumberTestsuiteExtended3` werden fehlerfrei absolviert. Beachten Sie, dass Sie für den Ablauf der Tests die Klasse `PlumberTestAdapterExtended3` implementieren müssen!

Solver-Funktion - 5 Punkte Implementieren Sie eine Solver-Funktion, die der Spieler durch Druck auf die **x**-Taste aufrufen kann. Ist das Spiel nicht lösbar, so soll der Spieler hierüber informiert werden. Wurde eine Zug-Sequenz gefunden, die das Spiel löst, so soll diese durchgeführt werden.

Hinweis: Für die automatisierten Tests muss diese Funktion ohne weitere Verzögerung oder Animation ablaufen.

Generator-Funktion - 5 Punkte Implementieren Sie eine Generator-Funktion, die der Spieler durch Druck auf die **g**-Taste aufrufen kann. Dabei soll ein Spielfeld mit parametrisierbarer Höhe und Breite erzeugt werden. Alle generierte Level müssen lösbar sein. Achten Sie weiterhin darauf, dass im Allgemeinen die so erzeugten Spielfelder eine angemessene Schwierigkeit haben; das bedeutet unter Anderem, dass generierte Level nicht übermäßig trivial lösbar oder bereits gelöst sind.

Durch den Generator erzeugte Level sollen in keiner Highscore berücksichtigt werden.

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.

Nutzung von Audio-Clips - 2 Punkte Audio-Clips werden bei bestimmten Ereignissen abgespielt (z.B. einem ungültigen Zug oder einem gelösten Level). Sie können hierfür das von uns im Portal bereitgestellte *JavaLayer*-Framework nutzen. Siehe auch Abschnitt 5.4 auf Seite 15.

Übersetzung - 3 Punkte Übersetzen Sie das Programm in verschiedene Sprachen, mindestens jedoch in Englisch. Die Sprachen sollen sich im Spiel auswählen lassen und direkt übernommen werden können, ohne das Spiel abbrechen oder das Programm neu starten zu müssen.

Hinweis: Nutzen Sie dazu die aus Foliensatz T21 bekannte *translator*-Bibliothek, die im Portal bereitgestellt wird.

Level-Erweiterungen - 5 Punkte Erweitern Sie Ihre Level, indem Sie z.B. mehrere Quellen und Senken erlauben oder neue Spielelemente kreieren.

Hinweis: Achten Sie darauf, dass die Erweiterungen die Testfälle nicht beeinträchtigen. Dies betrifft besonders die Spiellogik und die Level-Verarbeitung.

Überraschen Sie uns - 0 Punkte Sie können auch eigene Ideen zum Ausbau des Spiels einbringen. Die Bewertung mit Punkten ist dann Sache des Tutors und der Veranstalter. Die 0 ist also *nicht wörtlich zu nehmen*.

4 Dateiformate

4.1 Level-Dateien

Plumber verwendet ein einfaches Textformat zur Definition von Spielleveln. Jeder Level steht in einer eigenen Datei, deren Name die Form `name.lvl` hat. Die Datei ist zeilenweise aufgebaut. Jede Zeile der Datei entspricht einer Zeile im Level. Alle Zeilen werden mit einem Zeilenvorschub „\n“ abgeschlossen. Um 144 Spielsteine in einem 8x18-Feld zu repräsentieren, werden entsprechend 8 Zeilen mit jeweils 18 Zeichen gespeichert.

Zur Vereinfachung sind alle Spielfelder rechteckig.

Ein Zeichen δ repräsentiert einen Spielstein, wobei $\delta \in \{1, ..., 6\} \cup \{a, ..., m\}$. Siehe Tabelle 1.

Objekt	Zeichen	Ausgänge
Quelle	a	oben
Quelle	b	rechts
Quelle	c	unten
Quelle	d	links
Senke	k	oben
Senke	l	rechts
Senke	m	unten
Senke	n	links
Ecke	1	links und unten
Ecke	2	links und oben
Ecke	3	rechts und unten
Ecke	4	rechts und oben
Gerade	5	oben und unten
Gerade	6	rechts und links

Tabelle 1: Format der vorgegebene Spielsteine

Optional kann eine Zeile mit ### beginnen. In dieser Zeile sind nur Zusatzinformationen zum Level gespeichert. Eine Zusatzinformationszeile sähe z.B. so aus:

```
###time_to_water:10000|time_between_water:1000
```

Dabei kann die Zahl jeweils eine beliebige Zeitangabe (in Millisekunden) sein. Falls eine Datei diese Zusatzinformationen nicht enthält, soll das Wasser nach *30 Sekunden* zu fließen beginnen und *alle 750ms* ein Feld weiterfließen.

Abbildung 1 auf Seite 3 zeigt ein Beispiel für eine denkbare grafische Umsetzung eines gültigen Levels. In dem Beispiel ist bereits Wasser geflossen.

Abbildung 2 zeigt die Codierung des aktuellen Spielstands aus der Abbildung 1 gemäß Tabelle 1:

```
16231
14325
24a65
2623k
```

Abbildung 2: Codierung des Levels aus Abbildung 1

4.2 Highscore-Dateien

Eine Highscore-Liste gehört zu genau einem Levelset. Der Dateiname der Highscore-Liste ist `highscore.hsc`. Die Datei ist zeilenweise aufgebaut, wobei jede Zeile einen Datensatz in der Liste repräsentiert. Woraus ein Datensatz (mindestens) bestehen soll und wie er sortiert werden soll wurde auf Seite 9 beschrieben.

4.3 Levelset

Ein Verzeichnis stellt ein *Levelset* dar. Alle Levels aus einem Verzeichnis stellen ein Levelset dar. Dieses soll in immer gleicher Reihenfolge (gemäß der alphabetischen Sortierung der Dateinamen) durchspielt werden können. Es soll pro Levelset eine Highscore-Datei geben.

Wenn nach dem manuellen Laden eines Levels (oder gespeicherten Spieles) nicht mehr bekannt ist, aus welchem Levelset dieses Level ursprünglich stammt, so muss kein Highscore-Eintrag geschrieben werden.

5 Materialien

Auf der Veranstaltungsseite stellen wir Ihnen einige Dateien zur Verfügung, die Sie für Ihre Lösung verwenden können. Dabei handelt es sich um vorgefertigte Level und die öffentlichen Testfälle. Wenn Sie möchten, können Sie unser bereitgestelltes Framework nutzen. Sie können aber auch vollständig eigene Lösungen implementieren.

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 *de.tu_darmstadt.gdi1.plumber.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 und Mausklicks zu reagieren.

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 *de.tu_darmstadt.gdi1.plumber.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. Um unschöne Ränder bei der Darstellung zu vermeiden, sollten alle Bilder die gleiche Größe haben.

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 oder das Portal. 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 die übergebenen Parameter. Die Exception speichert eine Fehlermeldung.

ParameterOutOfRangeException 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 *JLayer*

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

Hinweis: Diese Soundbibliothek ist zwar angemessen einfach zu bedienen—ist jedoch nicht perfekt. Es treten betriebssystemabhängig Schwierigkeiten auf, wenn in schneller Folge hintereinander—oder parallel—Sounds abgespielt werden sollen. Dies wird von uns *nicht* bewertet. Bitte verwenden Sie daher keine Zeit zum Lösen solcher Probleme.

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 20 gezeigt aussehen. Eine denkbare Gesamteinteilung für alle Ausbaustufen wird in Tabelle 6 auf Seite 20 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 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.
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.
Die folgende Methoden können überschrieben werden. Sie sind jedoch **nicht** abstrakt, da sie erst für spätere Ausbaustufen benötigt werden.		
keyLeftPressed()	Diese Methode wird aufgerufen, wenn die linke Pfeiltaste gedrückt wurde.	Überschreiben Sie diese Methode, um auf den Tastendruck zu reagieren.
keyRightPressed()	Diese Methode wird aufgerufen, wenn die rechte Pfeiltaste gedrückt wurde.	Überschreiben Sie diese Methode, um auf den Tastendruck zu reagieren.
keyUpPressed()	Diese Methode wird aufgerufen, wenn die Pfeiltaste nach oben gedrückt wurde.	Überschreiben Sie diese Methode, um auf den Tastendruck zu reagieren.
keyDownPressed()	Diese Methode wird aufgerufen, wenn die Pfeiltaste nach unten gedrückt wurde.	Überschreiben Sie diese Methode, um auf den Tastendruck zu reagieren.
keyNewGamePressed()	Diese Methode wird aufgerufen, wenn die Taste *n* gedrückt wurde.	Überschreiben Sie diese Methode, um den Level auf den Ursprungszustand zurückzusetzen.
keyQuitPressed()	Diese Methode wird aufgerufen, wenn die Taste *q* gedrückt wurde.	Überschreiben Sie diese Methode, um das Spiel zu beenden.
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.

Tabelle 2: Ereignisbehandlungsmethoden in der Klasse *de.tu_darmstadt.gdi1.plumber.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 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 bestimmtes Bild 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 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 3: Wesentliche Methoden aus *de.tu_darmstadt.gdi1.plumber.ui.GamePanel*

Aspekt	**Geschätzter Zeitaufwand**
Einarbeitung in die Aufgabenstellung und Vorlagen	1 Tag
Einigung auf die grundsätzliche Klassen- und Packagehierarchie	0.5 Tage
Entwicklung der grafischen Benutzerschnittstelle	2 Tage
Umsetzung der grafischen Objekte	0.25 Tage
Implementierung und Test des Parsers	1 Tag
Implementierung von *toString()*	0.25 Tage
Korrekte Leveldarstellung	0.75 Tage
Check auf syntaktische Korrektheit	0.5 Tage
Spielsteuerung über Maus	0.5 Tage
Erkennen gelöster Level	0.25 Tage
Neustart und Beenden	0.10 Tage
Wasserfluss	0.75 Tage
Durchführen der öffentlichen Tests und Debugging	0.5 Tage

Tabelle 4: Zeitplanung für die Bearbeitung „nur“ der minimalen Ausbaustufe. Die Aufgaben werden in der Regel parallel von einzelnen oder mehreren Gruppenmitgliedern bearbeitet.

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 5: Zeitplanung für Bearbeitung aller Ausbaustufen