Projekt zu GdI 1 - Sommersemester 2012

Guido Rößling, Timo Bähr, Bernd Conrad, Alexander Jandousek

4. Juli 2012

Version RC3

Inhaltsverzeichnis

1 Einführung in das Projekt

Im Rahmen des Projekts implementieren die Studierenden in Gruppen von jeweils vier Personen eine Java-Version des Spiels *Tanks*. 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 *Tanks*

Im Spiel *Tanks* steuern Sie einen Panzer und versuchen, ein Territorium zu erobern. Sie fahren dazu mit dem Panzer durch das Gebiet, beseitigen im Weg stehende Hindernisse und zerstören gegnerische Panzer. Sie haben eine Karte gewonnen, wenn alle Gegner auf der Karte eliminiert sind. Sie haben verloren, wenn Ihr eigener Panzer zerstört wird.

Auf einem beliebig großen rechteckigen Spielfeld kann es folgende Spielobjekte geben:

Panzer Ein Panzer hat so viele Lebenspunkte, wie in der geladenen Karte vorgegeben. Panzer unterschiedlicher Parteien müssen visuell unterscheidbar sein.

Wand Eine Wand hat so viele Lebenspunkte, wie in der geladenen Karte vorgegeben. Panzer können nicht durch Wände fahren.

Unbesiegbare Wand Es gibt unbesiegbare Wände. Diese sind für den Spieler nicht sichtbar. Sie dienen also ausschließlich zur Begrenzung des Spielfelds.

Schuss Ein Schuss besitzt eine Schussstärke abhängig von dem Panzer, der ihn abgefeuert hat. Trifft der Schuss auf ein zerstörbares Spielobjekt, so löst er eine Kollisionsaktion aus und erniedrigt den Lebenszähler der getroffenen Einheit um die Schussstärke. Anschließend wird das Schussobjekt wieder entfernt. Bei der Kollision mit einem sichtbaren Spielobjekt wird eine Explosion angezeigt.

Spielobjekte dürfen sich generell nicht überlagern.

Abbildung 1 auf der nächsten Seite zeigt ein mögliches Menü. Abbildung 2 auf Seite 4 zeigt ein mögliches Spielfenster. Wir erwarten nicht, dass das im Rahmen des Projekts erstellte Spiel der Ausgabe optisch ähnlich sieht; die Abbildung soll nur zum besseren Nachvollziehen dienen.

Abbildung 1: Beispiel einer grafischen Umsetzung des Menüs von Tanks

2 Organisation des Projekts

Der offizielle Bearbeitungszeitraum des Projekts beginnt *Montag, den 13.08.2012* und endet *Freitag, den 24.08.2012*. 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. Wir raten **dringend** dazu, sich möglichst früh in einer gemeinsamen Projektgruppe anzumelden. Bitte beachten Sie, dass bei der Eintragung in die Projektgruppe nur Studierende einer Projektgruppe beitreten können, 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 16.07.2012*, 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 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 Sommersemester 2012 kann für alle gruppenübergreifenden Fragen zum Verständnis der Aufgabe oder Unklarheiten bei der Nutzung der Vorlagen genutzt werden.

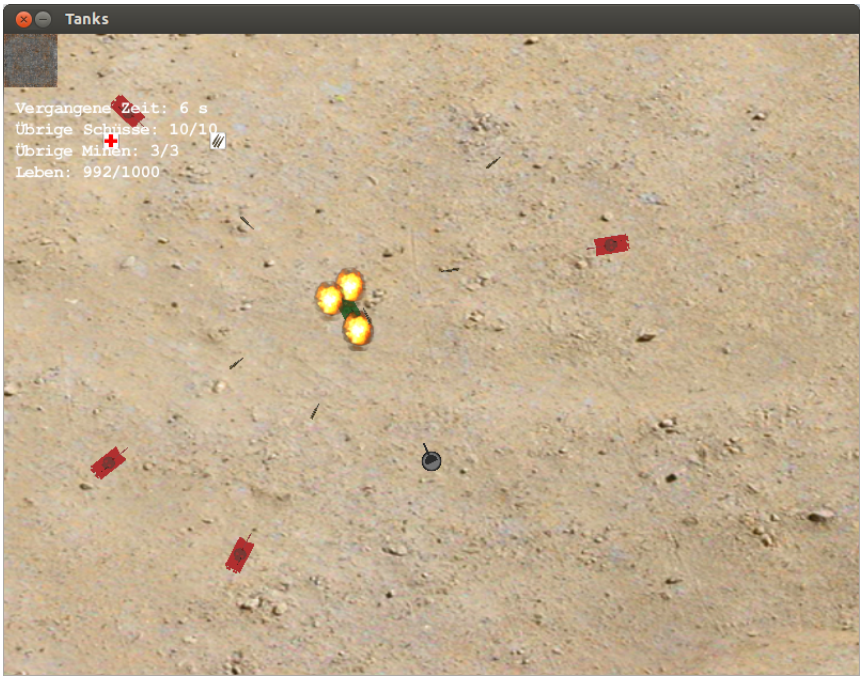

Abbildung 2: Beispiel einer grafischen Umsetzung des Spielfensters von Tanks

- 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 zu vereinbaren.

Sie sollten auch einen Termin für die erste Besprechung mit dem Tutor absprechen. 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 *Tanks*, die mindestens der „minimalen Ausbaustufe“ entspricht.

Das fertige Spiel muss vom Tutor *vor Ende des Projekts 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 sowie 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 Projekt 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

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

Sollten Sie sich für die Nutzung des vorgegebenen Frameworks entscheiden, so nutzen Sie das Konzept von Entitäten, Komponenten und Aktionen. Ihre Tutoren werden bewerten, wie gut Ihnen das gelungen ist.

Sollten Sie nicht das vorgegebene Framework verwenden, so sollten Sie in Ihrem Code strikt zwischen Logik (Code) und Darstellung (Design) unterscheiden. Ihre Tutoren werden bewerten, wie gut diese Trennung bei Ihnen gelungen ist.

3.4 Minimale Ausbaustufe

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

Fehlende Punkte aus der minimalen Ausbaustufe können nur in Ausnahmefällen ausgeglichen 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. **Die Projektgruppe** ist 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 Projektgruppen 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 oder Bibliotheken, 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 `TanksTestAdapterMinimal` 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 im Konstruktor erzeugen und „passend“ initialisieren.

Beispiel: Haben Sie sich entschieden, die vergangene Zeit der aktuellen Karte in dem statischen Feld `elapsedTime` der Klasse `StopWatch` zu speichern, so wäre dies eine passende Implementierung der Methode `getElapsedTime()` der Klasse `TanksTestAdapterMinimal`:

```
public boolean getElapsedTime() {
  return StopWatch.elapsedTime;
}
```

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 `TanksTestAdapterMinimal`, Build path, exclude).

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

Parsen von Karten - 10 Punkte Das von der Gruppe erstellte Spiel kann dem Dateiformat in Abschnitt 4 auf Seite 15 entsprechende Karten korrekt parsen. Das bedeutet, dass Karten fehlerfrei aus einer Datei eingelesen und in die interne Darstellung umgewandelt werden können. Die Art der internen Datenrepräsentation (Datenstruktur) bleibt Ihnen überlassen.

Beim Einlesen einer Karte wird überprüft, ob die eingelesene Karte syntaktisch korrekt ist. Ist eine Karte nicht syntaktisch korrekt, so muss eine **de.tu_darmstadt.gdi1.tanks.model.exceptions.SyntaxException** geworfen werden. Für unser Spiel Tanks bedeutet das, eine Karte ist genau dann syntaktisch korrekt, wenn ...

1. ... sie das Tupel `Map` beschrieben in Abschnitt 4 auf Seite 15 enthält
2. ... alle Tupel dem Dateiformat in Abschnitt 4 auf Seite 15 entsprechen.

Hinweis: Sie müssen nicht alle Features implementieren, ihr Parser muss aber trotzdem mit dem Format umgehen können. So muss zwar die Zeitbegrenzung (Ausbaustufe II) nicht implementiert werden, der Parser muss aber trotzdem den Wert dafür parsen können.

Tipp: Erinnern Sie sich bitte an das Anwendungsbeispiel aus T20-Streams Folie 59. :-)

Check auf semantische Korrektheit - 2 Punkte Beim Einlesen einer Karte wird überprüft, ob die eingelesene Karte semantisch korrekt ist. Ist eine Karte nicht semantisch korrekt, so muss eine **de.tu_darmstadt.gdi1.tanks.model.exceptions.SemanticException** geworfen werden.

Für unser Spiel Tanks bedeutet das: Eine Karte ist genau dann semantisch korrekt, wenn ...

1. ... Panzer nicht die gleiche Position wie ein anderes Spielobjekt haben.
2. ... es mindestens einen Spielerpanzer und mindestens einen gegnerischen Panzer gibt.

Korrekte *toString()*-Methode - 3 Punkte Jede geladene Karte kann als String ausgegeben werden. Der ausgegebene String soll dabei dem Dateiformat in Abschnitt 4 auf Seite 15 entsprechen. Es soll der aktuelle Spielzustand wiedergegeben werden, nicht der Anfangszustand.

Grafische Benutzerschnittstelle - 3 Punkte Es gibt eine grafische Benutzerschnittstelle („GUI“), mit der man *Tanks* spielen kann.

Grafische Umsetzung der Objekte - 4 Punkte Für jedes Spielobjekt und jeden Spielzustand existiert eine passende grafische Umsetzung gemäß der Spielbeschreibung.

Korrekte Kartendarstellung - 4 Punkte Alle eingeladenen Kartendateien müssen korrekt dargestellt werden. Betrachten Sie bitte dazu Abbildung 2 auf Seite 4 als Beispiel für eine grafische Umsetzung.

Tastatursteuerung - 3 Punkte Das Spiel soll mit der Tastatur gesteuert werden. Über die Pfeiltasten (siehe auch Abbildung 3 auf Seite 18) kann der eigene Panzer auf der Karte navigiert werden. Über die **k**-Taste wird ein Schuss ausgelöst.

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

Einfacher Computergegner - 6 Punkte Der Computergegner bewegt sich in Richtung des Spielers und schießt pro Sekunde einmal, wenn er in Richtung des Spielers gedreht ist.
Hinweis: Später soll der Computergegner „intelligenter" implementiert werden. Der Benutzer soll dann selbst die Schwierigkeit wählen können. Wenn Sie den erweiterten Computergegner implementieren, so bekommen Sie auch diese Punkte.

Schussverhalten - 5 Punkte Ein Schuss beginnt an der Position des Panzers und bewegt sich nur in eine Richtung. Trifft er auf ein zerstörbares Hindernis, so werden die Lebenspunkte des getroffenen Objekts um die Schussstärke erniedrigt und es wird eine Explosion ausgelöst. Anschließend wird das Schussobjekt entfernt. Beträgt der Lebenszähler des getroffenen Objekts 0, so wird das Objekt nicht länger angezeigt und das entsprechene Objekt entfernt. Fliegt ein Schuss aus dem sichtbaren Spielfeldbereich, so wird das Schussobjekt ebenfalls gelöscht. Die Explosion soll keinen Schaden an benachbarten Objekten auslösen.

Keine Bewegung durch Hindernisse - 2 Punkte Panzer können nicht durch Wände oder andere Panzer fahren. Wird dies versucht, so führt dies zu keiner Reaktion und ist gleichzusetzen mit keiner Bewegung.

Erkennen gewonnener und verlorener Karten - 4 Punkte Das Spiel erkennt von selbst, wenn ein Spiel gewonnen wurde. Bei Tanks ist das der Fall, wenn alle gegnerischen Panzer auf dem Spielfeld zerstört wurden. Sobald dies passiert, soll eine entsprechende Meldung ausgegeben werden. Ist in der Karte eine Folgekarte gespeichert, so soll die nächste Karte anschließend automatisch gestartet werden. Ist keine weitere Karte mehr verfügbar, so soll zurück in das Hauptmenü gewechselt werden.

Eine Karte gilt als verloren, wenn der eigene Panzer vernichtet wurde, also 0 oder weniger Lebenspunkte besitzt. In Karten mit Zeitbeschränkung (siehe auf Seite 12) gilt eine Karte auch als verloren, wenn der Spieler sie nicht innerhalb der Zeitbeschänkung gewonnen hat.

Tipp: Das Framework lässt sich auch mit Swing-Komponenten kombinieren. Werfen Sie einen Blick auf *javax.swing.JOptionPane.showMessageDialog(...)*. Fehlt Ihnen ein javax.swing.JFrame für ein Dialogfenster, so können Sie ein leeres und nicht sichtbares Fenster mit `new JFrame("")` erzeugen.

Neues Spiel starten - 2 Punkte Der Spieler soll über die Taste **n** aus dem Hauptmenü heraus ein neues Spiel starten können. Dazu wird die erste Karte mit dem Namen `map00.tanks` aus der Menge der mitgelieferten Karten geladen.

Spielmenü - 2 Punkte Das Spiel Tanks muss in einem Spielmenü beginnen. Das Menü muss mindestens jeweils einen Button „Neues Spiel starten" und „Beenden" mit entsprechender Semantik enthalten. Im Spiel soll das Spielmenü durch Drücken der **Escape**-Taste erscheinen und bei nochmaligem Drücken soll wieder das Spiel selbst angezeigt werden.

Hinweis: Sollten Sie nicht das *eea*-Framework nutzen, so dürfen Steuerungsbefehle im Menü keine Auswirkung auf Spielzüge haben. Während das Menü sichtbar ist, ist das Spiel komplett pausiert. Im *eea*-Framework ist dieses Verhalten bereits standardmäßig der Fall.

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 `TanksTestsuiteExtended1`.

Öffentliche Tests der Ausbaustufe 1 sind erfolgreich - 0 Punkte Die öffentlichen Testfälle der Klasse `TanksTestsuiteExtended1` werden fehlerfrei absolviert. Beachten Sie, dass Sie für den Ablauf der Tests die Klasse `TanksTestAdapterExtended1` implementieren müssen!

Sinnvolle Modellierung - 5 Punkte Wenn Sie das vorgegebene *eea*-Framework verwenden, dann achten Sie bei Ihrer Implementierung auf die sinnvolle Verwendung von Entitäten, Komponenten und Aktionen. Verwenden Sie Packages zur Strukturierung Ihres Codes. Ihr Tutor wird im Rahmen des Code-Review bewerten, wie gut Ihnen die Umsetzung gelungen ist.

Andernfalls strukturieren Sie Ihr Programm nach dem Prinzip *MVC („Model View Controller“)*, siehe T21.41-48. 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.

Spielstand sichern - 4 Punkte Der aktuelle Spielstand (und nicht der anfangs geladene Spielstand) soll sich speichern lassen. Ein Spielstand entspricht dem Format beschrieben in Abschnitt 4 auf Seite 15. Explosionen müssen ebenso wie die *bisher vergangene Zeit* und die *verbrauchten Schüsse* mitgespeichert werden. Die einzelnen Spielstände sollen mit der Dateiendung `.tanks` im Verzeichnis `saves` gespeichert werden. Erweitern Sie dazu das Menü um einen Eintrag “Spiel speichern“ mit entsprechender Semantik. Das Menu muss sich aus dem Spielfenster über die **Escape**-Taste aufrufen lassen. Das Speichern muss neben einem Klick mit der Haus auch über die **s**-Taste erfolgen können.

Hinweis: Der Eintrag muss sich nicht zwangsläufig im Hauptmenü befinden. Sie können gerne eine bessere Unterteilung im Menü vornehmen.

Hinweis: Beachten Sie, dass bei den automatisierten Tests keine Fenster geöffnet werden dürfen. Beachten Sie also stets, dass ihr Programm auch ohne UI vollständig lauffähig ist. Für das Speichern bei Start ohne Fenster soll der aktuelle Spielstand automatisch in die Datei `saves/autosave.tanks` in ihrem Projektordner gespeichert werden.

Tipp: Informieren Sie sich über die Klasse *javax.swing.JFileChooser*. Diese bietet eine Komponente zur Auswahl von Dateinamen für das Laden und Speichern.

Spielstand laden - 2 Punkte Spielstände, die wie oben gesichert wurden, sollen sich wieder laden lassen. Dabei wird der gespeicherte Zustand wieder hergestellt. Erweitern Sie das Hauptmenü um einen Eintrag "Spiel laden" mit entsprechender Semantik.

Hinweis: Die bisher vergangene Zeit soll natürlich korrekt weiter gezählt werden. Dabei bezieht sich die Zeit nur auf die Zeiträume die der Spieler gespielt hat. Zeiträume in denen das Spiel pausiert oder nur gespeichert war sollen also *nicht* mitgezählt werden.

Spiel pausieren - 2 Punkte Durch Drücken der Taste **p** soll das aktuelle Spiel angehalten werden können. Wird erneut **p** gedrückt, so wird das Spiel fortgesetzt.

Highscore-Liste - 5 Punkte Für jede Karte soll eine Highscore-Liste persistent gespeichert und aktualisiert (und daher auch eingelesen) werden. Das Dateiformat, das für die Highscore-Liste verwendet werden soll bleibt Ihnen überlassen, muss aber die Endung `hsc` haben. Die Liste soll höchstens 10 Einträge aufnehmen können.

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

1. benötigte Zeit (aufsteigend)
2. Anzahl getätigter Schüsse (aufsteigend)

Einträge in die Highscore-Listen werden nur dann erstellt und gespeichert, wenn die jeweilige Karte gewonnen wurde. Bei Gleichheit der Kriterien wird nach Namen sortiert.

Highscore GUI - 5 Punkte Die Highscore-Liste soll in der GUI angezeigt werden, wobei die Einträge entsprechend der Sortierung der Highscore-Liste angezeigt werden sollen. Durch Drücken der **Escape**-Taste soll zurück ins Hauptmenü gewechselt. Sie müssen bei Auftreten eines neuen Highscores nach Beenden des Spiels den Namen des Spielers abfragen, um ihn in die Highscore-Liste eintragen zu können. Beachten Sie, dass alle Testfälle ohne Benutzerinteraktion durchlaufen müssen.

Minen legen - 2 Punkte Führen Sie ein Hindernis Mine ein. Diese ist dauerhaft sichtbar. Ein Panzer kann eine Mine nicht abschießen. Fährt ein Panzer über die Mine, so bekommt er doppelt so viel Schaden wie von einem Schuss. Beträgt der Lebenszähler 0, so ist das Spiel verloren. Spieler können mit der Taste **m** selbst Minen legen. Diese werden erst nach 2000 ms aktiv und können auch den Spieler selbst zerstören. Zudem ist die Anzahl an Minen begrenzt und die verbleibende Anzahl wird im Spiel angezeigt. Alle fünf Sekunden erhält der Spieler eine weitere Mine. Die Maximalanzahl an Minen kann aber nicht überschritten werden.

Der Parser muss damit um folgendes Tupel erweitert werden:

`Mine strength scale x y`
`strength`: int - Schadenswert
`scale`: int - Skalierung der Mine (je größer, desto größer die Mine)
`x`: int - x-Position des Mittelpunkts (in Pixeln)
`y`: int - y-Position des Mittelpunkts (in Pixeln)

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 `TanksTestsuiteExtended2`.

Öffentliche Tests der Ausbaustufe 2 sind erfolgreich - 0 Punkte Die öffentlichen Testfälle der Klasse `TanksTestsuiteExtended2` werden fehlerfrei absolviert. Beachten Sie, dass Sie für den Ablauf der Tests die Klasse `TanksTestAdapterExtended2` implementieren müssen!

Zeitbegrenzung - 2 Punkte Eine Karte muss innerhalb einer gewissen Zeit absolviert werden. Ist die Zeit abgelaufen, so hat der Spieler ebenfalls verloren. Die Restzeit soll im Spielfenster dargestellt werden. Beachten Sie hierzu das Format, beschrieben in Abschnitt 4 auf Seite 15.

Begrenzte Munition - 3 Punkte Erweitern Sie das Spiel so, dass die Munition des Spielers nur begrenzt verfügbar ist. Der Munitionsstand soll im Spielfenster angezeigt werden. Beim Laden soll der Munitionsstand wieder angezeigt werden. Alle fünf Sekunden bekommt der Spieler wieder eine Munition hinzu.

Hinweis: Der Computergegner hat weiter unbegrenzt viele Schüsse.

Streuschüsse - 3 Punkte Führen Sie einen Typ Streuschuss ein. Dieser teilt sich ein Mal nach 500 ms in fünf Schüsse mit jeweils 1/5 der Schusstärke auf. Die Richtung der kleineren Schüsse muss unterschiedlich sein. Wie, bleibt Ihnen überlassen. Der Spieler kann den Schuss durch Drücken der **l**-Taste ausführen.

Erweitern Sie dafür den Parser um folgendes Tupel:

`Scattershot time strength rotation scale x y`
`time`: int - Verbleibende Zeit in ms, bis sich der Schuss aufspalten soll
`strength`: int - Schussstärke
`rotation`: int zwischen 0 und 359 – Ausrichtung des Schusses als Gradzahl
`scale`: int - Skalierung des Schusses (je größer, desto größer der Schuss)
`x`: int - x-Position des Mittelpunkts (in Pixeln)
`y`: int - y-Position des Mittelpunkts (in Pixeln)

Hinweis: Ein Streuschuss wird in Kombination mit der beschränkten Munition genauso wie ein normaler Schuss gehandhabt.

Abwehrgeschütze - 3 Punkte Führen Sie ein Abwehrgeschütz ein, welches wie ein Panzer auf den Spieler schießt, sich aber nur drehen und nicht seine Position verändern kann. Es sei ausdrücklich erwähnt, dass Spieler gewonnen haben, wenn alle gegnerischen Panzer zerstört sind. Es dürfen also Abwehrgeschütze übrig bleiben.

Der Parser muss damit um folgendes Tupel erweitert werden:

`Tower maxlife life maxshot shot strength speed rotation scale x y`
`maxlife`: int - Maximal mögliche Lebenspunkte

`life`: int - Aktuelle Lebenspunkte
`maxshot`: int - Maximal mögliche Schüsse
`shot`: int - Aktuell verbleibende Schüsse
`strength`: int - Schussstärke
`speed`: int - Geschwindigkeit der Drehung
`rotation`: int zwischen 0 und 359 – Ausrichtung des Abwehrgeschützes als Gradzahl
`scale`: int - Skalierung des Abwehrgeschützes (je größer, desto größer das Abwehrgeschütz)
`x`: int - x-Position des Mittelpunkts (in Pixeln)
`y`: int - y-Position des Mittelpunkts (in Pixeln)

Pickups - 4 Punkte Führen Sie zwei Arten von Pickups ein, eines zur Regenerierung von Lebenspunkten und eines zum Nachladen von Munition. Diese erscheinen zufällig auf dem Spielfeld und werden bei Berührung/Kollision mit einem Panzer eingesetzt. Anschließend sind diese nicht länger verfügbar. Weiterhin sind diese nur für eine zufällige Zeiteinheit (mindestens 5 Sekunden) verfügbar und werden 2 Sekunden vor Verschwinden alle 100 ms von sichtbar auf unsichtbar bzw. umgekehrt gestellt.

Der Parser muss damit um folgendes Tupel erweitert werden:

`Pickup type strength rotation scale x y`
`type`: {Healthpack,Ammopack} - Gesundheits- oder Munitionspack
`strength`: int - Regenerationsstärke
`rotation`: int zwischen 0 und 359 – Ausrichtung des Pickups als Gradzahl
`scale`: int - Skalierung des Pickups (je größer, desto größer das Pickup)
`x`: int - x-Position des Mittelpunkts (in Pixeln)
`y`: int - y-Position des Mittelpunkts (in Pixeln)

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 `TanksTestsuiteExtended3`.

Öffentliche Tests der Ausbaustufe 3 sind erfolgreich - 0 Punkte Die öffentlichen Testfälle der Klasse `TanksTestsuiteExtended3` werden fehlerfrei absolviert. Beachten Sie, dass Sie für den Ablauf der Tests die Klasse `TanksTestAdapterExtended3` implementieren müssen!

Intelligenter Gegner - 7 Punkte Implementieren Sie eine intelligentere Steuerung (KI) für den Computergegner. Dieser kennt die Position des Spielers und schießt auf ihn, wenn der Schuss den gegnerischen Spieler erreichen kann. Weiterhin verfolgt die KI den Gegner. Parametrisieren Sie den Computergegner, so dass sich die Schwierigkeit des Gegners über das Hauptmenü in einem eigenen Fenster **Einstellungen** einstellen lässt.

Mehrspielermodus mit einer Tastatur - 3 Punkte Ermöglichen Sie das Spiel zwischen zwei menschlichen Spielern über eine Tastatur. Zusätzlich zu den Pfeiltasten soll der zweite Spieler seinen

Panzer über W,A,S,D navigieren können. Der zweite Spieler kann über Drücken der **g**-Taste schießen. Die Streubombe wird über die **h**-Taste ausgelöst. Minen legen geht mit der **f**-Taste.

Hinweis: Der Mehrspielermodus erfolgt auf dem selben Computer. Ein Netzwerkmehrspielermodus würde extra bepunktet werden. Eventuelle Computergegner müssen sich nicht auf beide Spieler konzentrieren. Deaktivieren Sie außerdem die Abfrage des Namens im Mehrspielermodus für den Highscore. Ändern Sie weiterhin das Dialogfenster so, dass dem Gewinner gratuliert wird (z.B. "Herzlichen Glückwunsch Spieler 1, Sie haben gewonnen. Spieler 2, viel Glück im nächsten Spiel!").

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 - 1 Punkt Audio-Clips werden bei bestimmten Ereignissen abgespielt (z. B. einer gewonnenen Karte).

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

Map-Editor - 10 Punkte Implementieren Sie einen Map-Editor, welcher im Hauptmenü aufgerufen werden kann. Dabei soll eine Karte mit parametrisierbarer Höhe und Breite angelegt werden, auf der alle von Ihnen implementierten Spielobjekte gesetzt werden können. Achten Sie darauf, dass der Map-Editor nur regelkonforme Setzoperationen erlaubt. Es soll zum Beispiel nicht möglich sein, einen Panzer auf eine Wand zu setzen.

Der Spieler soll weiterhin auswählen können, ob die Map gespeichert werden soll oder direkt gespielt.

Laserpointer - 8 Punkte Zum Zielen ist es praktisch, wenn man sehen kann, welches Objekt vom Schuss getroffen würde. Zeichnen Sie eine rote dünne Linie (Laserpointerstrahl), die bis zum nächsten Hindernis reicht.

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*.

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. Zum Testen darf nicht der *org.newdawn.slick.AppGameContainer* werden, da dieser zur Anzeige von Dialogfenstern führt. Verwenden Sie stattdessen *de.tu_darmstadt.gdi1.tanks.tests.TestAppGameContainer*.

Bitte achten Sie bei den Erweiterungen darauf, dass alte Funktionalität nicht verloren geht.

4 Dateiformat

Tanks verwendet ein einfaches Textformat zur Definition von Spielkarten. Jede Karte wird in einer eigenen Datei mit der Endung `*.tanks` gespeichert.

4.1 Tupel für jedes Spielobjekt

Die Datei enthält durch Zeilenumbrüche getrennte Tupel der folgenden Form:
`Identifier parameter1 parameter2 ...`

Weiterhin seien folgende Objekte definiert:

`Map` **`background actualMap nextMap maxDuration elapsedTime shots`**
`background`: String - Der Name der Bilddatei der aktuellen Karte (in Anführungszeichen) mit Angabe des Formats. Wir empfehlen das PNG-Format.
`actualmap`: String - Name der aktuellen Karte in Anführungszeichen ohne Dateiendung
`nextmap`: String - Name der nächsten Karte ohne Dateiendung; wenn `null`, dann soll ins Hauptmenü zurück gewechselt werden.
`maxduration`: int - Anzahl in Millisekunden, die das Spiel dauern darf; bei `0` ohne Begrenzung
`elapsedTime`: int - Vergangene Zeit in Millisekunden
`shots`: int - Verbrauchte Schüsse

`Tank` **`name maxlife life maxshot shot maxmine mine strength speed rotation scale x y`**
`name`: String - Spielername in Anführungszeichen. `"Player1"` steht dabei für den (ersten) menschlichen Spieler.
`maxlife`: int - Maximal mögliche Lebenspunkte
`life`: int - Aktuelle Lebenspunkte
`maxshot`: int - Maximal mögliche Schüsse
`shot`: int - Aktuell verbleibende Schüsse
`maxmine`: int - Maximal mögliche Minen
`mine`: int - Aktuell verbleibende Minen
`strength`: int - Schussstärke
`speed`: int - Geschwindigkeit der Drehung
`rotation`: int zwischen 0 und 359 – Ausrichtung des Panzers als Gradzahl
`scale`: int - Skalierung des Panzers (je größer, desto größer der Panzer)
`x`: int - x-Position des Mittelpunkts (in Pixeln)
`y`: ints - y-Position des Mittelpunkts (in Pixeln)

Eine unzerstörbare Wand (Border) wird durch ihren Mittelpunkt sowie Höhe und Breite definiert. Außerdem ist sie unsichtbar.

Border **x0 y0 width height**

`x0`: int - x-Position (Mittelpunkt) (in Pixeln)
`y0`: int - y-Position (Mittelpunkt) (in Pixeln)
`width`: int - Breite der Wand (in Pixeln)
`height`: int - Höhe der Wand (in Pixeln)

Wall **maxlife life rotation scale x y**

`maxlife`: int - Maximal mögliche Lebenspunkte der Wand
`life`: int - Aktuelle Lebenspunkte
`rotation`: int zwischen 0 und 359 – Ausrichtung der Wand als Gradzahl
`scale`: int - Skalierung der Wand (je größer, desto größer die Wand)
`x`: int - x-Position des Mittelpunkts (in Pixeln)
`y`: int - y-Position des Mittelpunkts (in Pixeln)

Shot **strength rotation scale x y**

`strength`: int - Schussstärke
`rotation`: int zwischen 0 und 359 – Blickrichtung des Schusses als Gradzahl
`scale`: int - Skalierung des Schusses (je größer, desto größer der Schuss)
`x`: int - x-Position des Mittelpunkts (in Pixeln)
`y`: int - y-Position des Mittelpunkts (in Pixeln)

Explosion **width height speed x y**

`width`: int - Breite in Pixeln
`height`: int - Höhe in Pixeln
`speed`: int - Dauer der Anzeige eines Explosionsbilds * 100 ms
`x`: int - x-Position des Mittelpunkts (in Pixeln)
`y`: int - y-Position des Mittelpunkts (in Pixeln)

Es sei ausdrücklich darauf hingewiesen, dass freie Bereiche nicht explizit gespeichert werden.

4.2 Beispiel

Im folgenden sehen Sie ein Beispiel für eine gültige Karte. Diese Karte ist durch unbesiegbare Wände begrenzt, enthält einen Gegner und den menschlichen Spieler sowie eine Wand. Außerdem kann die Karte unbegrenzt lange gespielt werden. Nach Gewinnen diesen Karte würde die Datei `map01.tanks` geladen.

```
Map "/assets/background.png" "map00" "map01" 0 0 0
Border 400 0 800 0
Border 0 300 0 600
Border 400 600 800 0
Border 800 300 0 600
Tank "Player1" 1000 1000 10 10 3 3 30 0 10 300 200
Tank "Opponent1" 30 30 1 1 0 0 1 270 10 100 200
Wall 30 30 0 10 25 25
```

4.3 Erweiterung des Speicherformats

Selbstverständlich darf das bestehende Speicherformat angepasst werden. In Ausbaustufe 1 und 2 wird von Ihnen erwartet, neue Objekte hinzuzufügen. Sie dürfen sogar die Anzahl an Parametern in bestehenden Objekten erweitern, wenn weiterhin auch die Karten der Testfälle (mit weniger Parametern) gelesen werden können. Dies wird von Ihnen aber nicht verlangt und würde extra bepunktet 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 Karten und die öffentlichen Testfälle. Es wird Ihnen dringend nahe gelegt, auch unser bereitgestelltes Framework nutzen. Das Framework ist threadsicher!

Hinweis: Sie dürfen auch eine vollständig eigene Lösungen implementieren. Der dadurch entstehende Mehraufwand wird aber nicht punktuell gewürdigt. Ihre Lösung muss in jedem Fall mit den bereitgestellten Testklassen überprüfbar sein.

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.

5.1 Klasse *de.tu_darmstadt.gdi1.tanks.ui.Tanks*

Tanks erbt von *org.newdawn.slick.state.StateBasedGame*. *Tanks* dient sowohl als Container für Spiele als auch zum Starten des gesamten Spiels. Sie müssen diese Klasse erweitern, um zusätzliche States hinzuzufügen. Lesen Sie sich dazu das *Drop of Water Tutorial* durch.

5.2 Exceptions

Das Framework verwendet zwei Exceptionklassen, die Sie im Rahmen Ihrer eigenen Implementierung verwenden sollen. Hierbei handelt es sich um:

SyntaxException Diese Exception wird geworfen, wenn die Syntax der Karte nicht korrekt ist.

SemanticException Diese Exception wird geworfen, wenn die Semantik der Karte nicht korrekt ist. Anders ausgedrückt heißt das, die Karte gibt so keinen Sinn.

5.3 Die ersten Schritte

Sie sollten sich zunächst das *Drop of Water Tutorial* durchlesen, um das Konzept des Frameworks zu verstehen. Anschließend können Sie in `Tanks` zusätzliche States (Fenster) definieren. Sie sollten mit einem Hauptmenü und einem Spielfenster beginnen.

Anschließend sollten Sie in der Gruppe diskutieren, wie Ihr Projekt strukturiert werden soll und Aufgaben in der Gruppe verteilen.

5.4 Basisereignisse und Basisaktionen

Das **eea**-Framework arbeiten mit **Entitäten**, **Kompontenten** wie der **ImageRenderComponent** und **Events** sowie zugehörigen **Actions**. Die zwei Tabellen am Ende geben einen Überblick über die mitgelieferten Events und Actions. Ein Neuzeichnen eines Fensters geschieht zusammen mit einem Frame.

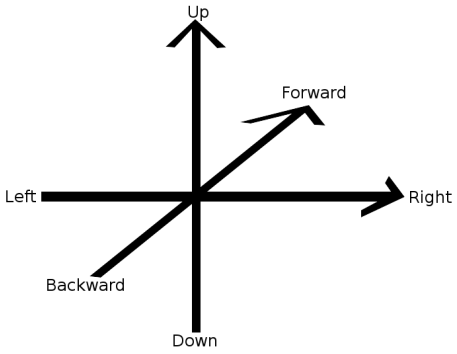

Abbildung 3: Es gibt zahlreiche Aktionen, die eine Bewegung ausdrücken.

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 23 gezeigt aussehen. Eine denkbare Gesamteinteilung für alle Ausbaustufen wird in Tabelle 6 auf Seite 23 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 *SVN* oder *Git* 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

Konstruktor	**Beschreibung**
CollisionEvent()	Dieses Event wird (mit jedem Frame) ausgelöst, wenn sich zwei Entitäten überlappen.
KeyPressedEvent(String id, Integer... key)	Dieses Event wird ausgelöst, wenn die Taste gerade erstmalig nach unten gedrückt wurde.
KeyDownEvent(String id, Integer... key)	Dieses Event wird ausgelöst, wenn die Taste aktuell unten gehalten wird.
LeavingScreenEvent()	Dieses Event wird ausgelöst, wenn die an diesem Event interessierte Entität aus dem Bildschirm heraus gefahren ist.
LoopEvent(String id)	Dieses Event wird mit jedem Frame ausgelöst. Dieses Event eignet sich also dafür, in jedem Frame eine gewisse Aktion auszuführen.
MouseClickedEvent()	Dieses Event wird ausgelöst, wenn die Maus auf die an dem Event interessierte Entität geklickt hat.
MouseEnteredEvent()	Dieses Event wird ausgelöst, wenn die Maus über die an dem Event interessierte Entität fährt.
MovementDoesntCollideEvent(float speed, IMovement move)	Dieses Event wird ausgelöst, wenn die Bewegung (Action) keine Kollision mit einer anderen Entität verursacht.

Tabelle 1: Basisereignisse des **eea**-Frameworks aus dem package *eea.engine.events.basicevents*

Konstruktor	**Beschreibung**
ChangeStateAction(int state)	Diese Action wechselt in einen Zustand (State) mit der State-ID **state**. Ist der State schon einmal initialisiert worden, so wird die **init**-Methode nicht erneut aufgerufen.
ChangeStateInitAction(int state)	Diese Action wechselt in einen State mit der State-ID state. Auch wenn der State schon einmal initialisiert worden ist, so wird die **init**-Methode erneut aufgerufen.
DestroyEntityAction()	Diese Action zerstört die Entität bei Eintreten eines gewissen Events.
MoveBackwardAction(float speed)	Diese Action führt mit einer Geschwindigkeit **speed** eine Rückwärtsbewegung entgegen der Blickrichtung aus.
*MoveDownAction(float speed()**	Diese Action führt mit einer Geschwindigkeit **speed** eine Bewegung nach unten aus.
MoveForwardAction(float speed)	Diese Action führt mit einer Geschwindigkeit **speed** eine Vorwärtsbewegung in Blickrichtung aus.
*MoveLeftAction(float speed)**	Diese Action führt mit einer Geschwindigkeit **speed** eine Bewegung nach links aus.
*MoveRightAction(float speed)**	Diese Action führt mit einer Geschwindigkeit **speed** eine Bewegung nach rechts aus.
*MoveUpAction(float speed)**	Diese Action führt mit einer Geschwindigkeit **speed** eine Bewegung nach oben aus.
RotateLeftAction(float speed)	Diese Action führt mit einer Geschwindigkeit **speed** eine Drehung nach links aus.
RotateRightAction(float speed)	Diese Action führt mit einer Geschwindigkeit **speed** eine Drehung nach rechts aus.
QuitAction()	Diese Action beendet das laufende SlickGame.
RemoveEventAction()	Diese Action zerstört ein Event.
SetEntityPositionAction(Vector2f pos)	Diese Action setzt die Position einer Entität neu.

Tabelle 2: Basisaktionen des **eea**-Frameworks aus dem package *eea.engine.actions.basicactions.* Die mit einem * markierten Aktionen sind für das Spiel Tanks nicht von Bedeutung, denn Panzer „springen“ nicht. Die Richtungen werden verdeutlicht durch Abbildung 3 auf Seite 18.

Aspekt	**Aufwand (ca.)**
Einarbeitung in die Aufgabenstellung und Vorlagen, Lesen des Tutorials	2 Tage
Einigung auf die grundsätzliche Klassen- und Packagehierarchie	0.25 Tage
Implementierung und Test des Parsers	1,5 Tage
Check auf semantische Korrektheit	0.25 Tage
Implementierung von *toString()*	0.25 Tage
Entwicklung der grafischen Benutzerschnittstelle (Spielfenster und Menüfenster)	0.25 Tag
Umsetzung der grafischen Objekte	0.5 Tage
Korrekte Kartendarstellung	0.10 Tage
Steuerung über Tastatur und Maus	0.5 Tage
Schussverhalten	0.5 Tage
Keine Bewegung durch Hindernisse	0.25 Tage
Computergegener umsetzen	1 Tag
Erkennen gewonnener und verlorener Karten	0.25 Tage
Neues Spiel per Tastendruck	0.10 Tage
Durchführen der öffentlichen Tests und Debugging	1 Tage

Tabelle 3: 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	3 - 4 Tage
Ausbaustufe I	2 - 3 Tage
Ausbaustufe II	2 - 3 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 4: Zeitplanung für die Bearbeitung aller Ausbaustufen mit 4 Personen