Projekt zu GdI 1 - Wintersemester 2021/22

Guido Rößling, Kris Früwein, Elmar Shaksalykov

27. Februar 2022

Version 1.0.1

Inhaltsverzeichnis

1 Einführung in das Projekt

Im Rahmen des Projekts implementieren die Student_innen[1] in Gruppen von jeweils vier Personen eine Java-Version des Spiels *Mario vs. Donkey Kong: Aufruhr im Miniland!*. 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 *Mario vs. Donkey Kong: Aufruhr im Miniland!*

Im Spiel *Mario vs. Donkey Kong: Aufruhr im Miniland!* wird gegen den Computer gespielt. Dabei steuern Sie eine oder mehrere Spielfiguren in Form von Mario-Aufziehrobotern über verschiedene Hindernisse zu einer Tür, um das nächste Level zu erreichen.
Die Steuerung erfolgt grundsätzlich über die Maus. Klickt man auf einen Mario-Aufziehroboter (kurz: Mario), ßieht man ihn aufünd er rennt sofort los. Ist ein Mario einmal aufgezogen, kann der Spieler mit ihm nicht mehr interagieren und der Mario hört nicht auf zu laufen. Sogar Wände halten ihn nicht auf, läuft ein Mario gegen eine Wand, dreht er einfach um und läuft in die entgegengesetzte Richtung weiter. Um zu verhindern, das ein aufgezogener Mario blind in sein Verderben läuft (z.B. in Form eines Abgrunds), kann der Spieler für ihn Stahlträger bauen. Diese können dem Mario als Brücke dienen oder schlichtweg als Wand, um Gegner fernzuhalten.
Stahlträger können nur zwischen Sockeln gebaut werden. Klickt der Spieler auf zwei unterschiedliche Sockel, so wird ein Stahlträger zwischen ihnen gebaut, auf dem der Mario laufen kann. Einen Stahlträger kostet jedoch (je nach Länge) Ressourcen und der Spieler hat davon nur begrenzt viele pro Level. Das ganze Level komplett zubauen ist also nicht möglich. Hier muss abgewogen werden, wann es sich lohnt einen Träger zu bauen und wann nicht! Hat man sich verbaut, ist dies jedoch kein Problem. Klickt man zwei mal auf den gleichen Sockel, so verschwinden alle angrenzenden Stahlträger und man erhält seine eingesetzten Ressourcen zurück.
Um noch etwas mehr Spannung in das Spiel zu bringen, gibt es zusätzlich noch Gegner, welche die Aufziehroboter zerstören wollen sowie sammelbare Gegenstände, wie z.B. einen Schlüssel, welcher erst aufgesammelt werden muss, bevor man eine verschlossene Tür öffnen kann.

Abbildung 1 auf der nächsten Seite zeigt ein mögliches Menü. Abbildung 2 auf der nächsten Seite 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.

[1] Diese Schreibweise wird durchgängig verwendet, um alle möglichen Personen gleichberechtigt anzusprechen, auch wenn sie für den_die Leser_in vielleicht teilweise irritierend wirkt. Betrachten Sie dies als Reflektionsmöglichkeit! Mehr Informationen unter `http://de.wikipedia.org/wiki/Geschlechtergerechte_Sprache#Sichtbarmachung` bei „Gender Gap“.

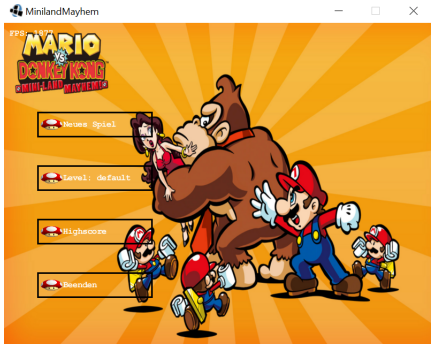

Abbildung 1: Beispiel einer grafischen Umsetzung des Menüs von Mario vs. Donkey Kong: Aufruhr im Miniland!

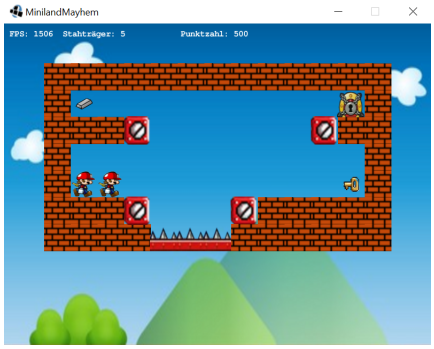

Abbildung 2: Beispiel einer grafischen Umsetzung des Spielfensters von Mario vs. Donkey Kong: Aufruhr im Miniland!

2 Organisation des Projekts

Der offizielle Bearbeitungszeitraum des Projekts beginnt am *Dienstag, den 19.10.2022* und endet am *Montag, den 28.02.2022*. 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 etwa vier Wochen vor Beginn der Projektphase, d.h. am *Montag, den 28.02.2022*, 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 Student_innen 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 2021/22* 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 die jeweiligen Tutor_innen der Gruppe ebenfalls der Projektgruppe angehören werden, sind sie in die Diskussionen eingebunden und können 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 gemeinsam in einer WG erledigen, bringt eine Abstimmung über das im Portal erstellte Projektgruppenforum vermutlich mehr Aufwand als Nutzen.
- Eine Gruppe von Tutor_innen betreut das Projekt. Allen Tutor_innen wird dabei eine gewisse Anzahl Projektgruppen zugeteilt. Die Aufgabe der Tutor_innen ist es, die Gruppe im Rahmen des Projekts bei offenen Fragen zu unterstützen, nicht aber bei der tatsächlichen Implementierung. Insbesondere helfen die Tutor_innen nicht bei der Fehlersuche und geben auch keine Lösungsvorschläge. Die Tutor_innen stehen der Gruppe auch nur zeitlich begrenzt zur Verfügung: pro Gruppe wurden bis zu drei 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.

Wir weisen *ausdrücklich* darauf hin, dass Sie sich **möglichst früh** mit den Tests vertraut machen sollten, um unliebsame Überraschungen „kurz vor Fertigstellung" zu vermeiden!

Die *Abnahme* oder *Testierung* des Projekts (beschrieben in Abschnitt 3) erfolgt durch den_die Tutor_in der Gruppe und erfordert eine *vorherige Terminabsprache*. Bitte bedenken Sie, dass auch unsere Tutor_innen 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_r Tutor_in 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_die Tutor_in) lesbare Form denkbar. Bitte bringen Sie diesen Entwurf zum ersten Treffen mit dem_der Tutor_in mit, damit Sie direkt Feedback erhalten können, ob dieser Ansatz funktionieren kann. Auch hier kann die Nutzung des Portals mit dem Projektgruppenforum helfen, den_die Tutor_in „früher zu erreichen".

3 Ihre Aufgabe

Implementieren Sie eine lauffähige Java-Version des Spiels *Mario vs. Donkey Kong: Aufruhr im Miniland!*, die mindestens der „minimalen Ausbaustufe“ entspricht.

Das fertige Spiel muss von dem_der Tutor_in *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_die Tutor_in startet das Spiel und testet, ob alles so funktioniert wie spezifiziert. Dazu werden potenzielle bestimmte vorgegebene Szenarien durchgespielt, aber auch zufällig „herumgespielt“.

Software-Test Der_die Tutor_in testet die Implementierung mit den für alle Teilnehmer_innen bereitgestellten (öffentlichen) und nur für die Tutor_innen und Mitarbeiter_innen verfügbaren (privaten) JUnit-Tests. Alle Tests müssen **ohne Benutzerinteraktion** abgeschlossen werden können.

Code-Review Der_die Tutor_in sieht sich den Quellcode sowie die Dokumentation an und stellt Fragen dazu.

3.1 Ablauf des Code-Review

Im Hinblick auf den 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_die Tutor_in wird einzelne Gruppenmitglieder seiner Wahl zu Teilen des Quelltexts befragen. Daher sollte sich jedes Gruppenmitglied mit allen Codeteilen auskennen—der_die Tutor_in wählt aus, zu *welchem Thema* eine Frage gestellt wird und 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_die Falsche“ gefragt wird. Durch eine entsprechend bessere Abstimmung in der Gruppe steigen die Lernmöglichkeiten **aller** Gruppenteilnehmer. Auch für (vermeintliche?) „Expert_innen“ 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_der Tutor_in geben, wenn Sie das Feedback lieber anonym geben wollen.

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

- *doxygen* (`https://www.doxygen.nl`) 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* (`https://web.cs.upb.de/archive/fujaba`)
- *BlueJ* (`https://bluej.org/`)

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

Zusätzlich sind mindestens *vier* repräsentative Screenshots Ihres Spiels einzureichen, darunter ein Bild vom Startmenü.

3.3 Hinweise

Denken Sie bitte daran, dass **Testfälle keine Interaktion mit den Spieler_innen erfordern dürfen**.

Sollten Sie sich für die Nutzung des vorgegebenen Frameworks entscheiden, so nutzen Sie das Konzept von Entitäten, Ereignissen und Aktionen. Die Tutor_innen 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. Die Tutor_innen werden bewerten, wie gut diese Trennung bei Ihnen gelungen ist.

3.4 Minimale Ausbaustufe

Um das Projekt 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 von dem_der Tutor_in 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_innen (nicht die Tutor_innen!), 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 `MinilandTestAdapterMinimal` 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 Anzahl an verfügbaren Ressourcen in dem statischen Feld `ressources` der Klasse `GamePlayState` zu speichern, so wäre dies eine passende Implementierung der Methode `getRessources()` der Klasse `MinilandTestAdapterMinimal`:

```
public int getRessources() {
  return GamePlayState.ressources;
}
```

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 `MinilandTestAdapterMinimal`, Build path, exclude).

Öffentliche Tests sind erfolgreich—0 Punkte Die öffentlichen Testfälle der Klasse `MinilandTestsuiteMinimal` 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 `MinilandTestAdapterMinimal` implementieren müssen!

Generell sind die öffentlichen Tests eine Hilfe für Sie. Die Testfälle durch „Schummeln“ erfolgreich zu bestehen, bringt Sie nicht weiter.

Grafische Benutzerschnittstelle—3 Punkte Es gibt eine grafische Benutzerschnittstelle („GUI“), mit der man *Mario vs. Donkey Kong: Aufruhr im Miniland!* spielen kann.

Spielmenü—3 Punkte Das Spiel Mario vs. Donkey Kong: Aufruhr im Miniland! muss in einem Hauptmenü beginnen. Das Menü muss mindestens jeweils die Buttons „Start“, „Level“ und „Beenden“ mit entsprechender Semantik enthalten.
Im Spielmenü soll die aktuelle Anzahl der Ressourcen sichtbar sein. Der Startwert liegt hier bei 5. Zusätzlich soll beim Drücken der **P**-Taste das Spiel pausiert werden, beim nochmaligen Drücken soll das Spiel fortgesetzt werden. Ist Spiel pausiert, soll ein "pausiert"Text angezeigt werden. Wenn die **Escape**-Taste gedrückt wird, dann soll das Spiel sofort beendet werden und in einen Endscreen übergehen.
Im Endscreen sollen die drei Buttons „Neustart“, „Hauptmenü“ und „Beenden“ vorhanden sein mit jeweiliger Funktion.

Grafische Umsetzung der Objekte—2 Punkte Für jedes Spielobjekt und jeden Spielzustand existiert eine passende grafische Umsetzung gemäß der Spielbeschreibung. Die folgenden Objekte sind in der Minimalstufe enthalten: Mario-Aufziehroboter, Wände, Stahlträgersockel, Stahlträger, Türen und Gefahren (in Form von Stacheln).

Hinweis: die Bilder der Vorlage gehen dabei von einer Größe von 50 LE x 50 LE pro Spielobjekt aus (LE = Längeneinheit). Sollten Sie eigene Bilder verwenden, achten Sie darauf, dass die Größenverhältnisse der Spielobjekte zueinander passen.

Levelgenerator—8 Punkte Wird im Hauptmenü auf den Button „Level“ geklickt, so soll der Spieler eine Datei auswählen können, welche eingelesen werden soll. Anschließend soll das eingelesene Level intern gespeichert werden. Dabei gibt es folgende Übersetzung von Zeichen zu Objekt:

- M : Mario-Aufziehroboter
- W : Wand
- D : Tür
- S : Stahlträger-Sockel
- X : Gefahr (Stacheln)
- _ : nichts (leeres Feld)

Zusätzlich soll jedes unbekannte Zeichen, welches hier nicht aufgelistet ist, als nichts bzw. leeres Feld interpretiert werden. Wichtiger Hinweis: das Schema wird in allen Ausbaustufen erweitert um neue Objekte mit neuen Zeichen.
Es soll auch sichergestellt sein, dass das eingelesene Level rechteckig ist. Das bedeutet, dass jede Zeile in der einzulesenden Textdatei gleich viele Zeichen besitzt. Besteht eine Zeile aus "WWW" und die nächste aus "WWWW" so ist dies nicht erlaubt und die entsprechende Datei soll nicht gelesen werden. Beispieldateien befinden sich in der Vorlage im Ordner „level“.

Neues Level starten—3 Punkte Wurde ein Level ausgewählt und erfolgreich geparst, so soll ein Klick auf den Button „Start“ im Hauptmenü ein neues Spiel starten. Dabei wird das zuvor gespeicherte Level geladen. Die Spielobjekte (Entities) befinden sich dann an den entsprechenden Positionen wie in der Textdatei. Aušerdem soll es noch eine Bounding-Box aus Wand-Objekten geben, welche das eigentliche Level komplett umrandet.
Wurde das Level nicht erfolgreich geparst (oder keines ausgewählt), so soll bei Klick auf

„Start“ ein (nicht leeres!) default Level erstellt und angezeigt werden. Wie dieses default Level aussehen soll wird nicht weiter spezifiziert. Lassen Sie Ihrer Kreativität freien Lauf!

Verhalten des Marios—6 Punkte Mario-Aufziehroboter starten das Spiel inaktiv. Wird ein Mario mit der Maus angeklickt, so wird er aufgezogen und läuft los. Seine Laufrichtung ist anfangs immer rechts und seine Geschwindigkeit ist 0,1 LE/Frame. Sollten sie andere Größen für Ihre Spielobjekte als 50 LE x 50 LE haben, müssen Sie auch seine Geschwindigkeit anpassen. Sorgen Sie dafür, dass ein Mario nach 500 Frames Distanz zurückgelegt hat in Höhe von genau einer Spielobjektbreite.

Ist ein Mario aufgezogen, so kann der Spieler nicht mehr mit ihm interagieren. Läuft er frontal gegen eine Wand, einen Sockel oder einen anderen Mario so dreht der Mario um und läuft von der Wand weg. (Bei einem anderen Mario dreht der andere Mario auch um!)
Berührt ein Mario eine Gefahr in irgendeiner Weise, so wird er sofort vom Spiel entfernt. Bei einer Tür ist es (vorerst) identisch: kollidiert ein Mario irgendwie mit einer Tür, so wird er auch vom Spiel entfernt.

Spielende—6 Punkte Wurden alle Marios aus dem Spiel entfernt, so wird das Spiel beendet und der Spieler gelangt in den Endscreen (wie bei Druck auf die **ESC**-Taste). Zusätzlich wird dem Spieler angezeigt, wie viele von wie vielen Marios erfolgreich durch eine Tür gelaufen sind und nicht durch eine Gefahr zerstört wurden. Ein Beispiel: „1 von 2 Marios haben es geschafft“.

Stahlträger bauen und entfernen—5 Punkte Bis jetzt kann der Spieler nur Marios aufziehen und sonst nichts tun. Das wird nun geändert. Der Spieler soll in der Lage sein, zwischen zwei Stahlträger-Sockeln einen Stahlträger zu bauen. Dazu klickt er zuerst auf einen der zwei Sockel und danach auf den anderen. Dann erscheint sofort ein Stahlträger, welcher beide Sockel in einer geraden Linie verbindet. Außerdem sollen dem Spieler Ressourcen abgezogen werden entsprechend der (euklidischen) Länge des Stahlträgers. Dabei wird eine Ressource pro 50 Längeneinheiten (aufgerundet) des Trägers abgezogen. Hat der Spieler nicht genug Ressourcen, um einen Stahlträger zu bauen (nach dem Bau w"ären die Ressourcen negativ), so soll der Träger nicht gebaut werden.
Klickt der Spieler zwei mal hintereinander auf den gleichen Sockel, so werden **alle** an diesen Träger angebauten Stahlträger entfernt und die jeweiligen Kosten werden zurückerstattet.

Stahlträger Hoch- und Runterlaufen—8 Punkte Marios interagieren auf folgende Weise mit gebauten Stahlträgern:

- Stahlträger mit einer Steigung von über 44 Grad sind Für die Marios zu steil und können nicht erklommen werden. Trifft ein Mario auf solch einen Träger, so dreht er um, wie als wäre er gegen eine Wand gelaufen.

- Läuft ein Mario auf einen Stahlträger, welcher eine Steigung von weniger (oder gleich) 44 Grad besitzt, so läuft der Mario diesen Stahlträger hoch bzw. runter. Dabei läuft er parallel zum Stahlträger, bis er das Ende erreicht hat. Dann läuft er wieder normal (parallel zum Boden).

- Während ein Mario einen Träger hoch bzw. runterläuft, ignoriert er außerdem alle Träger, welche flach genug sind, um sie hoch- bzw. runterzulaufen und ändert seinen Kurs **nicht**.

- Kollidiert ein Mario mit etwas, was ihn seine Richtung ändern lässt, während er gerade einen Stahlträger hoch- oder runterläuft, so ändert er seine Richtung und läuft runter, wenn er davor hochgelaufen ist und umgekehrt.

- läuft ein Mario von der falschen Seite gegen einen Träger (und ist unter ihm), so zählt dieser auch als Wand und lässt den Mario umkehren, auch wenn der Stahlträger flach genug ist, um ihn hochzulaufen. Eine Veranschaulichung befindet sich in untenstehender 3. Außnahme dieser Regel: Der Mario läuft gerade einen anderen Träger hoch oder runter. Dann behält der Mario seinen aktuellen Kurs bei.

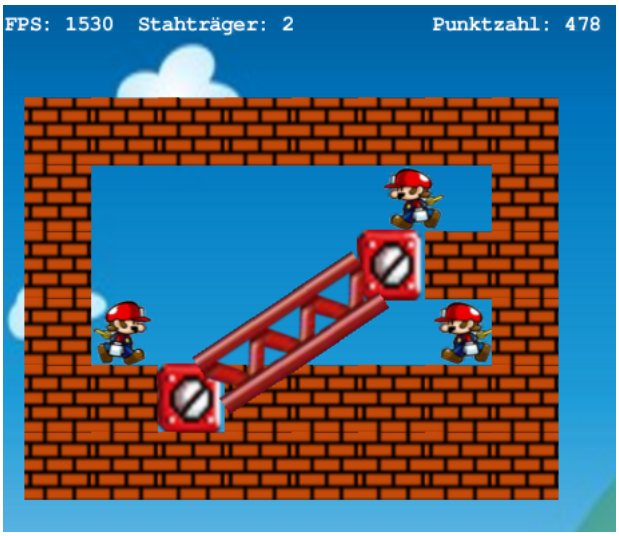

Abbildung 3: Der Mario ganz links sowie der obere Mario können den Stahlträger hoch- und runterlaufen. Der Mario unten rechts jedoch wird von ihm eingeschlossen.

Realistische Schwerkraft—6 Punkte Befindet sich zu einem Zeitpunkt **kein** Boden unter einem Mario, so fällt er. Seine Fallbewegung gleicht dabei einer gleichmäßig beschleunigten Bewegung mit Beschleunigung $a = 0.981 L\ddot{a}ngeneinheiten/s^2$. Dies entspricht einem Zehntel der tatsächlichen Erdbeschleunigung (die normale Erdbeschläunigung wäre zu schnell und der Spieler könnte kaum reagieren). Mit dem `delta` Parameter der Update Methode, können sie auf die vergangene Zeit in Millisekunden zwischen Aufrufen der Update Methode zugreifen.
Landet der Mario auf einem Boden, so hört er sofort auf zu fallen und verliert jegliche Bewegungsgeschwindigkeit nach unten. Als Boden zählen die Spielobjekte Wand, gebauter Stahlträger und Stahlträger-Sockel. Sollte der Mario auf einen Stahlträger mit passender Steigung falle, so beendet er seine Fallbewegung und läuft den Träger hoch bzw. runter.

Achten Sie auch darauf, dass der Mario sich während dem Fallen weiterhin nach links oder rechts bewegt!

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 `MinilandTestsuiteExtended1`.

Öffentliche Tests der Ausbaustufe 1 sind erfolgreich—0 Punkte Die öffentlichen Testfälle der Klasse `MinilandTestsuiteExtended1` werden fehlerfrei absolviert. Beachten Sie, dass Sie für den Ablauf der Tests die Klasse `MinilandTestAdapterExtended1` 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, Ereignissen und Aktionen. Verwenden Sie Packages zur Strukturierung Ihres Codes. Ihr_e Tutor_in 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 T20.42-47. 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_e Tutor_in wird sich das Klassendiagramm ansehen und ggf. Fragen hierzu stellen. Insbesondere beim Code-Review wird auf die Umsetzung des MVC-Prinzips geachtet.

Sammelbare Gegenstände—3 Punkte Es soll sammelbare Gegenstände geben. Läuft ein Mario über einen solchen, so wird der Gegenstand vom Spiel entfernt und seine Wirkung tritt in Kraft. Unterschiedliche Gegenstände haben unterschiedliche Effekte.

Fügen Sie dem Spiel einen sammelbaren Ressourcen-Gegenstand hinzu. Wird er aufgesammelt, so erhält der Spieler 3 zusätzliche Ressourcen, um Stahlträger zu bauen. Erweitern Sie das Übersetzungsschema des Levelgenerators um das Zeichen "R". Ein eingelesenes R soll als ein sammelbares Ressourcen-Spielobjekt übersetzt werden.

Verschlossene Türen—7 Punkte Neben den normalen, offenen Türen soll es nun auch verschlossene Türen geben. Läuft ein Mario gegen solch eine verschlossene Tür, so geht er nicht hindurch, sondern prallt ab und ändert seine Laufrichtung, wie als wäre er gegen eine Wand gelaufen. Um eine Verschlossene Tür zu aufzuschließen, muss zuerst ein Schlüssel (in Form eines neuen sammelbaren Gegenstandes) eingesammelt werden. Hat ein Mario dies getan und berührt er nun eine verschlossene Tür, so wird diese zu einer normalen Tür und der Mario wird vom Spiel entfernt. Auch andere Marios können nun durch die aufgeschlossene Tür gehen, die verschlossene Tür verhält sich dann identisch zu einer normalen Tür.

Beachten Sie, dass nur der Mario, welcher auch den Schlüssel eingesammelt hat, auch derjenige ist, welcher die Tür öffnet und kein anderer! Ausnahme: Der andere Mario hat einen

anderen Schlüssel eingesammelt. Jeder Shlüssel passt in jede verschlossene Tür.
Die Spielobjekte Schlüssel und verschlossene Tür sollen auch in den Übersetzungsschema des Levelgenerators eingefügt werden. Dabei soll das Zeichen "K"(eng. Key) einen Schlüssel erstellen und ein kleines "d" eine verschlossene Tür.

Kugelwilli und Kanonen—10 Punkte Erstellen Sie zwei neue Spielobjekte: Einen Kugelwilli und eine Kanone. Kugelwillis sind Raketen, welche durch die Karte fliegen. Sie sind von der Schwerkraft unbeeinflusst und ändern ihre Flugrichtung nie. Kollidiert ein Kugelwilli mit einem beliebigen anderen Spielobjekt, so wird der Kugelwilli zerstört. Handelt es sich bei dem anderen Spielobjekt um einen Mario, so wird zusätzlich auch der Mario zerstört.

Kanonen lassen alle 5 Sekunden einen neuen Kugelwilli erscheinen. Nach den ersten 5 Sekunden wird ein Kugelwilli links von der Kanone erschaffen, welcher nach links fliegt. Nach 5 weiteren Sekunden erscheint ein weiterer rechts von der Kanone und fliegt nach rechts. Der nächste ist wieder links und so weiter. Das Erscheinen von Kugelwillis kann nicht verhindert werden (aber sie können durch Wände bzw. gebaute Stahlträger sofort zertört werden). Das Kanonen-Spielobjekt selbst verhält sich sonst identisch zu einem Wand-Spielobjekt. Marios können auf einer Kanone stehen und drehen um, wenn sie mit ihr kollidieren. Im Levelgenerator hat die Kanone das Zeichen "B"(engl. Bill Blaster). Der Kugelwilli wird nicht hinzugefügt, da er von der Kanone erschaffen wird.

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 `MinilandTestsuiteExtended2`.

Öffentliche Tests der Ausbaustufe 2 sind erfolgreich—0 Punkte Die öffentlichen Testfälle der Klasse `MinilandTestsuiteExtended2` werden fehlerfrei absolviert. Beachten Sie, dass Sie für den Ablauf der Tests die Klasse `MinilandTestAdapterExtended2` implementieren müssen!

Score sichtbar—5 Punkte Während dem Spielen wird die Aktuelle Punktzahl angezeigt. Startet man ein neues Spiel, so startet man mit 500 Punkten. Jede Sekunde verliert der Spieler einen Punkt. Immer, wenn der Spieler einen Stahlräger baut verliert er 10 Punkte unabhängig von der Länge des Stahlträgers. Wird ein Mario zerstört, so werden 500 Punkte abgezogen. Daei spielt es keine Rolle, wie der Mario zerstört wurde. Erreicht ein Mario die Tür und geht hindurch, so erhält der Spieler 500 Punkte.

Sammelbare Münze—5 Punkte Fügen Sie dem Spiel das Spielobjekt Muünze hinzu. Die Münze ist ein sammelbarer Gegenstand, welcher dem Spieler 500 Punkte gutschreibt, wenn sie aufgesammelt wird. Beim Einlesen eines Levels steht das Zzeichen "C"(engl. Coin) für eine Münze.

Highscore speichern—5 Punkte Wird ein Spiel beendet, soll es im Endscreen einen neuen Button geben mit der Aufschrigt „Highscore Speichern“. Wird dieser Knopf gedrückt, so soll eine neue Textdatei im Ordner `highscores` erstellt werden oder die bereits bestehende Datei angepasst werden. Die neue Textdatei soll folgendes Namensschema bestzen: "Highscore_levelname.txt". Ist der Levelname beispielsweise „Level1“, so soll die Highscoredatei „Highscore_Level1.txt“heißen. In die Datei soll nun ein neuer Eintrag geschrieben werden, welcher die erreichte Wertung repräsentiert.

Der Aufbau des Eintrags ist folgender: „p,e/i“, wobei e für die Anzahl Marios steht, die erfolgreich durch eine Tür gelaufen sind, i steht für die Gesamtzahl der Marios des Levels und p steht für die erzielte Punktzahl. Haben es also 3 von 5 Marios durch die Tür geschafft und die resultierende Punktzahl ist 1234, so soll der neue Eintrag „1234,3/5“sein. Eine Zeile in der Textdatei darf nur einen Eintrag enthalten. Außerdem sollen in der Datei nur die fünf besten Einträge zu sehen sein. Ein Eintrag gilt als besser als ein anderer, wenn mehr Punkte erzielt wurden oder bei gleicher Punktzahl mehr Marios durch eine Tür gelaufen sind.

Highscore laden—5 Punkte Im Hauptmenü soll es einen neuen Button mit der Aufschrift „Highscore“geben. Wird dieser Button gedrückt, so soll eine Datei ausgewählt werden. Ist die gewählte keine Textdatei oder ihr Inhalt entspricht nicht dem Schema "p,e/i", so soll nichts geschehen.

Andernfalls sollen die einzelnen Einträge der Datei gelesen werden und der Spieler soll in einen Highscorebildschirm gelangen, welcher ihm die fünf besten Highscores (falls vorhanden) anzeigt. Mittels eines „Zurück“Buttons gelangt der Spieler wieder in das Hauptmenü.

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 `MinilandTestsuiteExtended3`.

Öffentliche Tests der Ausbaustufe 3 sind erfolgreich—0 Punkte Die öffentlichen Testfälle der Klasse `MinilandTestsuiteExtended3` werden fehlerfrei absolviert. Beachten Sie, dass Sie für den Ablauf der Tests die Klasse `MinilandTestAdapterExtended3` implementieren müssen!

Gegner: Feuer—4 Punkte Erstellen Sie einen neuen Gegner: das Feuer. Das Feuer ist ein bewegliches Spielobjekt. Einige Eigenschaften sind anders als die eines Marios und werden näher spezifiziert. Alle anderen Eigenschaften, welche nicht explizit erwähnt wurden sind **identisch** zu denen eines Marios (z.B. Bewegungsverhalten, Zerstörung durch Kugelwilli,...)

- Das Feuer ist von Anfang an aktiv und läuft sofort los. Es muss nicht erst angeklickt werden.
- Das Feuer läuft anfangs nach links und nicht nach rechts wie die Marios.

- Berührt ein Feuer einen Mario, so wird der Mario zerstört (und 500 Punkte werden abgezogen)
- Läuft ein Feuer gegen ein anderes Feuer, so drehen beide um und wechseln die Richtung
- Ein Feuer kann keine sammelbaren Gegenstände aufnehmen.
- Wird ein Feuer zerstört, erhält der Spieler 100 Punkte dafür.
- Läuft ein Feuer einen Stahlträger hoch oder runter, soll es parallel zur Steigung des Trägers gedreht werden. Es schaut also hoch beim Hochlaufen und runter beim Runterlaufen.
- Ein Feuer kann nicht durch eine Tür gehen und prallt an ihr ab wie an einer Wand.
- Das Feuer hat das Zeichen "Fïm Levelgenerator.

Powerup—2 Punkte Das Powerup ist ein sammelbarer Gegenstand. Wird er von einem Mario eingesammelt, so wird dieser dauerhaft (fast) unbesiegbar. Trifft ihn ein Kugelwilli, so wird der Mario nicht mehr zerstört. Auch ein Feuer kann ihm nichts mehr anhaben und bei Kollision wird statdessen das Feuer zerstört. Lediglich Gefahren zerstören den Mario weiterhin.

Ein Mario mit Powerup soll auch graphisch von anderen Marios unterscheidbar sein. In der Vorlage finden Sie neben den Bildern für einen normalen roten Mario auch Bilder für einen gelben Mario. Das Zeichen im Levelgenerator für das Powerup ist "P".

Trampolin—4 Punkte Es soll ein neues Spielobjekt erzeugt werden: das Trampolin. Dieses lässt einen Mario, welcher das Trampolin berührt, einmalig hochspringen. Die Schwerkraft wirkt ab dem Moment, ab dem der Mario nicht mehr auf dem Boden ist. Während des Sprungs bewegt sich ein Mario wie gehabt nach links bzw. rechts. In Kombination mit einem realistischen Fallverhalten entsteht so ein realistisches Sprungverhalten. Die Startgeschwindigkeit v_0 des Marios ist dabei das fünffache seiner normalen Bewegungsgeschwindigkeit.

Berührt ein Mario beim Springen eine Wand bzw. Decke, so hört das Springen sofort auf und falls der Mario noch in der Luft ist, fällt er. Es soll au ßerdem auch dem Feuer möglich sein, zu Springen. Im Übersetzungsschema des Levelgenerators steht "T"für das Trampolin.

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 Tutor_innen und der Veranstalter_innen. 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! :-)

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

Ein paar „Standardideen“ sind etwa *Easter Eggs*, eine Statuskonsole oder fetzige Hintergrundmusik.

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.gorillas.tests.TestAppGameContainer*.

Die Tests benutzen weiterhin nicht die Klasse *de.tu_darmstadt.informatik.fop.gorillas.main.Gorillas* sondern die Klasse *de.tu_darmstadt.gdi1.gorillas.test.setup.TestGorillas*. Diese Klasse erbt statt von *org.slick.StateBasedGame* von *de.tu_darmstadt.informatik.fop.gorillas.test.setup.TWLTestStateBasedGame*. Sie müssen folglich exakt wie Sie in der Klasse Gorillas Ihre States hinzugefügt haben, dies auch in TestGorillas tun. TWLTestStateBasedGame verzichtet im Gegensatz zu TWLStateBasedGame auf das Rendern der TWL-GUI-Elemente.

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

4 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 durch Punkte 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_zur Tutor_in**, um. . .

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

4.1 Klasse *de.tu_darmstadt.informatik.fop.gorillas.main.Gorillas*

Mario vs. Donkey Kong: Aufruhr im Miniland! erbt von *org.slick.StateBasedGame*. *Mario vs. Donkey Kong: Aufruhr im Miniland!* 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.

4.2 Die ersten Schritte

Sie sollten sich zunächst das *Drop of Water Tutorial* durchlesen, um das Konzept des Frameworks zu verstehen. Anschließend können Sie in `Mario vs. Donkey Kong: Aufruhr im Miniland!` 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.

4.3 Basisereignisse und Basisaktionen

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

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 die Spielfläche verlassen hat.
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, 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 Spiel.
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 Mario vs. Donkey Kong: Aufruhr im Miniland! nicht von Bedeutung. Um den Wurf eines Objektes zu simulieren, empfiehlt es sich eine eigene MoveAction zu implementieren. Die Richtungen werden verdeutlicht durch Abbildung 4 auf der nächsten Seite.

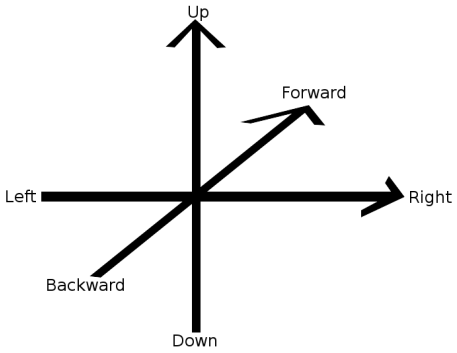

Abbildung 4: Es gibt zahlreiche Aktionen, die eine Bewegung ausdrücken.

5 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 3 auf der nächsten Seite gezeigt aussehen. Eine denkbare Gesamteinteilung für alle Ausbaustufen wird in Tabelle 4 auf der nächsten Seite 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_zur Tutor_in nicht scheuen, ehe Sie sich gleich an die nächste Stufe heranwagen. Fragen Sie Ihre_n Tutor_in nach eventuellen Unklarheiten, falls Sie z. B. inhaltlich die Aufgabenstellung nicht verstanden haben. Es ist Aufgabe der Tutor_innen, 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.

Aspekt	Aufwand (ca.)
Einarbeitung in die Aufgabenstellung und Vorlagen, Lesen des Tutorials	1.5 Tage
Einigung auf die grundsätzliche Klassen- und Packagehierarchie	0.25 Tage
Entwicklung der grafischen Benutzerschnittstelle (Spielfenster und Menüfenster)	0.25 Tag
Neues Spiel per Tastendruck	0.1 Tag
Eingabe und Anzeige der Spielernamen	0.5 Tag
Umsetzung der grafischen Objekte	0.1 Tag
Sprungverhalten	0.5 Tag
Erkennen Spielende: Mario im Tor	0.1 Tag
Erkennen Spielende: Gegner getroffen	0.25 Tag

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	4 - 5 Tage
Ausbaustufe I	3 - 4 Tage
Ausbaustufe II	2 - 3 Tage
Ausbaustufe III	1 - 2 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

6 Liste der Anforderungen

Anforderungen an das Projekt Seite