Projekt zu FOP - Wintersemester 2022/23

Studienarbeit von Tobias Lingenberg, Markus Reuter & Constanze Kramer
Datum: 24. März 2023

Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Projekt zu FOP - Wintersemester 2022/23

Studienarbeit von Tobias Lingenberg, Markus Reuter & Constanze Kramer

Datum: 24. März 2023

Darmstadt

Inhaltsverzeichnis

1 Einführung in das Projekt

Im Rahmen des Projekts implementieren die Student_innen in Gruppen von jeweils vier Personen eine Java-Version des Spiels *Space Apes*. Im Folgenden werden das Spiel und die Aufgabe vorgestellt.

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.

1.1 Das Spiel *Space Apes*

Bist du bereit für ein episches Eins-gegen-Eins-Duell im Weltraum? In *Space Apes* kannst du gegen eine_n andere_n Spieler_in antreten und deine Fähigkeiten im Umgang mit der Schwerkraft unter Beweis stellen!

Jede_r Spieler_in steuert einen Affen auf einem Planeten und versucht, den gegnerischen Affen durch geschicktes Ausnutzen von Gravitationskräften abzuschießen. Das Spiel ist rundenbasiert.
Der Zug eines_einer Spielers_Spielerin besteht aus der Option seine_n Affen auf dem Planeten zu bewegen, dem Einstellen von Schusswinkel und Schussstärke, und endet durch Abschuss des Projektils. Spieler_innen benötigen taktisches Vorgehen bei der Positionierung und müssen einen Kompromiss zwischen Offensive und Defensive finden, da nach Abschuss des Projektils keine Bewegung mehr möglich ist. Eine geschützte Position z. B. auf der Rückseite des Planeten erschwert das eigene Zielen. Die Wahl einer offenen Position, ermöglicht häufig eine bessere Schussbahn. Auch die restliche Umgebung, wie andere Planeten, können zum eigenen Vorteil genutzt werden.

Während des Spiels können Items eingesammelt werden, die sich auf der Flugbahn des Projektils befinden. Diese können eine Regeneration von Lebenspunkten oder Energie bewirken. Außerdem können Münzen eingesammelt werden, die den Kauf von stärkeren Projektilen für den nächsten Zug ermöglichen.
Das Spiel endet, wenn die Lebenspunkte des gegnerischen Affens auf 0 gesunken sind.

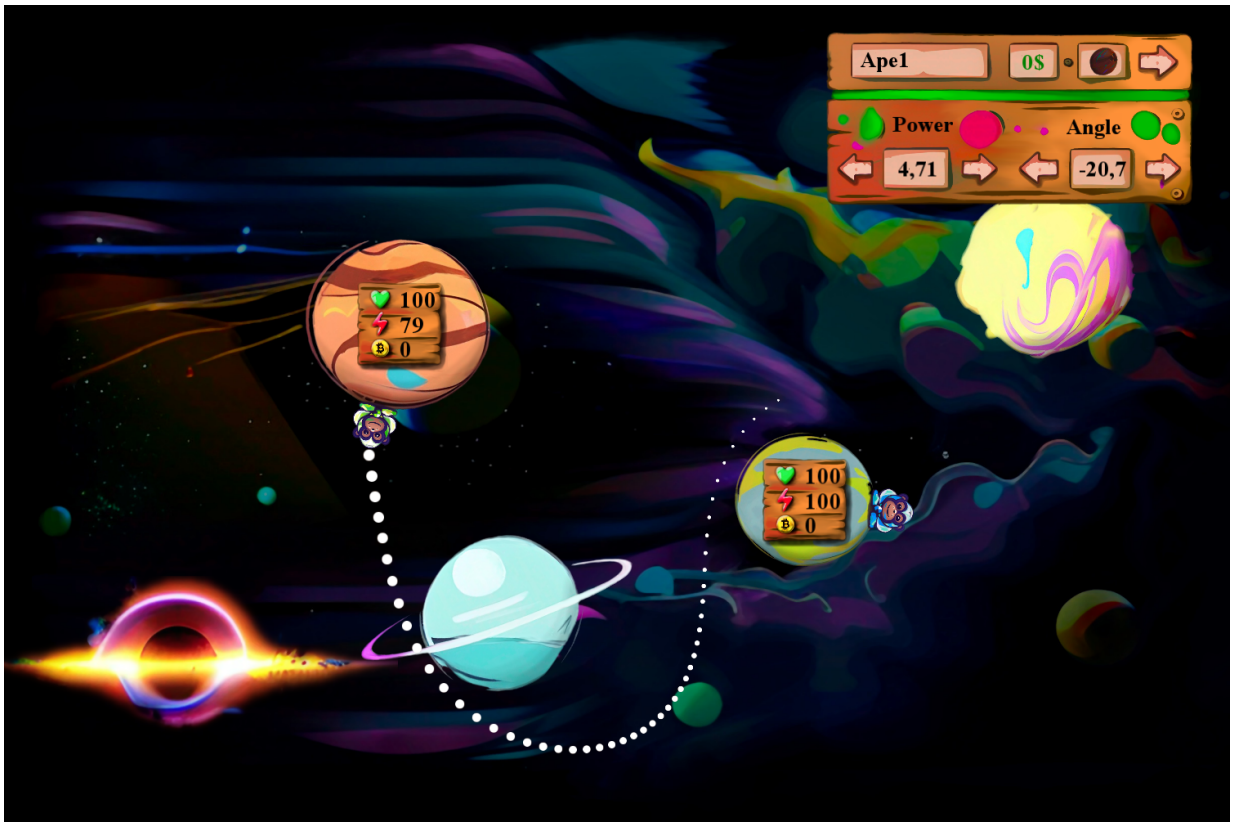


Abbildung 1.1: Screenshot des fertigen Spiels

1.2 Übersicht der wichtigsten Entitäten

Affe Jede_r Spieler_in verfügt über einen Affe, welcher auf einem eigenen Planeten platziert ist. Ist ein_e Spieler_in am Zug, so kann er_sie den jeweiligen Affen auf der Planeten Oberfläche bewegen und die Schuss-Parameter einstellen. Nach Abschuss eines Projektils ist der Zug vorbei und keine Interaktion mehr möglich. Affen unterschiedlicher Parteien sollten farblich visuell unterscheidbar sein. Alle zur Verfügung stehenden Grafiken befinden sich im Unterverzeichnis `SpaceApes/img.apes`. Sinnvolle Attribute eines Affen können sein:

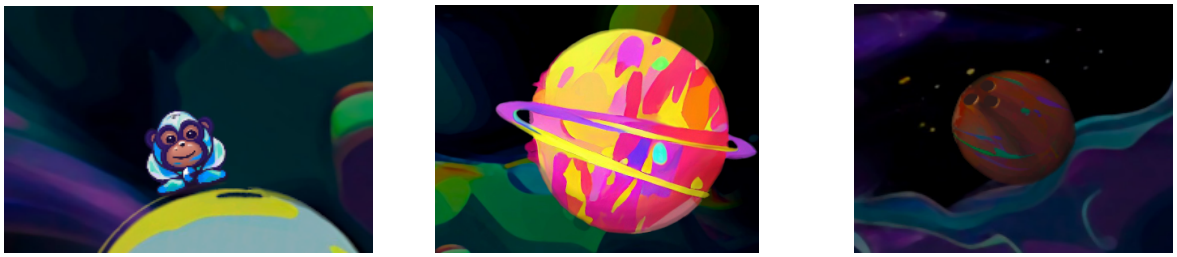
- Heimatplanet, Position auf dem Planet, Absolute Position
- Leben, Energie, Coins
- Schusswinkel, Schussstärke
- Aktivitätszustand

Planet Planeten werden bei Initialisierung an einer fixen Position in der Welt platziert. Sie können von einem Affen bewohnt sein. Planeten sollten immer einen gewissen Mindestabstand zu anderen Planeten besitzen. Verschiedene Grafiken für Planeten finden Sie im Unterverzeichnis `SpaceApes/img.planets`. Sinnvolle Attribute eines Planeten können sein:

- Zugehöriger Affe
- Absolute Position in der Welt
- Radius, Masse

Projektil Tätigt ein_e Spieler_in einen Schuss, so fliegt eine Projektil in Abhängigkeit der Schussparameter los. Die Flugbahn eines Projektils sollte ab Ausbaustufe I [4.2] von der Anziehungskraft der Planeten beeinflusst sein. Ein Projektil fliegt so lange, bis es zu weit außerhalb des sichtbaren Spielbereichs ist, oder es einen Planeten/Affen getroffen hat. Verschiedene Projektil Grafiken finden sich unter `SpaceApes/img.projectiles`, sowie Explosionen unter `SpaceApes/img.explosions`. Sinnvolle Attribute eines Projektils können sein:

- x und y Koordinate, Geschwindigkeit in x und y Richtung
- Ausrichtungswinkel
- Projektiltyp
- Kosten, maximaler Schaden, Explosionsradius



(a) Affe (b) Planet (c) Projektil

Abbildung 1.2: Die drei wichtigsten Entitäten im Spiel

Control Panel Um weitere Spieler_innen-Interaktionen zu ermöglichen, befindet sich in einer Ecke des Fensters ein Controlpanel. Dieses bietet Optionen zum Einstellen des Schusswinkels und der Schussstärke, sowie die Option zur Auswahl von Projektilen. Des weiteren werden aktuelle Werte angezeigt, wie der Name des_der aktiven Spielers_Spielerin, die Kosten des ausgewählten Projektils, sowie die aktuellen Schussparameter. Sobald ein Spielzug vorbei ist verschwindet das Controlpanel und wird wieder sichtbar, sobald der_die nächste Spieler_in am Zug ist. Sinnvolle Attribute des Controlpanels können sein:

- Anzeigeposition
- Liste der zum Erwerb stehenden Projektile

1.3 Pixelkoordinaten und Weltkoordinaten

Um nicht in Pixeldimensionen denken und arbeiten zu müssen, soll ein weiteres Koordinatensystem eingeführt werden: Die Weltkoordinaten. Der Zusammenhang dieser beiden Koordinatensysteme ist in Abbildung 1.3 dargestellt. Die Pixel spannen eine Ebene über 1200×900 auf und die Weltkoordinaten spannen eine Ebene von 16×12. Die Ursprünge sind wie in der Abbildung dargestellt. Wir stellen Ihnen für diverse Umrechnungen

die Klasse `SpaceApes/src/utils/Utils.java` zur Verfügung. Wenn Sie mit einem größerem Fenster (mehr Pixel) oder anderen Weltkoordinaten arbeiten möchten, können Sie dort die entsprechenden Variablen verändern. Beachten Sie jedoch, dass dann auch eine Umrechnung der Werte für die Tests vorgenommen werden muss. In den Tests und in den Aufgabenstellungen werden ausschließlich Weltkoordinaten beutzt, wie sie hier definiert sind.

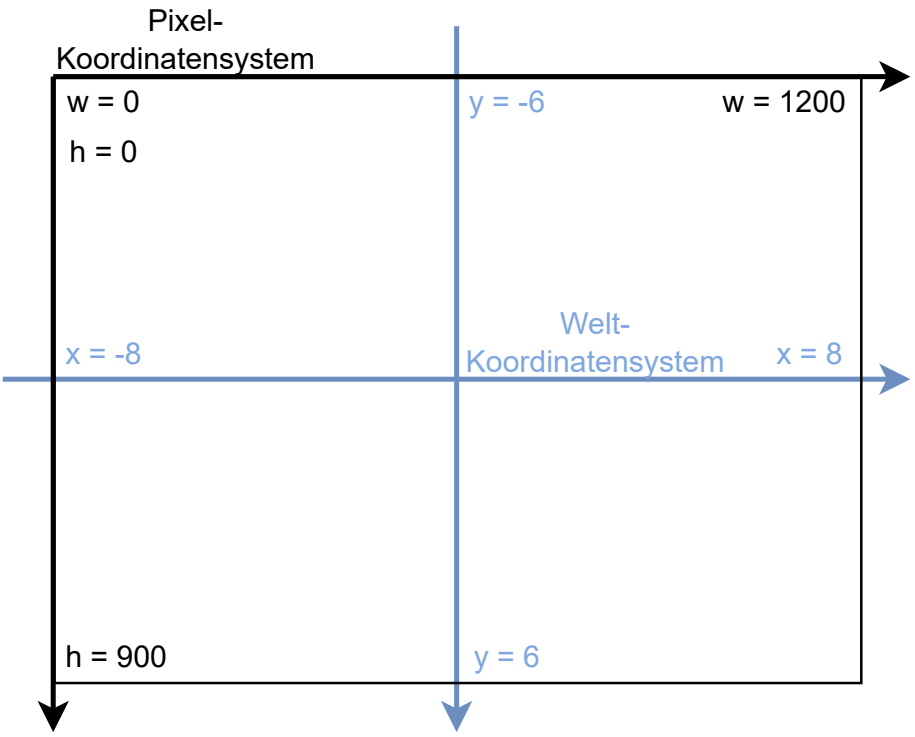


Abbildung 1.3: Vergleich der Weltkoordinaten (blau) und der Pixelkoordinaten (schwarz). Das Rechteck mit der etwas dünneren Linie markiert den Rand des Fensters.

2 Organisation

Der offizielle Bearbeitungszeitraum des Projekts beginnt am *Mittwoch, den 04.03.2015* und endet am *Freitag, den 20.03.2015*.

Die Aufgabenstellung wird am *Mittwoch, den 04.02.2015*, im Portal veröffentlicht. Ab diesem Zeitpunkt ist prinzipiell bereits die Bearbeitung der Aufgabe möglich. Wir raten ab diesem Zeitpunkt *dringend* zu einer Anmeldung als Gruppe von vier Student_innen.

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 2014/15* 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 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 Bewertung

Implementieren Sie eine lauffähige Java-Version des Spiels *Space Apes*, 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.

4 Aufgaben

4.1 Minimale Ausbaustufe

Ziel der minimalen Ausbaustufe ist es, das Spiel so weit zu entwickeln, dass es sinnvoll spielbar ist. Durch vollständiges Implementieren der minimalen Stufe, werden alle zum Bestehen benötigten Punkte erreicht. Fehlende Punkte aus der minimalen Ausbaustufe können nur in Ausnahmefällen ausgeglichen werden. In dieser Ausbaustufe können 50 von 100 Punkten erreicht werden. Die öffentlichen Testfälle für alle Ausbaustufen befinden sich in der Klasse `SpaceApes/src/tests.students.suites`. Grafische Aspekte sind durch die Tests nicht abgedeckt und werden vom Tutor separat bewertet.

Wechsel zwischen Gamestates - 7 Punkte

Das Spiel soll in einem Spielmenü beginnen, welches mindestens jeweils einen Button „Spiel starten“ und „Beenden“ mit entsprechender Funktionalität besitzt. Zusätzlich zur Interaktion durch das Klicken der Buttons, soll das Spiel mit „N“ gestartet und mit „Esc“ beendet werden können. Dies erlaubt die Anbindung der Tests. Auch im Spiel, soll das Spielmenü wieder durch Drücken der Escape-Taste aufgerufen werden können. Das Spielmenü muss zwingend über den Wert 0 identifizierbar sein und der Spielzustand (wenn „N“ gedrückt wurde) über den Wert 1 Hintergundbilder finden Sie im Verzeichnis `SpaceApes/img.assets`.

Initalisieren einer simplen Spielwelt - 5 Punkte

Für die minimale Ausbaustufe soll nach dem Klicken auf „Spiel starten“ eine einfache Spielwelt angezeigt werden. Dazu soll ein Hintergrundbild gesetzt, sowie 2 Planeten initialisiert werden. Ein Planet soll in der linken Spielfeldhälfte, der Andere in der rechten Hälfte lokalisiert sein (siehe Grafik 4.1). Für die minimale Ausbaustufe reicht es aus, wenn die Planeten an zwei, durch den Aufrufer der Methode festgelegten, Positionen erstellt werden. Sie können davon ausgehen, dass in den Tests nur sinnvolle Positionen übergeben werden. Die Radien und Massen der beiden Planeten (nur von denen, auf denen später Affen platziert werden), sollen vom Aufrufer der Methode übergeben werden. Außerdem soll es über eine Indikation möglich sein den Radius zufällig zwischen 0,75 und 1,5 zu wählen und davon linear abhängig eine Masse. In den Tests wird eine 0 als Indikation übergeben. Die pseudozufällige Positionierung der Planeten soll einen unterschiedlichen Spielverlauf für jede Partie ermöglichen. Grafiken für Planeten finden Sie im Unterverzeichnis `SpaceApes/img.planets`, Hintergundbilder in `SpaceApes/img.assets`.

Abbildung 4.1: Initiale Spielwelt: 2 Planeten mit jeweils einem Affen

Platzieren der Affen - 7 Punkte

Um später mit dem Spiel interagieren zu können, werden Spielfiguren benötigt. Jeder Spieler soll jeweils einen Affen besitzen, welcher sich auf der Oberfläche seines Heimatplaneten bewegen kann. Dazu soll auf beiden zuvor plazierten Planeten jeweils ein Affe bei Spielstart initialisiert werden. Hierfür ist es sinnvoll, dass die angezeigte absolute Position eines Affen abhängig ist von der Position seines Heimatplaneten und der relativen Position zum Planetenmittelpunkt. Für seine relative Position, sind Polarkoordinaten eine sinnvolle Darstellungsweise. Durch einen konstanten Abstand zum Planetenkern und einen Winkel ist die Position des Affen eindeutig bestimmt. Dies vereinfacht auch die im nächsten Schritt zu implementierende Affenbewegung. Ebenso lässt sich mithilfe des Winkels auch die Rotation des Affen an unterschiedlichen Positionen auf dem Planeten handhaben. Für die initiale Positionierung der Affen auf den Planeten gibt es analog zu den Planeten zwei Möglichkeiten. Zum einen soll der Aufrufer die Option besitzen, den Winkel des jeweiligen Affen auf seinem Planeten vorzugeben. Zum anderen soll, wenn der Wert 999 übergeben wurde, der Affe zufällig auf dem Planeten platziert werden. Bei einem Winkel von 0 Grad befindet sich der jeweilige Affe rechts auf seinem Planeten. Der Winkel läuft im Uhrzeigersinn (bei 90 Grad befindet sich der jeweilige Affe unten auf seinem Planet).

Affenbewegung - 7 Punkte

Ein Affe, der am Zug ist (siehe Spielzuglogik am Ende dieser Ausbaustufe), soll durch Drücken der rechten/-linken Pfeiltaste auf der Oberfläche seines Planeten laufen. Da seine Bewegung immer einer Kreisbahn folgt, (Der Radius dieser Bahn entspricht dem Planetenradius + halbe Höhe des Affen) ist eine Realisierung durch

Polarkoordinaten sinnvoll. Wird eine Pfeiltaste gedrückt, so wird der Winkel des Affen auf dem Planeten verändert.

Schießen entlang einer Geraden - 8 Punkte

In dieser Ausbaustufe wird zunächst ein einfaches Schussverhalten implementiert. Wird die Leertaste betätigt, soll bei dem aktiven Affen ein Projektil erzeugt werden. Ein Projektil besitzt x und y Koordinaten sowie eine Geschwindigkeit in beide Raumrichtungen v_x, v_y. Bei Initialisierung wird die Koordinatenposition etwas über den Affen gesetzt (damit später keine Kollision detektiert wird) und der Geschwindigkeitsvektor zeigt senkrecht vom Affen weg. Nun fehlt noch die Logik zum Aktualisieren der Position, um das Projektil fliegen zu lassen. Das Projektil soll nach jedem Zeitintervall δt das Positionsupdate

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{x}_t + \delta t \mathbf{v}_t \\ \mathbf{v}_{t+1} &= \mathbf{v}_t \end{aligned} \tag{4.1}$$

ausführen. Nach Implementierung dieser Gleichungen führt das Projektil eine geradlinige Bewegung unabhängig von seiner Umwelt aus. Der Geschwindigkeitsvektor soll in der minimalen Ausbaustufe fest auf 5 Koordinaten pro Sekunde gesetzt werden. Die Maße des Projektils sind zunächst nicht relevant, jedoch empfielt es sich den Radius als 0,25 zu wählen. Das ist ein sinnvoller Wert und der geforderte Wert, falls sie den CE-Aufgabenteil bearbeiten.

Kollision mit Planeten - 4 Punkte

Nach jedem Zeitschritt soll getestet werden, ob sich das Projektil innerhalb eines Planeten befindet und somit eine Kollision vor liegt. Ist dies der Fall, so wird das Projektil entfernt und optional eine Explosionsanimation abgespielt (siehe spätere Ausbaustufe). Der Test erfolgt mathematisch durch die Kreisgleichung

$$(p_x - m_x)^2 + (p_y - m_y)^2 \leq r^2. \tag{4.2}$$

Hierbei ist p_x, p_y die zu prüfende Position und m_x, m_y, r der Kreismittelpunkt und Radius.

Damit Schüsse nicht im Universum abhanden geraten, wenn beispielsweise kein Affe oder Planet getroffen wurde, muss eine Begrenzung eingebaut werden. Hierbei haben Sie keine konkrete Vorgabe. Sie können das Projektil entfernen, unmittelbar, nachdem es das Fenster verlassen hat oder auch erst ein wenig später (um ab Ausbaustufe 1 eventuell die Anziehungskraft der Planeten zu nutzen, damit es wieder zurückfliegt).

Kollision mit Affen - 2 Punkte

Analog zur Planeten Kollision, soll nun auch in jedem Zeitschritt überprüft werden, ob das Projektil einen Affen getroffen hat. Hierzu gilt wieder Gleichung 4.2. Der Radius beziehungsweise die Größe der Hitbox ist so zu wählen, dass das Spielerlebnis optimal ist. Daraufhin wird das Projektil entfernt und eine Explosion (die Explosion muss erst in einer späteren Ausbaustufe realisiert werden) angezeigt.

Spielzug Logik - 4 Punkte

Um die Spielzug Logik zu realisieren, ist es sinnvoll Spielerinteraktionen über ein Attribut `isActive` zu steuern. Nach Abschuss eines Projektils wird der aktive Affe auf Inaktiv gesetzt. Während der Flugzeit des Geschosses ist keine Interaktion möglich. Der Schuss ist beendet, wenn Kollision mit einem Planeten/Affen detektiert wurde, oder das Projektil sich zu weit aus dem Sichtbaren Spielbereicht entfernt hat, worauf hin es entfernt wird. Tritt eins dieser Szenarien ein, wird der Affe des nächsten Spieler aktiv.

Schadensberechnung - 4 Punkte

Damit ein Sieger ermittelt werden kann, müssen Projektile abhängig von ihrem Kollisionsort Schaden an Affen verursachen. Ein Projektil verfügt nun über 2 Eigenschaften: maximaler Schaden und Schadensradius. Der maximale Schaden gibt an, wie viele Lebenspunkte einem Affen bei einem direkten Treffer abgezogen werden. Der Schadensradius sagt aus in welchem Umkreis um die Explosion Schaden an Affen verursacht wird. An der Grenze des Schadensradius ist der Schaden 0, im Zentrum entspricht er dem maximalen Schaden. Werte dazwischen werden linear interpoliert. Ein Affe kann sich auch selbst abschießen und Schaden zufügen. Zu Beginn soll der Affe 100 Lebenspunkte haben.

Spielende - 2 Punkte

Sollten nach einem Schuss die Lebenspunkte eines Affen auf oder unter 0 sinken, ist das Spiel vorbei und der Gewinner wird in der Kommandozeile angezeigt. Nach dem Sieg eines Spielers kehrt man ins Hauptmenü zurück.

4.2 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 weitere 25 aus 100 möglichen Punkten erreicht werden. Die öffentlichen Testfälle für alle Ausbaustufen befinden sich in der Klasse `SpaceApes/src/tests.students.suites`.

Anzeigen von Explosionen - 2 Punkte

Um das Spielgefühl zu verbessern sollten Projektile bei Kollision nicht einfach verschwinden, sondern es sollte eine kleine Interaktion mit dem getroffenen Objekt sichtbar sein. Hierzu ist es möglich ein Explosionsbild für kurze Zeit anzuzeigen oder mehrere Bilder hintereinander als Animation abspielen zu lassen. Grafiken für die Explosion finden Sie unter `SpaceApes/img.explosions`.

Abbildung 4.2: Explosion

Ausgabe von Affendaten auf dem Planeten - 3 Punkte

Die aktuellen Lebenspunkte, Energie (Logik zum Energieverbrauch folgt in Ausbaustufe 2) und Coins (Items folgen in Ausbaustufe 3) der Affen sollten jederzeit im Zentrum des zugehörigen Planeten angezeigt werden. Beispielbilder stehen Ihnen unter `SpaceApes/img.assets` und `SpaceApes/img.items` zur Verfügung.

Abbildung 4.3: Entität `ApeInfoSign`

Bahnberechnung mittels explizitem Euler Verfahren - 10 Punkte

Das Projektil soll von Planeten angezogen werden. Hierzu wird zu jedem Zeitschritt eine numerische Integration mittels explizitem Eulerverfahren durchgeführt, um eine realistische Flugbahn zu approximieren. Die auf das Projektil wirkende Beschleunigung kann mathematisch durch die Gleichung

$$\ddot{\mathbf{x}} = G \sum_{i=1}^{n} m_i \frac{\mathbf{p}_i - \mathbf{x}}{\|\mathbf{p}_i - \mathbf{x}\|^3} \tag{4.3}$$

beschrieben werden. Hierbei ist $\mathbf{x}$ die aktuelle Position und $\ddot{\mathbf{x}}$ die aktuelle Beschleunigung des Projektils. Ein Planet besitzt die Masse m_i und den Positionsvektor $\mathbf{p}_i$. Die Summe läuft über alle n Planeten auf der Karte und G ist die Graviationskonstante, die vom Aufrufer gesetzt werden kann.

Wendet man das explizite Euler Verfahren an, erhält man die folgende Berechnungsvorschrift:
Die neue Position $\mathbf{x}_{t+1}$ des Projektils berechnet sich nach

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \delta_t \mathbf{v}_t \tag{4.4}$$

aus der alten Position und Geschwindigkeit sowie aus der vergangenen Zeit δ_t. Die neue Geschwindigkeit $\mathbf{v}_{t+1}$ des Projektils berechnet sich aus der alten Geschwindigkeit und der alten Beschleunigung (siehe Gleichung 4.3)

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \delta_t G \sum_{i=1}^{n} m_i \frac{\mathbf{p}_i - \mathbf{x}_t}{\|\mathbf{p}_i - \mathbf{x}_t\|^3}. \tag{4.5}$$

Die Realisierung kann durch Echtzeit Berechnung erfolgen. Hierfür wird für jeden Schritt die vergangene Zeit seit der letzten Berechnung übergeben. Die `update()` Funktion des Frameworks stellt für diesen Anlass den Parameter `delta` zur Verfügung. Die Masse des Projektils wird zu 1kg festgelegt (da die Anfangsgeschwindigkeit vorgegeben ist, benötigt man die Masse eigentlich nicht).

Schussparameter - 3 Punkte

Damit ein strategisches Spiel ermöglicht wird, ist es nötig, dass der Affe seine Schussrichtung und Schusskraft verändern kann. Jeder Affe hat einen lokalen Schusswinkel realtiv zu seiner aktuellen Ausrichtung auf dem Planeten (initialisiert mit 0 und Wertebereich $[-90°, 90°]$, wobei $90°$ nach rechts vom Affen aus gesehen bedeutet) und eine Schussstärke (initialisiert mit 5 und Wertebereich $[1, 7]$).

Control Panel - 7 Punkte

Damit die Benutzerinteraktion verbessert und erweitert wird, soll ein Control Panel in einer der Ecken angezeigt werden (siehe Grafik 1.2). Dieses besitzt Buttons zum Anpassen des Schusswinkels und der Schusstärke. Ebenso wird Spielername des aktiven Spielers angezeigt. Später bietet das Control Panel auch eine Option zum Wechseln von Projektilen und zeigt deren Preis an. Das Control Panel verschwindet immer nachdem ein Schuss getätigt wurde und erscheint wieder für den nächsten Spieler, sobald dieser an der Reihe ist. Dadurch wird visuell deutlich, wann ein Zug endet und beginnt. Das Panel zeigt immer zu Beginn die für den aktiven Affen gespeicherten Schussparameter aus dem letzten Zug an. Dies erlaubt dem Spieler seinen letzten Schuss zu justieren ohne sich die expliziten Werte merken zu müssen. Beispielbilder stehen Ihnen unter `SpaceApes/img.assets` und zur Verfügung.

Abbildung 4.4: Control Panel

4.3 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 weitere 15 aus 100 möglichen Punkten erreicht werden. Die öffentlichen Testfälle für alle Ausbaustufen befinden sich in der Klasse `SpaceApes/src/tests.students.suites`.

Anzeigen von Schadenswerten - 3 Punkte

Wird Schaden an einem Affen verursacht, soll dieser direkt sichtbar als Zahl neben dem Affen angezeigt werden. Optional kann die Visualisierung verschiedener Schadenswerte durch unterschiedliche Textfarben verstärkt werden (siehe Grafik 4.5).

Abbildung 4.5: Schadensanzeige nach einem Treffer

Anzeigen von Flugbahn Hilfslinie - 7 Punkte

Eine weitere Verbesserung der Visualisierung und des Spielgefühls soll durch Implementierung einer Ziellinie geschaffen werden (siehe Grafik 4.6). Hierzu wird die Flugbahn mithilfe eines unsichtbaren „Dummy“-Projektils berechnet und es werden in gewissen Abständen der Flugbahn sichtbare Punkte platziert. Diese Hilfslinie muss nun immer aktualisiert werden, sollte sich die vorhergesagte Flugbahn verändern. Dies ist in folgenden Szenarien der Fall:

- der Affe ändert seine Position
- der Schusswinkel ändert sich
- die Schussstärke ändert sich
- ein neuer Spieler ist am Zug

Um die Rechenintensität zu verringern sollte die Flugbahn nur für eine gewisse Strecke berechnet werden, die Hilfslinie muss also nicht so lang sein (siehe Grafik 4.6). Sie können damit gerne ein wenig herumspielen und auch längere Hilfslinien testen. Mit langen Hilfslinien können interessante Flugkurven schon vor dem Schuss gesehen werden (siehe Grafik 1.1). Die Hilfslinie soll während eines Schusses verschwinden. Die Grafik für Hilfslinienpunkte finden Sie unter `SpaceApes/img.assets`

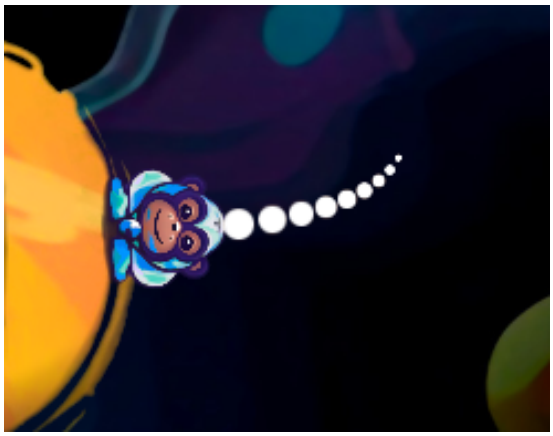


Abbildung 4.6: Hilflinie

Erweiterte Spielkartengenerierung - 4 Punkte

Um das Spiel interessanter zu gestalten, werden zusätzlich zu den zwei Planeten, auf denen sich Affen befinden, weitere Planeten gespawnt. Die Anzahl dieser Nicht-Affen-Planeten ist zufällig, wichtig ist nur, dass alle Planeten am Ende einen gewissen minimalen Abstand zueinander besitzen. Dafür reicht die minimale Distanz von 4 zu anderen Planetenmittelpunkten aus. Auch der Radius und die damit korrelierende Masse der Planeten (und im Idealfall auch das Bild) soll zufällig sein. Zufällig bedeutet dabei: Zufällig im Intervall wie in Aufgabenteil 4.1 beschrieben. Es muss jede Runde mindestens ein Nicht-Affen-Planet an einer zufälligen zulässigen Position gespawnt werden. Außerdem sollen die Planeten der Affen nun zufällig in einem geeigneten Bereich auf der linken und rechten Spielfeldhälfte platziert werden.

Bewegung auf dem Planeten verbraucht Energie - 1 Punkte

Um dem Spiel eine taktische Komponente hinzuzufügen besitzt jeder Affe nun eine limitierte Menge Energie. Energie wird benötigt, um sich auf dem Planeten zu bewegen. Ist diese aufgebraucht, ist keine Interaktion durch die Pfeiltasten mehr möglich und der Affe ist ein leichtes Ziel. Bei jedem Drücken der Pfeiltasten wird dem betroffenen Affen eine kleine konstante Menge Energie abgezogen. Hierfür muss etwas ausprobiert werden, um einen sinnvollen Energieverbrauch festlegen zu können.

4.4 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 weitere 10 aus 100 möglichen Punkten erreicht werden. Die öffentlichen Testfälle für alle Ausbaustufen befinden sich in der Klasse `SpaceApes/src/tests.students.suites`.

Musik - 2 Punkte

Musik soll abgespielt werden, sobald das Startmenü sichtbar wird. Eine Beispiel Sound Datei finden Sie im Verzeichnis `SpaceApes/snd`.

Item Spawning - 4 Punkte

Nach jedem Schuss soll mit einstellbaren Wahrscheinlichkeiten Items zufällig auf der Welt generiert werden (siehe Grafik 4.7). Dies sind entweder Coins verschiedener Wertigkeit oder Heilpakete/Energiepakete, welche dem Affen Leben/Enegie zurück geben. Items sollten nur im freien Weltraum generiert werden und können durch Abschießen eingesammelt werden. Grafiken für Items finden Sie unter `SpaceApes/img.items`. Um zumindest die öffentlichen Tests zu bestehen muss mindestens alle 6 Runden ein Item gespawnt werden. In den privaten Tests werden selbstverständlich noch weitere Eigenschaften getestet.

Kauf von Projektilen - 3 Punkte

Im Control Panel soll die Möglichkeit bestehen durch eingesammelte Coins einmalig bessere Projektile für den nächsten Schuss zu kaufen. Mögliche Veränderungen an neuen Projektilen sind: Größerer maximaler Schaden, veränderter Schadensradius, verändertes Flugverhalten, mehrere Projektile mit einem Schuss (Streuung, Staffelung). Neue Projektile sollten nur dann verwendbar sein, wenn der Spieler ausreichend Coins besitzt. Nach einem Kauf sollen diese natürlich abgezogen werden. Weitere Grafiken für Projektile finden Sie unter `SpaceApes/img.projectiles`.
In dieser Aufgabe wird lediglich bewertet, ob die Kauffunktion vorhanden ist und wie erwartet funktioniert. Das Implementieren von besonderem Verhalten der neuen Projektile ist hier nicht gefordert, kann Ihnen aber Bonuspunkte einbringen (siehe Abschnitt 4.7). Um die Kauffunktion zu testen, muss mindestens ein Projektil vorhanden sein, dass mehr als 0 Coins kostet.

Abbildung 4.7: Ansammlung an möglichen Items

Schwarzes Loch und Anti Planet - 1 Punkte

Damit das Spiel noch ein wenig interessanter wird und spektakuläre Flugbahnen erzeugt werden können, soll das Hinzufügen von Schwarzen Löchern (Masse immer größer als die maximal mögliche Masse der anderen Planeten) oder Anti-Gravitations Planeten (negative Masse) sein. Dies kann durch anpassen der Planeten Generierungs Logik realisiert werden. Über einen Indikator soll übergeben werden können, ob in der Welt zwingender Weise ein Schwarzes Loch und ein Anti-Planet gespawnt werden müssen. Beispielgrafiken finden Sie unter `SpaceApes/img.planets`.

4.5 Anpassungen für Studierende im Studiengang CE

Die Anforderungen für Studierende im Studiengang CE ersetzen Teile der Ausbaustufe I-III, wie unten erläutert. Dabei ist ein Aspekt umzusetzen: eine *realistische Flugbahn unter Berücksichtigung der Luftreibung, wenn sich das Projektil in der Atmosphäre des Planeten befindet*.

Achtung: die Anforderung „Bahnberechnung mittels explizitem Eulerverfahren“ der ersten Ausbaustufe ist zusätzlich *unverändert* zu implementieren, da sonst zahlreiche Tests scheitern. Gehen Sie dazu so vor, dass in Ihrem Spiel *zwei* Funktionen zur Berechnung der Flugbahn existieren oder Sie die hinzukommenden Berechnungsanteile über einen Indikator steuern (`boolean`, der angibt, ob Luftreibung genutzt werden soll). Das *Spiel* selbst soll die realistischere, aber komplexere Formel nutzen, während die *Tests* auf das vereinfachte „Wurfverhalten“ zurückgreifen.

Die folgenden Anforderungen gelten für Studierende im Studiengang Computational Engineering (CE) sowie für gemischte Gruppen, an denen CE-Studierende beteiligt sind. Dabei gehören die ersten beiden Punkte zusammen:

Die parabolische Flugbahn soll die Luftreibung wie folgt berücksichtigen:

$$F = 0.5 \cdot \rho \cdot C \cdot A \cdot V^2$$

mit

- A = Fläche des Projektils, die in Flugrichtung zeigt (also nur die halbe Oberfläche!)
- ρ = Dichte (1.293 $\frac{kg}{m^2}$) bei 0° Celsius und einem Atmosphärischem Luftdruck)
- C = 2.1
- V = Geschwindigkeit relativ zur Luftgeschwindigkeit, d.h. $v(t) - v_{luft}(t)$.

Die Luftgeschwindigkeit kann hier vernachlässigt werden. Auf die Berücksichtigung der Rotation des Projektils kann ebenfalls verzichtet werden. Die Masse des Projektils wird als 1kg festgelegt, sodass sich die Beschleunigung direkt aus der Kraft ergibt. Der Radius des Projektils muss zum bestehen der Tests auf 0,25 gesetzt werden.

Da im All keine Luftreibung existiert, soll diese nur berücksichtigt werden, wenn sich das Projektil in der Atmosphäre eines Planeten befindet. Die Atmosphäre ist beschrieben durch $a \cdot r$, wobei r den Planetenradius darstellt und $a > 1$ einen Faktor der vom Aufrufer vorgegeben werden kann. Bei uns hat sich ein Standardwert von $a = 1,5$ als sinnvoll herausgestellt.

Diese Aufgabe ersetzt die Teilaufgabe „Anzeige von Flugbahn Hilfslinie“ aus Ausbaustufe II und gibt somit 7 Punkte.

„Nicht-CE-Gruppen“, die diese Aufgaben (korrekt) bearbeiten, erhalten die genannten Punktzahlen als *Bonuspunkte*. Analog erhalten CE-Gruppen, die die „gestrichenen“ Aufgabenteile (korrekt) bearbeiten, die entsprechende Punktzahl als Bonuspunkte.

Bitte beachten Sie, dass die obigen Aufgaben *nicht* Bestandteil der Testfälle sind. Weicht das von Ihnen berechnete Verhalten von der (einfacheren) allgemeinen Formel ab, werden die entsprechenden Tests entsprechend fehlschlagen. Dadurch müssen Sie nicht in Panik geraten; ihr_e Tutor_in wird Ihre Implementierung dieser Aspekte im Code begutachten und bewerten. Denken Sie an den Tip mit den „zwei Varianten“, um Fehler in den Tests zu vermeiden!

4.6 Optionale Aufgabe im Bereich Wirtschaft

Unterstellen Sie für diese Aufgabe, dass Ihr Spiel die Aufgabenstellung vollständig und „mit schöner Optik“ (hübsche Bilder, gute Soundeffekte etc.) umsetzt.

Legen Sie sich nun auf einen (theoretischen) Vermarktungsweg für ihr Spiel fest. Zur Wahl stehen insbesondere der *iTunes Store* (iOS-Anwendungen), der *Google Play Store* (Android), der *Mac App Store* (Mac-Anwendungen), sowie der Microsoft Store (Windows 10-Anwendungen)—im Folgenden kurz als „Store“ abstrahiert[1].

Finden Sie für den von Ihnen gewählten *Store* heraus, welche Vermarktungsmöglichkeiten und welche Preisstaffelungen es gibt. So stehen z.B. im *iTunes Store* nur bestimmte Preise zur Verfügung. Recherchieren Sie zudem, welche Angebotsmöglichkeiten es gibt (Vollversion, Demoversion mit Kaufangebot, In-App-Käufe, integrierte Werbung, ...). Bestimmen Sie ebenfalls, welchen Anteil an den Kaufpreisen (mit Ausnahme von Gratisangeboten) der Store-Betreiber einbehält.

[1] Ignorieren Sie für die Bearbeitung dieser Aufgabe, dass das von Ihnen geschriebene Spiel in der vorliegenden Fassung in der Regel so in keinen der genannten Stores eingestellt werden kann (nicht direkt lauffähig unter iOS oder Android, ...)

Suchen Sie in dem gewählten Store nach verwandten oder vergleichbaren Spielen und betrachten Sie deren Preisgestaltung.

Entscheiden Sie sich anhand der so zusammengetragenen Informationen nun für eine Ihnen geeignet erscheinende Vermarktung. Ihr Ziel sollte (natürlich) die Erzielung des maximal möglichen Gewinns sein.

Bestimmen Sie angesichts der von Ihnen recherchierten Preis- und Bereitstellungsmodelle, des Betreiberanteils und der „Konkurrenzangebote“ das aus Ihrer Sicht beste Vermarktungsmodell. Fassen Sie die gewonnenen Erkenntnisse zu den genannten Punkten in einer kurzen Dokumentation zusammen, die mindestens die folgenden Angaben enthält:

- Gewählter Store
- Recherchierte verfügbare Vermarktungsoptionen (Vollversion, Demo mit Upgrade, ...)
- Recherchierte verfügbare Preisstaffelungen
- Anteil der Einnahmen, die an den Betreiber fließen
- Recherchierte Konkurrenzprodukte mit Preisgestaltung (mindestens drei)
- Begründete (!) Festlegung auf die Ihnen am besten erscheinende Vermarktung.

Bitte versuchen Sie, die Dokumentation verständlich und nachvollziehbar, aber auch so knapp wie möglich zu halten. Es soll kein Roman werden—in der Regel sollten 2-4 Seiten ausreichen! Zu dieser Aufgabe gibt es keine „Musterlösung“—Ihre Punktzahl (max. 10 Punkte) bestimmt sich daran, wie vollständig und überzeugend Ihre Recherche und die Begründung der letzlichen Entscheidung ist.

Bitte beachten Sie, dass diese Bearbeitung *ausschließlich theoretisch („auf Papier“)* erfolgt. Es wird also weder „Code geschrieben“ noch sollen Sie wirklich versuchen, das Spiel „online zu stellen“!

4.7 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. Beispielhafte Erweiterungen könnten sein:

- Unterschiedliches Flugverhalten der im Spiel zur Auswahl stehenden Projektiltypen (z.B flummi-artiges Verhalten, Streuschuss, visuelle Effekte)
- Implementierung eines Computer Gegners
- Kompatibilität für mehr als 2 Spieler

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*.

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

4.8 Materialien

Auf der Veranstaltungsseite stellen wir Ihnen einige Dateien zur Verfügung, die Sie für Ihre Lösung verwenden können. Dabei handelt es sich um Grafik Dateien, die Documentation des *eea* Frameworks sowie die öffentlichen Testfälle. Es wird Ihnen dringend nahe gelegt, unser bereitgestelltes Framework zu nutzen.

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. Die weitere Dokumentation des zu Grunde liegenden Frameworks Slick2D finden Sie unter https://slick.ninjacave.com/javadoc.
Ein Blick in diese Quellen empfiehlt sich sehr, um ein grundlegendes Verständnis zum Aufbau des Spiels zu erhalten.

Kurzer Überblick über das Framework

Klasse *Launch* erbt von *org.slick.StateBasedGame*. *Launch* dient sowohl als Container für Spiele als auch zum Starten des gesamten Spiels. Sie müssen diese Klasse erweitern, um zusätzliche States hinzuzufügen. Sie sollten jedoch zunächst mit einem Hauptmenü und einem Spielfenster beginnen.

Das **eea**-Framework arbeitet mit **Entitäten**, wie der **ImageRenderComponent**, sowie **Events** und zugehörigen **Actions**. Die folgenden zwei Tabelle geben einen Überblick über die mitgelieferten Events und Actions.

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 4.1: 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 4.2: Basisaktionen des **eea**-Frameworks aus dem package *eea.engine.actions.basicactions*. Die mit einem * markierten Aktionen empfehlen sich für das Spiel Space Apes nicht. Um die Affenbewegung zu realisieren, ist es sinnvoll eigene MoveActions zu implementieren.

4.9 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. Diskutieren Sie in der Gruppe, wie Ihr Projekt strukturiert werden soll. Über zentrale Elemente wie die grundsätzlichen Klassennamen und die Vererbungshierarchie sollten Sie sich einig sein.

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 4.3 gezeigt aussehen. Eine denkbare Gesamteinteilung für alle Ausbaustufen wird in Tabelle 4.4 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!

Aspekt	**Aufwand (ca.)**
Einarbeitung in die Aufgabenstellung und grober Überblick des Frameworks	4 Stunden
Einigung auf die grundsätzliche Klassen- und Packagehierarchie	1 Stunde
Wechsel zwischen Gamestates	4 Stunden
Initalisieren einer simplen Spielwelt	3 Stunden
Platzieren der Affen	4 Stunden
Affenbewegung	5 Stunden
Schießen eines Projektils entlang einer Geraden	6 Stunden
Kollision mit Planeten und Affen	4 Stunden
Spielzug Logik	3 Stunden
Schadensberechnung	3 Stunden
Spielende	2 Stunden

Tabelle 4.3: 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	6 - 8 Tage
Ausbaustufe I	3 - 4 Tage
Ausbaustufe II	2 - 3 Tage
Ausbaustufe III	2 Tage
Bonusaufgaben	Falls hier ein *starkes* Interesse besteht

Tabelle 4.4: Zeitplanung für die Bearbeitung aller Ausbaustufen mit 4 Personen

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.