Studienleistung 2: Tap and Fly

Wichtige Informationen

In dieser Studienleistung werden Sie ein Computerspiel implementieren. Um die Aufgabe zu bearbeiten, müssen Sie zuerst das Projekt STUD02_16WS_TapAndFly.zip in IntelliJ öffnen. Nutzen Sie zum Lösen der Aufgabenstellung die bereitgestellten Klassen und Dateien. Zum Einreichen Ihrer Aufgaben nutzen Sie die entsprechende Funktion in GRIPS. Falls Sie Probleme mit dem Starterpaket oder dem Einreichen der Aufgabe haben, können Sie sich in den Handouts auf GRIPS informieren.

> Bitte denken Sie daran, bei Nachfragen zur Studienleistung keinen Code im GRIPS-Forum zu posten. Wenn Sie Ihre Frage nicht ohne Beispiel stellen können, schreiben Sie uns eine Email.

ABGABE Die fertige Lösung dieser Aufgabe muss bis *zum 19. Dezember, 23:55 Uhr* über GRIPS eingereicht werden. Bitte beachten Sie: Erstellen Sie ein zip-Archiv (mit der korrekte Endung **.zip*) mit dem *kompletten* Projektordner. Überprüfen Sie vor dem Hochladen Ihrer Lösung, ob das von Ihnen erstellte Archiv funktioniert. Entpacke Sie dieses dazu an einen beliebigen Ort auf Ihrem Rechner und öffnen Sie das entpackte Projekt mit IntelliJ. Abgaben, die nicht im geforderten Format eingereicht werden oder nicht alle benötigten Dateien enthalten, werden bei der Korrektur nicht berücksichtigt und dementsprechend mit 5.0 bewertet. Sollten Sie technische Probleme beim Hochladen haben, wenden Sie sich bitte rechtzeitig per Mail oder Forum an uns.

ACHTUNG Eine Verlängerung der Abgabefrist ist nicht möglich. Einreichungen die uns zu spät oder per E-Mail erreichen, werden nicht berücksichtigt. Alle nicht eingereichten Aufgaben werden mit 5.0 bewertet. Testen Sie den Upload am besten schon vor Ablauf der Frist in Ruhe: Sie können bis zum Abgabetermin beliebig viele neue Lösungen einreichen. *Denken Sie bitte auch daran, die bekannten* ***Regeln zu Plagiaten in der Medieninformatik*** *zu beachten.*

Bewertungskriterien

> Seien Sie kreativ! Wenn Sie möchten, können Sie Ihr Spiel gerne erweitern. Entwerfen Sie ein eigenes Design oder überlegen Sie sich neue Spielmechaniken.

Für die gesamte Studienleistung gilt, dass die eingereichten Lösungen nur die in der Aufgabenstellung beschriebenen Probleme lösen sollen. Lassen Sie keinen Teil der jeweiligen Aufgabe weg und interpretieren Sie die Fragestellung nicht selbstständig. Bewertet wird, inwieweit Sie das beschriebene Problem vollständig lösen. Wenn Sie die Aufgaben erfolgreich bearbeitet haben, können Sie Ihre Lösung gerne kreativ gestalten und erweitern; achten Sie dabei darauf, dass die eigentlichen Anforderungen weiterhin erfüllt bleiben.

Die Qualität Ihres Codes fließt in die Gesamtnote mit ein: Nutzen Sie das Decomposition-Pattern um Ihre Programme übersichtlich zu gestalten. Verwenden Sie sinnvolle Bezeichner für Variablen und Methoden und kommentieren Sie ausreichend. Beachten Sie dazu die Kriterien für guten und schlechten Code, die in der Vorlesung erwähnt wurden.

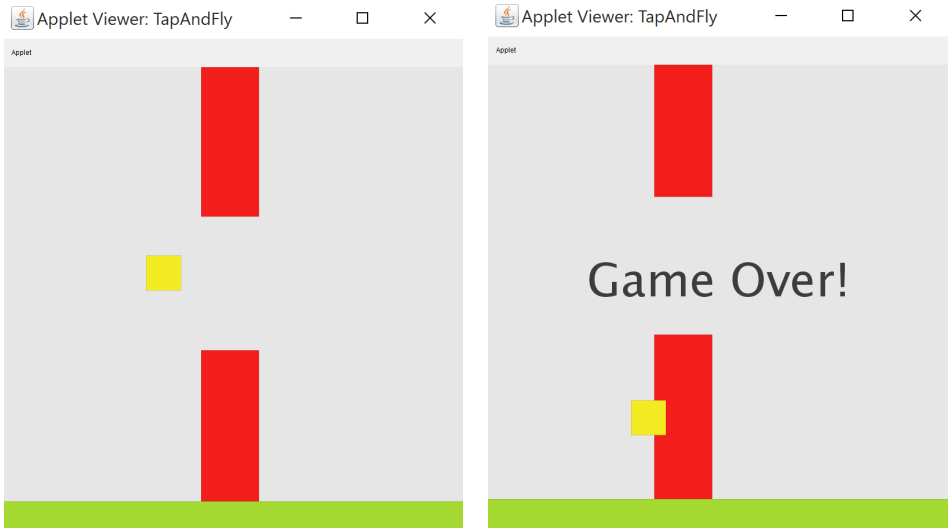
Zusammenfassung

In dieser Aufgabe sollen Sie ein Geschicklichkeits-Spiel entwickeln. Nutzen Sie dazu die `GraphicsApp`-Umgebung und die vorgegebene Programmstruktur. In dem Spiel soll eine Spielfigur zwischen mehreren Hindernissen hindurchfliegen. Ziel ist es dabei, möglichst viele der Hindernisse zu überleben, bevor das Spiel endet. Das Spiel ist verloren, wenn die Spielfigur gegen ein Hindernis fliegt oder den Boden berührt.

Alle wesentlichen Spielobjekte werden durch die bekannten Elemente der `GraphicsApp` dargestellt. Der Boden besteht aus einem Rechteck (`Rect`). Die Spielfigur sowie die Hindernisse werden ebenfalls als Rechtecke dargestellt. Über die *Leertaste* hat der Spieler die Möglichkeit, der fliegenden Spielfigur einen Impuls nach oben zu geben. Der Boden wird als statische Komponente auf den Bildschirm gezeichnet. Die Spielfigur bewegt sich *nur* vertikal. Der Effekt einer horizontalen Bewegung wird im Spiel dadurch erzeugt, dass die Hindernisse - über die entsprechenden Mechanismen der `Rect`-Klasse - vom rechten Rand des Bildschirms zum linken animiert werden. Die Hindernisse sind also die einzigen Objekte im Spiel, die sich horizontal bewegen.

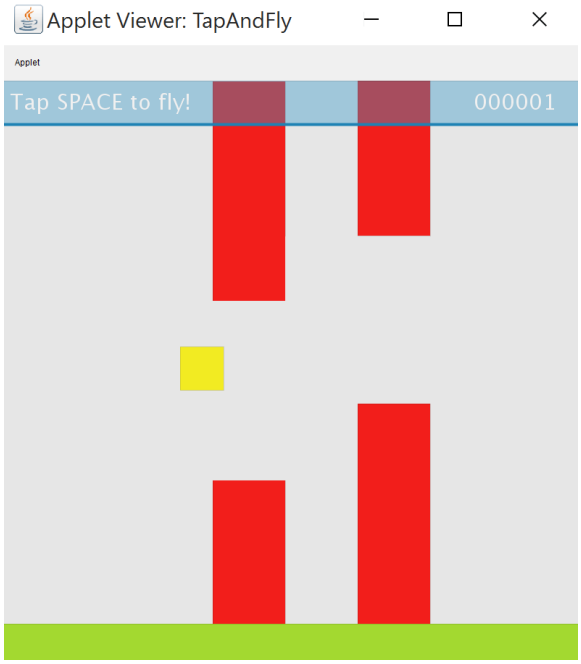
> Sie können die Farben und Positionen der Spielobjekte in diesen Screenshots als Anhaltspunkt für Ihr eigenes Spiel verwenden. Alternativ können Sie sich aber auch ein eigenes Design ausdenken.

Screenshots



Anforderungen

Um diese Aufgabe erfolgreich zu bestehen, müssen Sie eine Reihe von Anforderungen erfüllen. Diese werden im folgenden aufgeführt und erläutert. Bitte beachten Sie dabei: Die hier genannten *funktionalen* Anforderungen stellen den Mindestumfang Ihrer Lösung dar, um die Aufgabe zu bestehen. Für eine gute bis sehr gute Bewertung müssen Sie zusätzliche Elemente implementieren und auch auf qualitative Aspekte achten. An den entsprechenden Stellen sind daher sowohl die minimalen Anforderungen (bestehen) als auch die möglichen Erweiterungen für eine gute bis sehr gute Bewertung aufgeführt. Die Umsetzung einiger dieser Erweiterungen können Sie in diesem Screenshot anschauen:



Design und Parameter

Wählen Sie geeignete Farben, Positionen, Dimensionen und Parameter für die verschiedenen Komponenten der Anwendung. Es werden hierbei keine Vorgaben gemacht. Wählen Sie Werte, die zum einen eine benutzerfreundliche Verwendung des Spiels erlauben und zum anderen eine ansprechende Gestaltung bewirken. Achte Sie besonders auf die Relationen und Farbkontraste zwischen den Objekten. Ihr Spiel sollte am Ende einen positiven und durchdachten Gesamteindruck machen.

Minimale Anforderungen

Speichern Sie die Farben und Parameter, die Sie zur Platzierung und Gestaltung der Objekte benötigen in geeigneten Konstanten ab. Damit Sie die Darstellung des Spiels leichter anpassen und optimieren können, sollten Sie innerhalb Ihrer Anwendung immer nur auf diese Konstanten zurückgreifen. Vermeiden Sie auf jeden Fall sogenannte *Magic Numbers*. Überlegen Sie sich, in welchen Klassen Sie die unterschiedlichen Konstanten am besten unterbringen können.

Mögliche Erweiterung

Nutzen Sie eine zentrale Klasse zur Speicherung der Konfigurationswerte. Stellen Sie die dort gespeicherten Werte in den anderen Teilen Ihrer Anwendung dar. Dies können Sie erreichen, in dem Sie die Konstanten dort öffentlich zugänglich machen (`public`). Sind die Konstanten über das Schlüsselwort `static` auf Klassenebene angelegt, können Sie von überall - ohne Instanziierung - über den Namen der Konfigurations-Klasse auf die entsprechenden Werte zugreifen.

Boden

Erstellen Sie eine Klasse `Ground`, die Sie zur Darstellung des Bodens verwenden. Der Boden besteht aus einem Rechteck. Die Klasse verfügt über eine öffentliche Methode `draw`, mit deren Aufruf der Boden auf dem Bildschirm gezeichnet wird.

Spielfigur

Implementieren Sie für die Repräsentation der Spielfigur eine Klasse `Player`. Diese soll alle Variablen und Methoden beinhalten, die zur Realisierung der gewünschten Funktionalitäten benötigt werden. Erstellen Sie in der Klasse drei öffentliche Methoden, über die die Spielfigur gesteuert werden kann:

- `jump`: Über den Aufruf dieser Methode wird das Hochfliegen ausgelöst.
- `draw`: Über den Aufruf dieser Methode wird die Spielfigur an ihrer aktuellen Position gezeichnet.
- `update`: Über den Aufruf dieser Methode wird die Position der Spielfigur aktualisiert.

Minimale Anforderungen

Die Spielfigur wird als Rechteck dargestellt und *befindet sich in der Luft*. Betätigt der Spieler die Leertaste, fliegt die Spielfigur nach oben. Dabei bewegt sie sich vertikal bis zu einer bestimmten, selbst zu definierenden Stelle nach oben und fällt dann zum Boden herab. Der Spieler kann erneut einen Impuls geben, um weiter nach oben zu fliegen, ansonsten fliegt die Spielfigur stets nach unten. Berührt der Spieler den Boden, ist das Spiel vorbei.

Mögliche Erweiterung

Im Starterpaket finden Sie eine Audiodatei `jump.wav`. Nutzen Sie diese, um bei jedem Sprung der Spielfigur einen Sound abzuspielen.
Animieren Sie das *Abprallen*: Lassen Sie die Spielfigur von einem Hindernis abprallen, wenn sie dagegen stößt. Zusätzlich können Sie auch hier einen Sound abspielen lassen. Einen Vorschlag finden Sie in der Audiodatei `fall.wav`.

Hindernisse

Die Hindernisse in Ihrem *Tap and Fly*-Spiel sind rechteckige Blöcke, zwischen denen der Spieler hindurchfliegen muss. Stellen Sie diese als jeweils zwei Instanzen der `Rect`-Klasse dar. Die Hindernisse bewegen sich vom rechten zum linken Bildschirmrand. Initial wird jedes Hindernis außerhalb des sichtbaren Bildschirms platziert und erscheint erst durch die kontinuierliche Veränderung der x-Position. Sobald das aktuell sichtbare Hindernis aus dem linken Bildschirmrand hinaus bewegt wurde, wird ein neues Hindernis generiert. Dieses wird dann erneut hinter dem rechten Bildschirmrand platziert und bewegt sich dann von dort nach links. Erstellen Sie für die beschrieben Funktionaliät eine Klasse `ObstacleManager`. Die Klasse verwaltet die Hindernisse und bietet dazu eine Reihen von öffentlichen Methoden an:

- `createNewObstacle`: Mit dem Aufruf dieser Methode wird eine neue Grube erzeugt.
- `updateObstacle`: Mit dem Aufruf dieser Methode wird die Position des aktuellen Hindernisses aktualisiert.
- `drawObstacle`: Mit dem Aufruf dieser Methode wird die aktuelle Grube auf dem Bildschirm gezeichnet.
- `checkIfPlayerHitsObstacle(Rect playerBounds)`: Diese Methode bekommt ein Rechteck übergeben, das die aktuelle Position sowie die Dimensionen der Spielfigur repräsentiert. In der Methode wird überprüft, ob die Spielfigur mit der Grube kollidiert, also in diese hinein fällt.

Minimale Anforderungen

Es wird immer nur ein Hindernis dargestellt. Die Größe des Spalts wird für jedes neue Hindernis zufällig bestimmt. Im `ObstacleManger` legen Sie dazu geeignete Konstanten für eine minimale und maximale Spalthöhe fest. Alle Hindernisse bewegen sich mit der selben Geschwindigkeit über den Bildschirm.

Mögliche Erweiterung

Zählen Sie die überwundenen Hindernisse und blenden Sie die entsprechende Zahl auf dem Bildschirm ein.
Implementieren Sie einen Levelmechanismus, in dem Sie die Geschwindigkeit und/oder die Spaltgröße der Hindernisse mit zunehmender Spielzeit anpassen. Dadurch können Sie die Schwierigkeit und damit die Motivation für den Spieler bewusst gestalten.

Kollisionsabfrage

Sorgen Sie dafür, dass korrekt erkannt wird, wenn die Spielfigur gegen ein Hindernis fliegt. Dabei gilt folgende Spezifikation: *Die Spielfigur prallt gegen ein Hindernis, sobald sich ihre x-Position und y-Position mit der Position einer der Hindernis-Rechtecke überschneidet.* Sorgen Sie dafür, dass dieser Fall überprüft und korrekt verarbeitet wird. Bedenken Sie, dass Sie bei der Abfrage auch die Breite der Hindernisse und der Spielfigur berücksichtigen müssen.

Kommunikation innerhalb der Anwendung

Im Laufe des Spiels können zwei besondere Ereignisse auftreten, die im `ObstacleManger` erkannt werden und möglicherweise an anderen Stellen Ihres Programms relevant sind. Diese Ereignisse sind das erfolgreiche Überwinden eines Hindernisses und das Anstoßen des Spielers an einem Hindernis. Sie müssen diese Information innerhalb Ihrer Anwendung kommunizieren und haben dazu zwei Möglichkeiten.

Minimale Anforderungen

Erstellen Sie entsprechende, öffentliche Methoden in dem `ObstacleManager` und fragen Sie in der Hauptklasse Ihrer Anwendung (`TapAndFly`) regelmäßig ab, ob eines der Ereignisse eingetroffen ist. Reagieren Sie dann entsprechend darauf.

Mögliche Erweiterung

Implementieren Sie an geeigneter Stelle das vorgegebene Interface `ObstacleListener` und nutzen Sie dieses zur Realisierung eines *Observer/Listener*-Mechanismus mit dem Sie die Ereignisse (*Events*) aus dem `ObstacleManager` heraus kommunizieren können.

Spielende

Das Spiel endet, wenn die Spielfigur gegen eines der Hindernisse fliegt. Die Bewegung der Spielfigur und der Hindernisse wird in diesem Fall angehalten. Auf dem Bildschirm wird der letzte Zustand des Spiels angezeigt. Alternativ können Sie aber auch das *Herabfallen* der Spielfigur zum unteren Bildschirmrand animieren (Siehe dazu: Mögliche Erweiterungen der Spielfigur). Darüber hinaus gelten folgende Anforderungen:

Minimale Anforderungen	Mögliche Erweiterung
Blenden Sie in der Mitte des Bildschirms den Text "Game Over" ein.	Ermöglichen Sie es, das Spiel vom *Game Over*-Screen aus neu zu starten. Sie können dazu z.B. eine Taste verwenden und den Spieler durch einen eingeblendeten Text zum Neustart des Spiels auffordern.

Hinweise und Hilfestellungen

Während der Arbeit an der Studienleistung können Sie sich jederzeit per Mail oder über das GRIPS-Forum an uns wenden. Zögern Sie bitte nicht, bei Probleme oder Unklarheiten zu fragen.

Allgemeines Vorgehen

ACHTUNG Lesen Sie die Aufgabenstellung sorgfältig durch. Sollten Sie Teile der Anforderungen nicht verstehen, melden Sie sich rechtzeitig per Mail oder im Forum. Planen Sie ausreichend Zeit für die Bearbeitung der Aufgabe ein und beginnen Sie dementsprechend rechtzeitig mit der Implementierung Ihrer Lösung.

Bearbeiten Sie alle Teile der Anwendung losgelöst voneinander und in einer logischen Reihenfolge. Beginnen Sie mit den einfachen Bestandteilen des Spiels. Erweitern Sie dabei kontinuierlich den Rahmen Ihrer Anwendung in der zentralen Klasse `TapAndFly`. Testen Sie die neu hinzugekommenen Bestandteile, indem Sie Instanzen der entsprechenden Klassen erzeugen und deren Funktionen in den *draw loop* der Anwendung integrieren.

1. Implementieren Sie zuerst die Klasse `Ground` um den Boden der Spielwelt darzustellen.
2. Ergänzen Sie anschließend die Spielfigur durch Implementierung der `Player`-Klasse.
3. Zuletzt wird der `ObstacleManger` implementiert.
4. Kombinieren Sie die Funktionaliäten der so entstandenen Klassen sinnvoll in der Hauptklasse Ihrer Anwendung.

Sounds in der GraphicsApp

Die Klasse `Sound` der `GraphicsApp`-Umgebung erlaubt das Abspielen von Audiodateien. Diese Dateien müssen im *wav*-Format vorliegen und im *Package* `data.assets` des `src`-Ordners gespeichert sein. Übergeben Sie den korrekten Pfad zur gewünschten Audiodatei an den Konstruktor der Klasse. Auf dem instanziierten Objekt können Sie verschieden Methoden zur Steuerung der Audioausgabe aufrufen. Weitere Informationen dazu finden Sie auch in der Dokumentation.

```
private static final Sound JUMP_SOUND = new Sound("/data/assets/jump.wav");
JUMP_SOUND.play();
```

Stand: 5. Dezember 2016. Die Aufgabenstellung wurde von Alexander Bazo und Florin Schwappach verfasst. Dieser Text stellt die gültigen Anforderungen für die zweite Studienleistung im Kurs *Einführung in die objektorientierte Programmierung mit Java* dar.