Übung Softwaretechnik Sommersemester 2017

Aufgaben für die Studienleistung

Hinweise:

- Lesen Sie zunächst diese Hinweise komplett durch.
- Auf den folgenden Seiten finden Sie vierzig Aufgaben in vier Aufgabenbereichen.
- Aus jedem der Aufgabenbereiche müssen Sie eine Aufgabe bearbeiten, insgesamt also vier Aufgaben.
- Die von Ihnen zu bearbeitenden Aufgaben müssen Sie anhand Ihrer Matrikelnummer bestimmen.
  Sie müssen die Aufgaben mit den folgenden Nummern bearbeiten:
  - ⇨ 10 + letzte Ziffer Ihrer Matrikelnummer
  - ⇨ 20 + vorletzte Ziffer Ihrer Matrikelnummer
  - ⇨ 30 + drittletzte Ziffer Ihrer Matrikelnummer
  - ⇨ 40 + letzte Ziffer der Summe der letzten drei Ziffern Ihrer Matrikelnummer

  - Beispiele:
    Die Matrikelnummer 198681 führt zu den Aufgaben 11, 28, 36, 45
    Die Matrikelnummer 200142 führt zu den Aufgaben 12, 24, 31, 47
- Falls Sie sich unsicher sind, können Sie das Programm `Aufgabenbestimmung.jar` herunterladen und damit Ihre Aufgaben bestimmen lassen.
- Für die Bearbeitung der Aufgaben müssen Sie das Projekt *Swt-Student* (`Swt-Student.zip`) aus dem *moodle*-Arbeitsraum herunterladen.
- In der Zeit vom 29.5.2017 bis zum 14.6.2017 können Sie die Unterstützung von Tutoren im Pool OH12/4.031 in Anspruch nehmen. Beachten Sie die im *moodle*-Arbeitsraum angegebenen Arbeitszeiten.
- Die Tutoren werden Sie allerdings nur dann unterstützen, wenn Sie substantielle eigene Vorarbeiten erbracht haben. Sie müssen bereits Ideen für die Realisierung Ihrer Aufgaben entwickelt haben. Beachten Sie dazu die Beispiele, die für jeden der vier Aufgabenbereiche bereits im Projekt `Swt-Student` enthalten sind.
- Die Abgabe Ihrer Lösungen erfolgt im *moodle*-Arbeitsraum.
  - ⇨ Sie müssen für jede Aufgabe eine Java-Klasse in einer eigenen Datei erstellen.
  - ⇨ Ihre vier `.java`-Dateien können Sie im *moodle*-Arbeitsraum durch hochladen abgeben.
  - ⇨ Diese Abgabe (aller vier Klassen) nur ein einziges Mal vorgenommen werden. Geben Sie also erst ab, wenn Ihre Lösung vollständig ist.
  - ⇨ Die Abgabe muss bis zum 16.6.2017 erfolgen.
- Bewertung/Studienleistung:
  - ⇨ Aufgrund der langen Bearbeitungszeitraums, der vorliegenden Beispiele und der möglichen Unterstützung durch Tutoren gehen wir davon aus, dass nur Lösungen abgegeben werden, die die Aufgabenstellungen weitgehend korrekt umsetzen.
  - ⇨ Daher werden die Lösungen nur grob bewertet mit *akzeptierbar* (1 Punkt) oder *nicht akzeptierbar* (0 Punkte).
  - ⇨ Die Bewertung akzeptierbar setzt dabei eine weitgehende, aber nicht unbedingt vollständige korrekte Umsetzung der Aufgabe voraus.
  - ⇨ Abgaben, die nicht compilierbar sind, sind immer *nicht akzeptierbar*.
  - ⇨ Die Studienleistung wird mit 3 oder 4 Punkten erworben.
- Die Ergebnisse der Bewertung werden im *moodle*-Arbeitsraum angezeigt.

Aufgabenbereich 1

Entwurfsmuster Observer

Aufgabe 11:

Implementieren Sie eine Klasse `BulletCounter`, die ein `HudElement` darstellt, das zählt, wie oft der seinen Laser abgefeuert hat. Dazu soll ein `PlayerObserver` verwendet werden.

Für die grafische Darstellung kann die Bilddatei `"HUD/Counter"` mit einem Bildausschnitt von (0, 0, 32, 32) verwendet werden.

Zeigen Sie das `HudElement` während des Spiels am oberen Rand des Bildschirms an.

Aufgabe 12:

Implementieren Sie eine Klasse `PlayerHitCounter`, die ein `HudElement` darstellt, das zählt, wie häufig der Spieler bereits von Monstern getroffen wurde. Dazu soll ein `TargetObserver` verwendet werden.

Für die grafische Darstellung kann die Bilddatei `"HUD/Counter"` mit einem Bildausschnitt von (0, 0, 32, 32) verwendet werden.

Zeigen Sie das `HudElement` während des Spiels am oberen Rand des Bildschirms an.

Aufgabe 13:

Implementieren Sie eine Klasse `KillCounter`, die ein `HudElement` darstellt, das die Zahl der vom Spieler zerstörten Monster zählt und anzeigt. Dazu soll ein `PlayerObserver` verwendet werden.

Für die grafische Darstellung kann die Bilddatei `"HUD/Counter"` mit einem Bildausschnitt von (0, 0, 32, 32) verwendet werden.

Zeigen Sie das `HudElement` während des Spiels am oberen Rand des Bildschirms an.

Aufgabe 14:

Implementieren Sie eine Klasse `PickUpCounter`, die ein `HudElement` darstellt, das zählt, wie oft der Spieler ein PickUp eingesammelt hat. Dazu soll ein `PlayerObserver` verwendet werden.

Für die grafische Darstellung kann die Bilddatei `"HUD/Counter"` mit einem Bildausschnitt von (0, 0, 32, 32) verwendet werden.

Zeigen Sie das `HudElement` während des Spiels am oberen Rand des Bildschirms an.

Aufgabe 15:

Implementieren Sie eine Klasse `Speedometer`, welche ein `HudElement` darstellt, das die Geschwindigkeit des Spielers anzeigt. Dazu soll ein `PlayerObserver` verwendet werden.

Für die grafische Darstellung kann die Bilddatei `"HUD/Counter"` mit einem Bildausschnitt von (0, 0, 32, 32) verwendet werden.

Zeigen Sie das `HudElement` während des Spiels am oberen Rand des Bildschirms an.

Aufgabe 16:

Implementieren Sie eine Klasse `PickUpConsumption`, die ein `HudElement` darstellt, das zählt, wie viele PickUps ein Spieler verbraucht hat. Dazu soll ein `PlayerObserver` verwendet werden.

Für die grafische Darstellung kann die Bilddatei `"HUD/Counter"` mit einem Bildausschnitt von (0, 0, 32, 32) verwendet werden.

Zeigen Sie das `HudElement` während des Spiels am oberen Rand des Bildschirms an.

Aufgabe 17:

Implementieren Sie eine Klasse `TargetHitCounter`, die ein HudElement darstellt das zählt, wie viele Treffer der Spieler mit seinem Laser erzielt hat. Dazu soll ein `PlayerObserver` verwendet werden.

Für die grafische Darstellung kann die Bilddatei `"HUD/Counter"` mit einem Bildausschnitt von (0, 0, 32, 32) verwendet werden.

Zeigen Sie das `HudElement` während des Spiels am oberen Rand des Bildschirms an.

Aufgabe 18:

Implementieren Sie eine Klasse `HitRatio`, welche ein HudElement darstellt, das das Verhältnis der abgeschossenen Bullets zu den erzielten Treffern anzeigt. Dazu soll ein `PlayerObserver` verwendet werden.

Für die grafische Darstellung kann die Bilddatei `"HUD/Counter"` mit einem Bildausschnitt von (0, 0, 32, 32) verwendet werden.

Zeigen Sie das `HudElement` während des Spiels am oberen Rand des Bildschirms an.

Aufgabe 19:

Implementieren Sie eine Klasse `LaserUpgradeCounter`, die ein `HudElement` darstellt, das die Anzahl der vom Spieler eingesammelten LaserUpgrades zählt und anzeigt. Dazu soll ein `PlayerObserver` verwendet werden.

Für die grafische Darstellung kann die Bilddatei `"HUD/Counter"` mit einem Bildausschnitt von (0, 0, 32, 32) verwendet werden.

Zeigen Sie das `HudElement` während des Spiels am oberen Rand des Bildschirms an.

Aufgabenbereich 2

Entwurfsmuster Strategie

Aufgabe: 20

Implementieren Sie eine Klasse `DualLaser`, welche das Interface `EntityBehaviorStrategy` implementiert. Die Klasse `DualLaser` soll eine Fähigkeit des Spielers darstellen, welche durch den Druck der Taste `'D'` aktiviert werden kann. Die Fähigkeit sollte nicht zu schnell hintereinander verwendet werden können, daher sollen mindestens 200 Frames vergangen sein, bevor `DualLaser` erneut verwendet werden kann.

Sobald der `DualLaser` verwendet wird, sollen 2 Bullets von der Position des Spielers aus in einem Winkel von 30 Grad zueinander abgeschossen werden. Die Bullets sollen dem ersten getroffenen `Target`-Objekt, mit Ausnahme des Spielers, einen Schaden in Höhe von 2 zufügen. Die Bullets sollen eine Größe von 16 und eine Lebenszeit von 70 besitzen und sich mit einer Geschwindigkeit von 5 bewegen. Als Bullet kann die Klasse `edu.udo.cs.swtsf.core.player.BasicBullet` verwendet werden.

Fügen Sie ein Objekt der Klasse `DualLaser` beim Start des Spiels dem `Player`-Objekt hinzu.

Aufgabe 21:

Implementieren Sie eine Klasse `SmallBomb`, welche das Interface `EntityBehaviorStrategy` implementiert. Die Klasse `SmallBomb` soll eine Fähigkeit des Spielers darstellen, welche durch den Druck der Taste `'S'` aktiviert werden kann. Die Fähigkeit sollte nicht zu schnell hintereinander verwendet werden können, daher sollen mindestens 50 Frames vergangen sein, bevor `SmallBomb` erneut verwendet werden kann.

Sobald `SmallBomb` verwendet wird, sollen alle Entitäten vom Typ `Target` in einem Radius von 250 um die Position des Spielers einen Schaden in Höhe von 2 erhalten. Dabei soll der Spieler selbst nicht beschädigt werden. Für eine grafische Darstellung der Explosion kann die Klasse `exercise.starfighter.Explosion` verwendet werden.

Fügen Sie ein Objekt der Klasse `SmallBomb` beim Start des Spiels dem `Player`-Objekt hinzu.

Aufgabe 22:

Implementieren Sie eine Klasse `Stopper`, welche das Interface `EntityBehaviorStrategy` implementiert. Die Klasse `Stopper` soll eine Fähigkeit des Spielers darstellen, welche durch den Druck der Taste `'S'` aktiviert werden kann. Die Fähigkeit sollte nicht zu schnell hintereinander verwendet werden können, daher sollen mindestens 500 Frames vergangen sein, bevor `Stopper` erneut verwendet werden kann.

Sobald `Stopper` verwendet wird, sollen alle Entitäten vom Typ `Target` – mit Ausnahmen des Spielers – in einem Radius von 300 um die Position des Spielers einmal angehalten werden, also die Geschwindigkeit 0 erhalten. Anschließend sollen sie ihre Bewegung wieder fortsetzen können.

Fügen Sie ein Objekt der Klasse `Stopper` beim Start des Spiels dem `Player`-Objekt hinzu.

Aufgabe 23:

Implementieren Sie eine Klasse `Destroyer`, welche das Interface `EntityBehaviorStrategy` implementiert. Die Klasse `Destroyer` soll eine Fähigkeit des Spielers darstellen, welche durch den Druck der Taste `'D'` aktiviert werden kann. Die Fähigkeit sollte nicht zu schnell hintereinander verwendet werden können, daher sollen mindestens 1000 Frames vergangen sein, bevor `Destroyer` erneut verwendet werden kann.

Sobald `Destroyer` verwendet wird, sollen alle Entitäten vom Typ `Target` in einem Radius von 200 um die Position des Spielers einen Schaden in Höhe von 3 erhalten. Dabei soll der Spieler selbst nicht beschädigt werden. Für eine grafische Darstellung der Explosion kann die Klasse `exercise.starfighter.Explosion` verwendet werden.

Fügen Sie ein Objekt der Klasse `Destroyer` beim Start des Spiels dem `Player`-Objekt hinzu.

Aufgabe: 24

Implementieren Sie eine Klasse `QuadLaser`, welche das Interface `EntityBehaviorStrategy` implementiert. Die Klasse `QuadLaser` soll eine Fähigkeit des Spielers darstellen, welche durch den Druck der Taste `'Q'` aktiviert werden kann. Die Fähigkeit sollte nicht zu schnell hintereinander verwendet werden können, daher sollen mindestens 50 Frames vergangen sein, bevor `QuadLaser` erneut verwendet werden kann.

Sobald `QuadLaser`verwendet wird, sollen 4 Bullets von der Position des Spielers aus im rechten Winkel zueinander abgeschossen werden. Die Bullets sollen dem ersten getroffenen `Target`-Objekt, mit Ausnahme des Spielers, einen Schaden in Höhe von 1 zufügen. Die Bullets sollen eine Größe von 16 und eine Lebenszeit von 80 besitzen und sich mit einer Geschwindigkeit von 4 bewegen. Als Bullet kann die Klasse `edu.udo.cs.swtsf.core.player.BasicBullet` verwendet werden.

Fügen Sie ein Objekt der Klasse `QuadLaser` beim Start des Spiels dem `Player`-Objekt hinzu.

Aufgabe: 25

Implementieren Sie eine Klasse `BiLaser`, welche das Interface `EntityBehaviorStrategy` implementiert. Die Klasse `BiLaser` soll eine Fähigkeit des Spielers darstellen, welche durch den Druck der Taste `'B'` aktiviert werden kann. Die Fähigkeit sollte nicht zu schnell hintereinander verwendet werden können, daher sollen mindestens 100 Frames vergangen sein, bevor `BiLaser` erneut verwendet werden kann.

Sobald `BiLaser` wird, sollen 2 Bullets von der Position des Spielers aus im rechten Winkel zu seiner Bewegungsrichtung abgeschossen werden. Die Bullets sollen dem ersten getroffenen `Target`-Objekt, mit Ausnahme des Spielers, einen Schaden in Höhe von 2 zufügen. Die Bullets sollen eine Größe von 20 und eine Lebenszeit von 100 besitzen und sich mit einer Geschwindigkeit von 4 bewegen. Als Bullet kann die Klasse `edu.udo.cs.swtsf.core.player.BasicBullet` verwendet werden.

Fügen Sie ein Objekt der Klasse `BiLaser` beim Start des Spiels dem `Player`-Objekt hinzu.

Aufgabe: 26

Implementieren Sie eine Klasse `TriLaser`, welche das Interface `EntityBehaviorStrategy` implementiert. Die Klasse `TriLaser` soll eine Fähigkeit des Spielers darstellen, welche durch den Druck der Taste `'T'` aktiviert werden kann. Die Fähigkeit sollte nicht zu schnell hintereinander verwendet werden können, daher sollen mindestens 200 Frames vergangen sein, bevor `TriLaser` erneut verwendet werden kann.

Sobald `TriLaser` wird, sollen 3 Bullets von der Position des Spielers aus im rechten Winkel zueinander abgeschossen werden. Die Bullets sollen dem ersten getroffenen `Target`-Objekt, mit Ausnahme des Spielers, einen Schaden in Höhe von 1 zufügen. Die Bullets sollen eine Größe von 12 und eine Lebenszeit von 70 besitzen und sich mit einer Geschwindigkeit von 6 bewegen. Als Bullet kann die Klasse `edu.udo.cs.swtsf.core.player.BasicBullet` verwendet werden.

Fügen Sie ein Objekt der Klasse `TriLaser` beim Start des Spiels dem `Player`-Objekt hinzu.

Aufgabe 27:

Implementieren Sie eine Klasse `Shield`, welche das Interface `EntityBehaviorStrategy` implementiert. Die Klasse `Shield` soll eine Fähigkeit des Spielers darstellen, welche durch den Druck der Taste `'S'` aktiviert werden kann. Die Fähigkeit sollte nicht zu schnell hintereinander verwendet werden können, daher sollen mindestens 500 Frames vergangen sein, bevor `Shield` erneut verwendet werden kann.

Sobald `Shield` verwendet wird, soll der Spieler für 200 Frames nicht beschädigt werden können. Die Anzahl an Hitpoints des Spielers soll also während dieser Wirkungsdauer nicht verändert werden können.

Fügen Sie ein Objekt der Klasse `Shield` beim Start des Spiels dem `Player`-Objekt hinzu.

Aufgabe 28:

Implementieren Sie eine Klasse `BigBomb`, welche das Interface `EntityBehaviorStrategy` implementiert. Die Klasse `BigBomb` soll eine Fähigkeit des Spielers darstellen, welche durch den Druck der Taste `'B'` aktiviert werden kann. Die Fähigkeit sollte nicht zu schnell hintereinander verwendet werden können, daher sollen mindestens 100 Frames vergangen sein, bevor `BigBomb` erneut verwendet werden kann.

Sobald `BigBomb` verwendet wird, sollen alle Entitäten vom Typ `Target` in einem Radius von 500 um die Position des Spielers einen Schaden in Höhe von 1 erhalten. Dabei soll der Spieler selbst nicht beschädigt werden. Für eine grafische Darstellung der Explosion kann die Klasse `exercise.starfighter.Explosion` verwendet werden.

Fügen Sie ein Objekt der Klasse `BigBomb` beim Start des Spiels dem `Player`-Objekt hinzu.

Aufgabe 29:

Implementieren Sie eine Klasse `Guard`, welche das Interface `EntityBehaviorStrategy` implementiert. Die Klasse `Guard` soll eine Fähigkeit des Spielers darstellen, welche durch den Druck der Taste `'G'` aktiviert werden kann. Die Fähigkeit sollte nicht zu schnell hintereinander verwendet werden können, daher sollen mindestens 400 Frames vergangen sein, bevor `Guard` erneut verwendet werden kann.

Sobald `Guard` verwendet wird, soll der Spieler für 100 Frames nicht beschädigt werden können. Die Anzahl an Hitpoints des Spielers soll also während dieser Wirkungsdauer nicht verändert werden können.

Fügen Sie ein Objekt der Klasse `Guard` beim Start des Spiels dem `Player`-Objekt hinzu.

Aufgabenbereich 3

Entwurfsmuster Observer/Strategie

Aufgabe: 30

Implementieren Sie eine Klasse `MonsterBurst`, die von der Klasse `AbstractMonster` erbt und ein Monster im Spiel darstellt. Das Monster soll sich auf den Spieler zubewegen und bei der Kollision mit dem Spieler Schaden verursachen. Das Monster soll eine Geschwindigkeit von 4 erreichen, einen Schaden von 1 anrichten und 4 Trefferpunkte besitzen.

Sobald ein `MonsterBurst` zerstört wird, soll es 4 Bullets an seiner derzeitigen Position erzeugen, welche im rechten Winkel zueinander davon fliegen. Jedes Bullet soll eine Größe von 12 besitzen und sich mit einer Geschwindigkeit von 7 bewegen. Es soll nach 50 Frames von alleine aus dem Spiel entfernt werden.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel. Denken sie dabei besonders daran, wie das neue Monster grafisch im Spiel dargestellt werden kann.

Aufgabe: 31

Implementieren Sie eine Klasse `MonsterChange`, welche von der Klasse `AbstractMonster` erbt und ein Monster im Spiel darstellt. Das Monster soll sich auf den Spieler zubewegen und bei der Kollision mit dem Spieler Schaden verursachen. Das Monster soll eine Geschwindigkeit von 3 erreichen, einen Schaden von 2 anrichten und 5 Trefferpunkte besitzen.

Nachdem ein `MonsterChange` zerstört wurde, soll eine Instanz der Klasse `MonsterEasy` erzeugt und in der Nähe der eigenen Position zum Spiel hinzugefügt werden.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel. Denken sie dabei besonders daran, wie das neue Monster grafisch im Spiel dargestellt werden kann.

Aufgabe: 32

Implementieren Sie eine Klasse `MonsterLaser`, die von der Klasse `AbstractMonster` erbt und ein Monster im Spiel darstellt. Das Monster soll sich auf den Spieler zubewegen und bei der Kollision mit dem Spieler Schaden verursachen. Das Monster soll eine Geschwindigkeit von 1 erreichen, einen Schaden von 5 anrichten und 5 Trefferpunkte besitzen.

Jedes `MonsterLaser` soll alle 30 Frames ein Bullet abfeuern, welche sich in Richtung des Spielers bewegt und diesem bei einem Treffer einen Schaden von 1 zufügen. Das Bullet soll eine Größe von 10 besitzen und sich mit einer Geschwindigkeit von 4 bewegen. Es soll nach 100 Frames von alleine aus dem Spiel entfernt werden. Dieses Bullet soll keine anderen `Target`-Objekte außer dem Spieler beschädigen können.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel. Denken sie dabei besonders daran, wie das neue Monster grafisch im Spiel dargestellt werden kann.

Aufgabe: 33

Implementieren Sie eine Klasse `MonsterRandomSpeed`, von der Klasse `AbstractMonster` erbt und ein Monster im Spiel darstellt. Das Monster soll bei der Kollision mit dem Spieler Schaden verursachen. Das Monster soll einen Schaden von 2 anrichten und 5 Trefferpunkte besitzen.

Jedes `MonsterRandomSpeed` soll alle 100 Frames seine Geschwindigkeit zufällig wählen. Die gewählte Geschwindigkeit soll immer größer als 0 und kleiner als 6 sein.

Hinweis: Für die zufällige Wahl einer Geschwindigkeit kann die Klasse `java.util.Random` verwendet werden.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel. Denken sie dabei besonders daran, wie das neue Monster grafisch im Spiel dargestellt werden kann.

Aufgabe: 34

Implementieren Sie eine Klasse `MonsterSplit`, welche von der Klasse `AbstractMonster` erbt ein Monster im Spiel darstellt. Das Monster soll sich auf den Spieler zubewegen und bei der Kollision mit dem Spieler Schaden verursachen. Das Monster soll eine Geschwindigkeit von 2 erreichen, einen Schaden von 1 anrichten und 3 Trefferpunkte besitzen.

Nachdem ein `MonsterSplit` zerstört wurde, sollen drei Instanzen der Klasse `MonsterEasy` erzeugt und in der Nähe der eigenen Position zum Spiel hinzugefügt werden. Die neu erzeugten Monster sollten nicht alle die gleiche Position erhalten, damit sie leichter zu unterscheiden sind.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel. Denken sie dabei besonders daran, wie das neue Monster grafisch im Spiel dargestellt werden kann.

Aufgabe: 35

Implementieren Sie eine Klasse `MonsterOponent`, die von der Klasse `AbstractMonster` erbt und ein Monster im Spiel darstellt. Das Monster soll sich auf den Spieler zubewegen und bei der Kollision mit dem Spieler Schaden verursachen. Das Monster soll eine Geschwindigkeit von 3 erreichen, einen Schaden von 1 anrichten und 6 Trefferpunkte besitzen.

Immer wenn ein `MonsterOponent` einen Treffer erhält, soll es eine Bullet abfeuern, welche sich in Richtung des Spielers bewegt und diesem bei einem Treffer einen Schaden von 1 zufügen.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel. Denken sie dabei besonders daran, wie das neue Monster grafisch im Spiel dargestellt werden kann.

Aufgabe: 36

Implementieren Sie eine Klasse `MonsterAdd`, welche von der Klasse `AbstractMonster` erbt ein Monster im Spiel darstellt. Das Monster soll sich auf den Spieler zubewegen und bei der Kollision mit dem Spieler Schaden verursachen. Das Monster soll eine Geschwindigkeit von 2 erreichen, einen Schaden von 2 anrichten und 6 Trefferpunkte besitzen.

Immer wenn ein `MonsterAdd` einen Treffer erhält, soll eine Instanz der Klasse `MonsterEasy` erzeugt und in der Nähe der eigenen Position zum Spiel hinzugefügt werden.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel. Denken sie dabei besonders daran, wie das neue Monster grafisch im Spiel dargestellt werden kann.

Aufgabe: 37

Implementieren Sie eine Klasse `MonsterSpeedUp`, die von der Klasse `AbstractMonster` erbt und ein Monster im Spiel darstellt. Das Monster soll sich auf den Spieler zubewegen und bei der Kollision mit dem Spieler Schaden verursachen. Das Monster soll mit einer Geschwindigkeit von 1 beginnen, einen Schaden von 2 anrichten und 8 Trefferpunkte besitzen.

Immer wenn ein `MonsterSpeedUp` einen Treffer erhält, soll die Geschwindigkeit um 1 wachsen. Dabei soll aber eine Geschwindigkeit von 10 nicht überschritten werden.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel. Denken sie dabei besonders daran, wie das neue Monster grafisch im Spiel dargestellt werden kann.

Aufgabe: 38

Implementieren Sie eine Klasse `MonsterRandomDamage`, von der Klasse `AbstractMonster` erbt und ein Monster im Spiel darstellt. Das Monster soll bei der Kollision mit dem Spieler Schaden verursachen. Das Monster soll mit einer Geschwindigkeit von 3 beginnen und 5 Trefferpunkte besitzen.

Ein `MonsterRandomDamage` soll periodisch alle 100 Frames seine Schadenswirkung zufällig wählen. Der Schaden soll aber immer im Bereich zwischen 1 und 4 liegen.

Hinweis: Für die zufällige Wahl einer Bewegungsrichtung kann die Klasse `java.util.Random` verwendet werden.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel. Denken sie dabei besonders daran, wie das neue Monster grafisch im Spiel dargestellt werden kann.

Aufgabe: 39

Implementieren Sie eine Klasse `MonsterSpeedDown`, die von der Klasse `AbstractMonster` erbt und ein Monster im Spiel darstellt. Das Monster soll sich auf den Spieler zubewegen und bei der Kollision mit dem Spieler Schaden verursachen. Das Monster soll mit einer Geschwindigkeit von 5.5 beginnen, einen Schaden von 2 anrichten und 10 Trefferpunkte besitzen.

Immer wenn ein `MonsterSpeedDown` einen Treffer erhält, soll die Geschwindigkeit um 0.5 reduziert werden.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel. Denken sie dabei besonders daran, wie das neue Monster grafisch im Spiel dargestellt werden kann.

Aufgabenbereich 4

Entwurfsmuster Factory und Dekorierer

Aufgabe 40:

Implementieren Sie eine Klasse `MonsterSpriteFactoryRed`, die das Interface `GraphicalElementFactory` implementiert. `MonsterSpriteFactoryRed` soll für alle Monster der Klasse `MonsterEasy` einen Sprite erzeugen, der die Bilddatei "`Monsters/RedMonster`" verwendet.

Der Sprite soll animiert werden, indem über Zeit hinweg der Bildausschnitt der verwendeten Bilddatei verändert wird. Der Bildausschnitt soll abwechselnd zwischen (0, 0, 32, 32) und (32, 0, 32, 32) hin und her wechseln. Die Zeit zwischen dem Wechseln der Bildausschnitte sollte möglichst unabhängig von der Framerate des Spiels ablaufen.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel.

Aufgabe 41:

Implementieren Sie eine Klasse `MonsterSpriteFactoryRotateLeft`, die das Interface `GraphicalElementFactory` implementiert. `MonsterSpriteFactoryRotateLeft` soll für alle Monster der Klasse `MonsterEasy` einen Sprite erzeugen, der die Bilddatei "`Monsters/GreenMonster`" verwendet.

Der Sprite soll als Animation kontinuierlich gegen den Uhrzeigersinn um sein Zentrum rotieren.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel.

Aufgabe 42:

Implementieren Sie eine Klasse `MonsterSpriteFactoryScale`, die das Interface `GraphicalElementFactory` implementiert. `MonsterSpriteFactoryScale` soll für alle Monster der Klasse `MonsterEasy` einen Sprite erzeugen, welcher die Bilddatei "Monsters/GreenMonster" verwendet.

Der Sprite soll animiert werden indem über Zeit hinweg die Skalierung des Sprites geändert wird. Zunächst sollte der Sprite bis zu einer maximalen Skalierung von 2.0 wachsen, anschließend wieder auf eine Skalierung von 0.5 schrumpfen. Die Animation soll danach wieder von vorne beginnen und den Sprite erneut wachsen lassen. Die Zeit zwischen dem Wechseln der Bildausschnitte sollte möglichst unabhängig von der Framerate des Spiels ablaufen.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel.

Aufgabe 43:

Implementieren Sie eine Klasse `LaserUpgradeCooldown`, die das Interface `LaserUpgrade` implementiert. Eine Instanz von `LaserUpgradeCooldown` soll die Wartezeit zwischen zwei Schüssen des Lasers des Spielers um jeweils 1 verringern. Die Wartezeit darf jedoch niemals kleiner als 1 werden.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel.

Aufgabe 44:

Implementieren Sie eine Klasse `LaserUpgradeSizeUp`, die das Interface `LaserUpgrade` implementiert. Eine Instanz von `LaserUpgradeSizeUp` soll die Größe des Lasers des Spielers um jeweils 4 erhöhen. Die Größe des Lasers darf jedoch niemals über 64 liegen.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel.

Aufgabe 45:

Implementieren Sie eine Klasse `LaserUpgradeSpeedUp`, die das Interface `LaserUpgrade` implementiert. Eine Instanz von `LaserUpgradeSpeedUp` soll die Geschwindigkeit des Lasers vom Spieler um 1 erhöhen. Die Geschwindigkeit des Lasers darf jedoch niemals über 10 liegen.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel.

Aufgabe 46:

Implementieren Sie eine Klasse `MonsterSpriteFactoryRotateRight`, die das Interface `GraphicalElementFactory` implementiert. `MonsterSpriteFactoryRotateRight` soll für alle Monster der Klasse `MonsterEasy` einen Sprite erzeugen, der die Bilddatei "`Monsters/GreenMonster`" verwendet.

Der Sprite soll als Animation kontinuierlich im Uhrzeigersinn um sein Zentrum rotieren.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel.

Aufgabe 47:

Implementieren Sie eine Klasse `MonsterSpriteFactoryBlue`, die das Interface `GraphicalElementFactory` implementiert. `MonsterSpriteFactoryBlue` soll für alle Monster der Klasse `MonsterEasy` einen Sprite erzeugen, der die Bilddatei "`Monsters/BlueMonster`" verwendet.

Der Sprite soll animiert werden, indem über Zeit hinweg der Bildausschnitt der verwendeten Bilddatei verändert wird. Der Bildausschnitt soll abwechselnd zwischen (0, 0, 32, 32), (32, 0, 32, 32), (64, 0, 32, 32) und (96,0,32,32) wechseln. Die Zeit zwischen dem Wechseln der Bildausschnitte sollte möglichst unabhängig von der Framerate des Spiels ablaufen.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel.

Aufgabe 48:

Implementieren Sie eine Klasse `LaserUpgradeSizeAndSpeed`, die das Interface `LaserUpgrade` implementiert. Eine Instanz von `LaserUpgradeSizeAndSpeed` soll die Größe des Lasers um 2 erhöhen und die Geschwindigkeit um 1 vermindern. Dabei soll aber nie eine Größe von 32 überschritten und eine Geschwindigkeit von 1 unterschritten werden.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel.

Aufgabe 49:

Implementieren Sie eine Klasse `LaserUpgradeDamageAndCool`, die das Interface `LaserUpgrade` implementiert. Eine Instanz von `LaserUpgradeDamageAndCool` soll den Schaden des Lasers um 1 und die Wartezeit um 5 erhöhen.

Integrieren Sie Ihre Klasse auf geeignete Art und Weise in das Spiel.