Die Hausaufgaben müssen von jedem Studierenden einzeln bearbeitet und abgegeben werden. Für die Hausaufgabe sind die aktuellen Informationen vom Blog https://seblog.cs.uni-kassel.de/ws2223/programming-and-modelling/ zu berücksichtigen.

Abgabefrist ist der 08.12.2022 - 23:59 Uhr

Abgabe

Wir benutzen für die Abgabe der Hausaufgaben Git. Jedes Repository ist nur für den Studierenden selbst sowie für die Betreuer und Korrektoren sichtbar.

Für die Hausaufgabe benötigst du **kein neues** Repository. Es wird das gleiche Repository benutzt, das bereits in Hausaufgabe 4 angelegt wurde. Dieses kann über folgenden Link erstellt werden, falls nicht bereits geschehen:

https://classroom.github.com/a/n-8XoDkO

Nicht oder zu spät gepushte (Teil-)Abgaben werden mit 0 Punkten bewertet!

Vorbereitung

Zur Bearbeitung der Hausaufgabe sollte eine Entwicklungsumgebung verwendet werden. Wir empfehlen aufgrund der Nachvollziehbarkeit die Verwendung von IntelliJ (siehe Aufgabenblatt 3). Die Abgabe muss als **lauffähiges** Projekt abgegeben werden.

Abgaben, die nicht lauffähig sind, werden mit 0 Punkten bewertet!

Aufgabe 1 – Wireframes (13P)

Ziel dieser Aufgabe ist es, das Prototyping in Form von Wireframes zu üben. Erstellt aus den Anforderungen zu den zwei Oberflächen LoginScreen und BattleScreen jeweils ein Wireframe. Es darf sich an den Wireframes der Vorlesung und Übung orientiert werden.

Die Wireframes müssen als PDF-Datei (.pdf) abgegeben werden. Jedes Wireframe ist in einer eigenen Datei abzulegen.

Handschriftliche Abgaben werden mit 0 Punkten bewertet.

Zur Erstellung der Wireframes kann die Webanwendung diagrams.net verwendet werden.

1.1 LoginScreen

Im LoginScreen soll der Nutzer einen neuen Spieler erstellen und einen Kampf starten können.

- Um ein neues Spiel zu erstellen, werden Informationen zu einem Spieler benötigt.
- Für den Spieler wird diesem über eine Texteingabe ein Name zugewiesen.
- Über einen Button kann der Spieler erstellt und ein Kampf gestartet werden, woraufhin in den BattleScreen gewechselt wird.

1.2 BattleScreen

Im BattleScreen soll der Nutzer sein eigenes Team, sowie die gegnerischen Monster, sehen können.

- Alle gegnerischen Monster werden angezeigt.
- Alle Monster des eigenen Teams werden angezeigt.
- Das Team des Spielers und die gegnerischen Monster sind sichtbar voneinander getrennt.
- Es ist erkennbar, welche beiden Monster aktuell gegeneinander kämpfen.
- Für jedes Monster wird der Name, Typ, Hp/MaxHp und das Level angezeigt.
- Für die Monster des Spielers werden zusätzlich die aktuellen Exp dargestellt.
- Es gibt Buttons, welche die Abilities (Name/Typ) des aktuellen Monsters des Spielers darstellen.
- Es gibt einen Button, über den der Nutzer das Encounter verlassen kann.
- Es gibt einen Statustext, der Informationen über den Kampfverlauf wiedergeben kann.

Lege die erstellten **.pdf**-Dateien in einem Ordner mit dem Namen „Wireframes“ in deinem Repository ab. Handgezeichnete Abgaben sind nicht erlaubt. Committe und pushe die Änderung abschließend auf den main-Branch.

Bei der Bewertung wird vor allem darauf geachtet, dass auf den Wireframes alle benötigten Informationen und Eingabemöglichkeiten erkennbar sind.

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Aufgabe 2 – Test-First-Prinzip (37P)

Ziel dieser Aufgabe ist es, das Test-First-Prinzip einmal selbst durchzuführen. Hierzu ist folgende Aufgabenreihenfolge vorgegeben:

1. Tests designen
2. Tests implementieren
3. Methoden implementieren
4. Fehler dokumentieren

Jeder Schritt wird im Folgenden näher erläutert.

WICHTIG! Commit Message-Vorgaben

Der letzte Commit zu jeder Teilaufgabe soll mit der Commit Message „Finished Step <Nr>“ versehen sein.

Falls nach diesen Commits eine Aufgabe erneut bearbeitet wird, z. B. um einen Fehler zu korrigieren, soll die jeweilige Commit Message einfach noch einmal als letzte Commit Message verwendet werden.

Ohne separate Commits für jede Teilaufgabe können wir die Arbeit nicht nachvollziehen, wodurch diese automatisch mit **0 Punkten** bewertet wird

Also nochmal: Das Ignorieren dieser Vorgabe wird mit 0 Punkten bewertet!

Vorbereitung

Du kannst deine GenModel-Klasse[1] und die createPlayer- und createEncounter-Methoden der RandomGenerator-Klasse[2] mit den von uns zur Verfügung gestellten Dateien vergleichen und, wenn nötig, abändern.

Lege außerdem unter src/main/java im Package de.uniks.pmws2223 die Datei Constants.java[3] ab. Sie enthält Konstanten, die zur Implementierung der Spiellogik genutzt werden. Importiere die Konstanten in deinen BattleService mithilfe dieses Statements:

```
import static de.uniks.pmws2223.Constants.*;
```

Die Konstanten dürfen auch in den Tests verwendet werden.

Achte darauf, dass nach dem Kopieren der RandomGeneratorTest aus Hausaufgabe 4 weiterhin erfolgreich ist.

[1] GenModel.java:
https://github.com/sekassel/pmws2223-files/blob/main/HA05/GenModel.java
[2] RandomGenerator.java:
https://github.com/sekassel/pmws2223-files/blob/main/HA05/RandomGenerator.java
[3] Constants.java:
https://github.com/sekassel/pmws2223-files/blob/main/HA05/Constants.java

2.1 Tests designen

In der ersten Teilaufgabe sollen die Tests und Methoden in den jeweiligen Klassen erstellt, **aber noch nicht implementiert** werden. Ziel des ersten Schrittes ist es, die zu testende Struktur aufzubauen. Nach Bearbeitung hat das Projekt **keine Compiler-Fehler** und die erstellten **Tests schlagen nicht fehl**.

2.1.1 Neue Methoden im BattleService

Erstelle unter src/main/java im Package de.uniks.pmws2223.service die Klasse BattleService mit folgenden Methoden:

```
public void attack(Monster attacker, Ability ability, Monster defender) {}
public void selectNextMonster(Player player) {}
public void healAll(Player player) {}
public void gainExp(Monster monster, int exp) {}
public void levelup(Monster monster) {}
public double getAbilityMultiplier(Type abilityType, Type monsterType) {
    return 1;
}
```

2.1.2 Tests

Erstelle unter src/test/java im Package de.uniks.pmws2223.service die Test-Klasse BattleServiceTest mit folgenden Methoden:

```
@Test public void attack() {}
@Test public void attackAndDefeat() {}
@Test public void attackAndDefeatCurrent() {}
@Test public void selectNextMonster() {}
@Test public void healAll() {}
@Test public void gainExp() {}
@Test public void gainExpAndLevelup() {}
@Test public void levelup() {}
@Test public void getAbilityMultiplier() {}
```

Achte darauf, dass jede Methode mit @Test annotiert ist.

Committe und pushe die Änderungen auf den main-Branch, bevor du mit der nächsten Teilaufgabe fortfährst. Benenne möglichst eindeutig, dass die aktuelle Teilaufgabe (Step 1) abgeschlossen ist.

2.2 Tests implementieren

In der zweiten Teilaufgabe sollen die Tests zu den Methoden aus der vorangegangenen Teilaufgabe implementiert werden. Die Methoden im BattleService werden hier noch **nicht** implementiert. Folgende Verhaltensweise ist von den Methoden zu erwarten:

attack(Monster attacker, Ability ability, Monster defender)

Der Angreifer verwendet seine Fähigkeit gegen den Verteidiger. Dafür wird der Schadensmultiplikator berechnet und mit dem Atk-Wert der Fähigkeit multipliziert. Das Ergebnis wird abgerundet und dem Verteidiger an HP abgezogen.

Fällt der Verteidiger auf 0 HP oder weniger, wird er aus dem Encounter entfernt. Der Angreifer erhält (2 * Level des Verteidigers) Exp.

Ist der Verteidiger gerade das aktuelle Monster eines Spielers, wird ein neues Monster ausgewählt.

selectNextMonster(Player player)

Der Spieler wählt ein neues Monster, das noch am Leben ist. Dies wird verwendet, wenn das aktuelle Monster auf 0 HP fällt.

healAll(Player player)

Alle Monster des Spielers erhalten ihre HP zurück gemäß MaxHP.

gainExp(Monster monster, int exp)

Das Monster erhält die angegebenen Exp.

Wenn das Monster mindestens (10 + 10 * aktuelles Level) Exp hat, steigt es ein Level auf und die Exp werden um diesen Wert vermindert.

levelup(Monster monster)

Das Monster steigt ein Level auf. Es erhöht seine MaxHP um (5 + 0.5 * neues Level des Monsters) (abgerundet).

getAbilityMultiplier(Type abilityType, Type monsterType)

Wenn der Fähigkeitstyp stark gegen Monstertyp ist, gibt diese Methode 1.25 zurück.

Wenn er schwach dagegen ist, bzw. wenn der Monstertyp stark gegen den Fähigkeitstyp ist, gibt die Methode stattdessen 0.75 zurück.

Anderfalls wird 1 zurückgegeben.

Hinweis: Die Beschreibung der Methoden enthält sichtbar getrennt die zu behandelnden Fälle. Diese spiegeln sich auch in den Tests wieder.

Für das Aufbauen eines Teildatenmodells ist keine eigenständige Methode gefordert. Es wird empfohlen, jede Startsituation eines Testes in dem Rumpf der jeweiligen Testmethode zu implementieren, da sich diese von anderen unterscheiden kann. Für den initialen Aufbau des Spiels dürfen die createPlayer- und createEncounter-Methoden des RandomGenerators benutzt werden. Weitere Hilfsmethoden zum Aufbau der Startsituationen sind erlaubt.

Des Weiteren kann es sinnvoll sein, den Ablauf der einzelnen Testmethoden und deren zu testenden Eigenschaften zunächst mittels Kommentaren im Quelltext zu planen. Dies ist allerdings kein Muss.

Nach Bearbeitung dieser Teilaufgabe sollten einige der Tests bei der Ausführung fehlschlagen, da keine Logik in den Methoden implementiert wurde. Einige Tests bleiben allerdings weiterhin grün, da sie z. B. testen, dass keine Veränderung stattfindet.

Committe und pushe die Änderungen auf den main-Branch, bevor du mit der nächsten Teilaufgabe fortfährst. Benenne möglichst eindeutig, dass die aktuelle Teilaufgabe (Step 2) abgeschlossen ist.

2.3 Methoden implementieren

In der dritten Teilaufgabe sollen die Methodenrümpfe der Klasse BattleService implementiert werden. Diese folgen den schriftlichen Beschreibungen aus der vorangegangenen Teilaufgabe.

Nach Bearbeitung dieser Teilaufgabe sollten mehrere oder alle Tests bei der Ausführung durchlaufen. Sollte dies nicht der Fall sein, korrigiert diese Fehler erst in der folgenden Teilaufgabe. Weiterhin sind folgende Hilfestellungen gegeben:

attack(Monster attacker, Ability ability, Monster defender)

> Zur Berechnung des Schadensmultiplikators kommt die Methode getAbilityMultiplier(Type, Type) zum Einsatz. Die Auswahl eines neuen Monsters, wenn das aktuelle Monster des Spielers stirbt, übernimmt die Methode selectNextMonster(Player). Die Methode gainExp(Monster, int) übernimmt das Erhalten von Exp sowie den möglichen Levelaufstieg.

selectNextMonster(Player player)

> Zur Auswahl eines geeigneten (lebenden) Monsters kann ein Stream-Ausdruck oder eine Schleife verwendet werden.

gainExp(Monster monster, int exp)

> Für den Levelaufstieg soll die Methode levelup(Monster) verwendet werden.

Committe und pushe die Änderungen auf den main-Branch, bevor du mit der nächsten Teilaufgabe fortfährst. Benenne möglichst eindeutig, dass die aktuelle Teilaufgabe (Step 3) abgeschlossen ist.

2.4 Fehler dokumentieren

In der vierten Teilaufgabe sollen die konzeptionellen Fehler in Tests oder Logik korrigiert werden. Dabei soll bei jedem gefundenen Fehler ein Kommentar erstellt werden, welcher folgende Punkte beleuchtet:

- Wo trat der Fehler auf?
- Was hat den Fehler verursacht?
- War es ein Konzeptionsfehler im Test oder eine fehlerhafte Implementierung?

Schritt 4 ist vollständig abgeschlossen, sobald keine Fehler mehr korrigiert werden müssen und kein Test mehr fehlschlägt.

Committe und pushe die Änderungen abschließend auf den main-Branch. Benenne möglichst eindeutig, dass die aktuelle Teilaufgabe (Step 4) abgeschlossen ist.

Bei der Bewertung wird vor allem darauf geachtet, dass die Reihenfolge des Test-First-Prinzips eingehalten wurde.

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Anhang

Es folgt eine Auflistung hilfreicher Webseiten und weiteren Erklärungen zu den Themen dieser Hausaufgabe. Die Links sind als Startpunkt zur selbstständigen Recherche angedacht. Das Durcharbeiten der folgenden Quellen ist kein bewerteter Anteil der Hausaufgaben.

Zur Verfügung gestellte Dateien

- https://github.com/sekassel/pmws2223-files/tree/main/HA05

Wireframes

- diagrams.net: https://diagrams.net
- Balsamiq (Desktop-Anwendung): https://balsamiq.com/

Test Driven Development

- Agile Alliance zu TDD: https://www.agilealliance.org/glossary/tdd