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 22.12.2022 - 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! Abgaben, die nicht lauffähig sind, werden mit 0 Punkten bewertet!

Alle Tests (alt/neu) müssen nach wie vor funktionieren, sollte dies nicht der Fall sein, wird mit 0 Punkten bewertet!

Aufgabe 1 – Battle (14P)

In dieser Aufgabe soll der BattleService erweitert werden, um das Spiel für die Anbindung an die Oberfläche vorzubereiten. Hierfür müssen folgende Schritte ausgeführt werden, um bereits zuvor programmierte Funktionalität zu erweitern.

1.1 playRound

Füge dem BattleService eine neue Methode hinzu:

```
public int playRound(Monster attacker, Ability ability, Encounter encounter)
```

Die Methode soll folgenden Ablauf implementieren. Der BattleServiceTest muss die zugehörigen Fälle getestet werden. Befolge dabei das Test-First-Prinzip.

> Das aktuelle Monster des Spielers greift das erste Monster im Encounter mit der übergebenen Ability an.
>
> Sofern der Verteidiger noch mehr als 0 HP hat, macht er einen Gegenangriff mit einer geeigneten Attacke (Hinweis: wie diese ausgewählt wird, ist dir überlassen).
>
> Falls alle Monster im Encounter besiegt sind, werden die Monster des Spielers geheilt und die Methode gibt 1 zurück. Falls alle Monster des Spielers besiegt sind, werden die Monster des Spielers geheilt und die Methode gibt -1 zurück. Andernfalls wird 0 zurückgegeben.

1.2 levelup - Abilities

Die Methode levelup soll nun auch Abilities hinzufügen, die das Monster mit dem neu erreichten Level erlernen kann. Erweitere entsprechend den levelup-Test im BattleServiceTest.

Hinweis: Monster können nur Abilities vom Typ Normal und von ihrem eigenen Typ erlernen. Die zugehörigen Type-Objekte enthalten bereits alle lernbaren Abilities, diese müssen entsprechend ausgewählt und dem Monster hinzugefügt werden.

1.3 log

Erstelle im BattleService folgende Klassenvariable sowie einen zugehörigen Setter:

```
private Consumer<String> log = s -> {};
```

Füge an folgenden Stellen einen geeigneten Aufruf von `log.accept("...")` ein (dies dient später der Anzeige im Battle Log):

- Starker bzw. Schwacher Angriff, z.B. „Es ist sehr effektiv!“
- Verteidiger durch Angriff besiegt
- Monster erhält Exp
- Monster steigt ein Level auf
- Monster erlernt eine neue Ability
- Encounter gewonnen bzw. verloren

Aufgabe 2 – Controller (28P)

In dieser Aufgabe muss der BattleController angepasst und ein neuer MonsterController erstellt werden. Dafür müssen folgende Schritte durchgeführt werden:

2.1 Übergang Login – Battle

Der LoginController wird nun erweitert, sodass er mit dem RandomGenerator einen Spieler und ein Encounter erstellt und an den BattleController weitergibt.

2.2 Monster

Erstelle einen neuen MonsterController im Package de.uniks.pmws2223.controller unter src/main/java analog zum DevController aus der Vorlesung. Verwende die Elemente zur Darstellung eines Monsters aus Battle.fxml (nameLabel, hpLabel und lvlLabel, ggf. weitere) als Grundlage für eine neue Monster.fxml. Entferne alle Monster-Elemente (Boxen, Labels, etc.) aus Battle.fxml, sodass die Container encounterMonsterList und playerMonsterList keine Kind-Elemente enthalten.

Im MonsterController müssen die Werte des Monsters initial angezeigt und veränderbare Werte mit PropertyChangeListeners verknüpft werden.

Im BattleController muss für jedes Monster ein MonsterController erstellt, initialisiert und im entsprechenden Container dargestellt werden. Die MonsterController müssen in der destroy-Methode wiederum gestoppt und gelöscht werden. Außerdem muss ein PropertyChangeListener für die Monster des Encounters verwendet werden, um Änderungen in der Oberfläche anzuzeigen.

2.3 Battle Log

An dieser Stelle soll der zuvor implementierte Log in die Oberfläche übertragen werden. Beim Verwenden einer TextArea für den battleLog kann folgender Aufruf verwendet werden, um automatisch die Ausgabe zu übertragen:

```
battleService.setLog(battleLog::appendText);
```

Der BattleService muss dafür als Klassenvariable im BattleController angelegt werden.

2.4 Abilities

Nun sollen die Ability-Buttons dynamisch erstellt werden. Zur Vereinfachung verwenden wir eine neue Hilfsmethode im BattleController:

```
private void renderAbilities(Parent abilityBar, List<Ability> abilities)
```

Diese soll den Container abilityBar mittels `abilityBar.getChildren().clear()` leeren und alle Buttons neu hinzufügen.

Die Hilfsmethode kann dann sowohl beim initialen Darstellen, als auch in PropertyChangeListeners für currentMonster und dessen abilities einfach aufgerufen werden.

Aufgabe 3 – Testen (6P)

In dieser Aufgabe sollen die Erweiterungen der Oberfläche getestet werden.

Erweitere den AppTest, sodass er nun auch im Battle-Bildschirm verschiedene Angriffe durchführt, die zum Sieg für den Spieler führen, bevor der Test zum Login zurückkehrt. Prüfe dabei insbesondere die Werte der Monster sowie die Ausgabe im Battle Log.

Beachte, dass alle vorher angelegten Tests ebenfalls erfolgreich durchlaufen müssen! Passe wenn nötig alte Test-Klassen an, damit dies der Fall ist. Die Tests müssen weiterhin sinnvoll sein, insbesondere die geplante Funktionalität prüfen und Assertions enthalten.

Das Auskommentieren oder Löschen von alten Tests ist nicht erlaubt und führt zu Punktabzug!

Sollten die Tests nicht lauffähig sein oder nicht erfolgreich durchlaufen, führt dies zu 0 Punkten!

Hinweis

Das Spiel PMon soll am Ende dieser Hausaufgabe weitgehend spielbar sein. In Zukunft werden noch kleinere Spielmechaniken ergänzt, aber die Grundfunktionalität soll nun fertiggestellt sein.