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/ws2122/programming-and-modelling/` zu berücksichtigen.

Abgabefrist ist der 09.12.2021 - 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 benötigt ihr **kein neues** Repository. Es wird das gleiche Repository benutzt, das bereits in Hausaufgabe 4 angelegt wurde. Dieses kann über folgenden Link angelegt oder auch erstellt werden, falls nicht bereits geschehen:

`https://classroom.github.com/a/SlVERCvA`

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

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

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

Commit Message-Vorgaben

Wie in der letzten Hausaufgabe bereits vorgegeben, hier noch einmal eine Erinnerung an die Commit Message-Vorgaben. Diese gelten weiterhin, für diese und alle zukünftigen Hausaufgaben!

Der letzte Commit zu jeder Teilaufgabe soll mit der Commit Message
„HA<Number> finished Task <TaskNumber>" 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.

Das Ignorieren der Vorgaben wird mit 0 Punkten bewertet!

Diese Vorgaben gelten weiterhin! **Ab nächster Hausaufgabe sind diese Vorgaben allerdings nicht mehr zusätzlich auf dem Hausaufgabenblatt vermerkt!**

Aufgabe 1 - Wireframes (12P)

Ziel dieser Aufgabe ist es, das Prototyping in Form von Wireframes zu üben. Erstellt aus den Anforderungen zu den zwei Oberflächen SetupScreen und IngameScreen 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:

`https://www.diagrams.net`

SetupScreen

Im SetupScreen soll der Nutzer ein neues Spiel erstellen können.

- Um ein neues Spiel zu erstellen, werden Informationen zu zwei Spielern gebraucht.
- Je Spieler wird diesem über eine Texteingabe ein Name zugewiesen. Zusätzlich wird angezeigt, welcher der Spieler die weißen und welcher die schwarzen Spielsteine erhalten wird.
- Über einen Button kann das neue Spiel mit diesen beiden Spielern erstellt werden, woraufhin in den IngameScreen gewechselt wird.

IngameScreen

Im IngameScreen soll der Nutzer die Spielsteine auf dem Mühle-Spielbrett bewegen können.

- Das gesamte Spielfeld mit allen platzierten Steinen wird angezeigt.
- Die Spielphase wird angezeigt.
- Der aktuelle Spieler mit Farbe und Spieleraktion wird angezeigt.
- Es gibt einen Button, über den der aktuelle Spieler aufgeben 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 (33P)

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

- Schritt 1: Tests designen
- Schritt 2: Tests implementieren
- Schritt 3: Methoden implementieren
- Schritt 4: Fehler dokumentieren

Jeder Schritt wird im Folgenden näher erläutert. Bitte beachte, dass nach jeder Teilaufgabe ein Commit gemacht werden **muss**. Ohne diese können wir die Arbeit nicht nachvollziehen, wodurch diese automatisch mit **0 Punkten** bewertet wird.

Vorbereitung

Ihr könnt eure GenModel-Klasse[1] und die buildGame-Methode der ModelService-Klasse[2] mit den von uns zur Verfügung gestellten Dateien vergleichen und, wenn nötig, abändern.

Die startGame-Methode des ModelServices darf ebenfalls übernommen und in dieser Hausaufgabe verwendet werden.

Lege außerdem unter src/main/java im Package de.uniks.pmws2122 die Datei Constants.java[3] ab. Sie enthält String-Konstanten, die zur Implementierung der Spiellogik genutzt werden. Importiere die Konstanten in deinen ModelService mit Hilfe dieses Statements:
`import static de.uniks.pmws2122.Constants.*;`
Die Konstanten dürfen auch in den Tests verwendet werden.

[1] GenModel.java:
`https://github.com/sekassel/pmws2122-files/blob/main/HA05/GenModel.java`
[2] ModelService.java:
`https://github.com/sekassel/pmws2122-files/blob/main/HA05/ModelService.java`
[3] Constants.java:
`https://github.com/sekassel/pmws2122-files/blob/main/HA05/Constants.java`

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.

Neue Methoden im ModelService

Erstelle im ModelService folgende Methoden:

```
public void checkHorizontalMill(Man lastPutMan) {}
public void checkWinner() {}
```

Die ModelService-Klasse sollte bereits durch Hausaufgabe 4 im Projekt vorhanden sein.

Tests

Erstelle unter src/test/java im Package de.uniks.pmws2122.model folgende Testklassen mit entsprechenden Methoden:

Test-Klasse `HorizontalMillTest` mit den Methoden

```
public void testMillNoMill() {}
public void testMillOldMill() {}
public void testMillNewMillHorizontalFromMid() {}
public void testMillNewMillHorizontalFromLeft() {}
public void testMillNewMillHorizontalFromRight() {}
```

Test-Klasse `WinnerTest` mit den Methoden

```
public void testWinPlacingPhaseNoMenPlaced() {}
public void testWinPlacingPhasePlayerWithTwoMenPlaced() {}
public void testWinMovingPhaseNoWin() {}
public void testWinMovingPhaseWin() {}
```

Jede Test-Methode muss mit @Test annotiert sein!

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. Tests implementieren

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

checkHorizontalMill(Man lastPutMan)

> Hat ein Spieler durch den zuletzt platzierten/bewegten Spielstein eine horizontale Mühle gebildet, darf er anschließend einen gegnerischen Spielstein entfernen. Das wird dadurch repräsentiert, dass seine Spieleraktion dementsprechend „remove" ist. Hat er keine Mühle gebildet, ist seine Spieleraktion unverändert.Da ein Spieler mit weniger als drei verbleibenden Steinen keine Mühle mehr bilden kann, hat sein Gegenüber gewonnen. (Einen Gewinner kann es allerdings erst dann geben, wenn alle Steine initial platziert wurden.)

checkWinner()

> Da ein Spieler mit weniger als drei verbleibenden Steinen keine Mühle mehr bilden kann, hat sein Gegenüber gewonnen. (Einen Gewinner kann es allerdings erst dann geben, wenn alle Steine initial platziert wurden.)

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 buildGame- und startGame-Methoden des ModelServices benutzt werden. Die Implementierung weiterer 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.

3. Methoden implementieren

In der dritten Teilaufgabe sollen die Methodenrümpfe der Klasse ModelService 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:

checkHorizontalMill

In der späteren Spiellogik soll diese Methode aufgerufen werden, nachdem ein Spieler einen Man gesetzt hat. Möglicherweise darf ein Spieler dann einen gegnerischen Man entfernen, allerdings nur, wenn er gerade eine Mühle gebildet hat. Die Methode checkHorizontalMill soll dies prüfen und die Spieleraktion im Fall einer Mühle anpassen. Zur Vereinfachung wird zunächst nur auf horizontale Mühlen geprüft. Eine horizontale Mühle liegt vor, wenn das Feld, auf dem sich der zuletzt platzierte Man befindet, entsprechend linke und/oder rechte Nachbarfelder hat, auf denen ebenfalls Men der selben Farbe (oder des selben Owners) platziert sind, sodass es drei horizontal zusammenhängende Felder mit Men gleicher Farbe gibt. Wurde eine neue horizontale Mühle festgestellt, wird die Spieleraktion des zugehörigen Spielers auf „remove" gesetzt (hierfür kann die vorgegebene Konstante genutzt werden).

checkWinner

Diese Methode prüft, ob es einen Gewinner gibt und setzt die entsprechende Beziehung zwischen Spiel und Spieler. Ein Spieler gewinnt, wenn er so viele gegnerische Steine entfernen konnte, dass von diesen nur noch zwei verbleiben.

Tipp: Es kann geprüft werden, ob ein Spieler verloren hat. Daraus folgt, dass sein Gegner gewonnen hat.

Es ist außerdem zu beachten, dass das einfache Prüfen auf die Anzahl von Men zu einem Fehlverhalten führen kann. Beispielsweise dann, wenn ein Man immer erst beim Platzieren erstellt wird. Der Spieler hat somit anfangs bereits weniger als drei Men und könnte damit fälschlicherweise als Verlierer festgestellt werden. Daher sollte mithilfe eines zusätzlichen Checks sichergestellt werden, ob das Spiel so weit fortgeschritten ist, dass es sich wirklich um eine Niederlage bzw. einen Sieg handelt.

Als zusätzlicher Check könnte beispielsweise geprüft werden, ob das Spiel schon zur „moving"-Phase vorangeschritten ist, oder ob bereits alle Men des Spielers initial platziert wurden.

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.

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.

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/pmws2122-files/tree/main/HA05`

Wireframes

- Kurzer Artikel über Eigenschaften von Wireframes, Mockups und Prototypen:
  `https://uxplanet.org/wireframe-mockup-prototype-what-is-what-8cf2966e5a8b`