Das Projekt muss von jedem Studierenden einzeln bearbeitet und abgegeben werden. Für das Projekt sind die aktuellen Informationen vom Blog https://seblog.cs.uni-kassel.de/ws2223/programming-and-modelling/ zu berücksichtigen.

Abgabefrist ist der 23.02.2023 - 23:59 Uhr

Abgabe

Für das Projekt benötigst du ein **neues** Repository. Dieses kann über folgenden Link eingesehen oder auch erstellt werden, falls nicht bereits geschehen:

https://classroom.github.com/a/Iessai10

In dem Repository gibt es eine Selbstständigkeitserklärung. Diese **muss** zur Deadline des Projektes ausgefüllt und **unterschrieben** im Repository hinterlegt sein.

Vorbereitung

Mache dich mit dem Spiel „Uno"[1] vertraut. Es gelten vereinfachte Spielregeln, welche auf der folgenden Seite spezifiziert sind. **Diese Regeln müssen eingehalten werden.**

Auf den folgenden Seiten werden die zu bearbeitenden Aufgaben beschrieben. Das Programm darf in soweit frei gestaltet, mit zusätzlicher Funktionalität versehen und zusätzlich getestet werden, sodass die dargelegten Anforderungen erfüllt sind. Durch zusätzliche Funktionalität können allerdings keine Bonuspunkte erzielt werden.

Wir empfehlen, regelmäßig zu committen und zu pushen. Es gibt keine Vorgaben für Commit-Messages und zur Bearbeitungsreihenfolge. Dein Projekt muss vor der Deadline auf den main-Branch gepusht worden sein.

[1] Originale Spielregeln: https://www.unorules.com/

Spielregeln

Uno is a card game. The goal is to play all cards before every other player.

For our purpose the cards in a deck are infinite.

Setup: The game is for 2-4 players, 1 human and up to 3 Bots. Every player starts with seven cards, and they are dealt face down. The rest of the cards are placed in a Draw Pile face down. Next to the pile a space should be designated for a Discard Pile. The top card should be placed in the Discard Pile, and the game begins!

Game Play: The first player is always the human player and gameplay follows a clockwise direction. Every player views his/her cards and tries to match the card on the Discard Pile.

You have to match either by the number, color, or the symbol/Action. For instance, if the Discard Pile has a red card that is an 8 you have to place either a red card or a card with an 8 on it. You can also play a Wild card (which can alter current color in play).

If the player has no matches or they choose not to play any of their cards even though they might have a match, they must draw a card from the Draw pile. If that card can be played, play it. Otherwise, keep the card, and the game moves on to the next person in turn. You can also play a Wild card on your turn.

Once a player has no cards remaining, the game is over.

Action Cards: Besides the number cards, there are several other cards that help mix up the game. These are called Action or Symbol cards.

- **Reverse** - If going clockwise, switch to counterclockwise or vice versa. It can only be played on a card that matches by color, or on another Reverse card.
- **Skip** - When a player places this card, the next player has to skip their turn. It can only be played on a card that matches by color, or on another Skip card.
- **Draw Two** - When a person places this card, the next player will have to pick up two cards and forfeit his/her turn. It can only be played on a card that matches by color, or on another Draw Two.
- **Wild** - This card represents all four colors, and can be placed on any card. The player has to state which color it will represent for the next player. It can be played regardless of whether another card is available.

Aufgabe 1. Modellierung

Im Laufe des Projekts soll das Spiel „Uno" modelliert und programmiert werden, sodass als Endprodukt ein Programm entsteht, das es ermöglicht, gegen bis zu 3 Bots „Uno" über eine grafische Oberfläche zu spielen.

Hierzu sollen zunächst die kennengelernten Methoden zur Modellierung angewendet werden.

Achte auf die vorgegebenen Dateiformate. Von Hand gezeichnete Diagramme oder Wireframes werden nicht akzeptiert. Scenebuilder-Screenshots werden als Wireframes nicht akzeptiert.

1.1 Szenarien

Schreibe drei Szenarien zu „Uno". Die Szenarien müssen in Englisch verfasst sein.

Die Szenarien sollen in deinem Repository im Ordner `modelling` in der Datei `scenarios.md` zu finden sein.

1.2 Objektdiagramme ableiten

Leite aus deinen Szenarien Objektdiagramme für Start- und Endsituation ab. Lege die sechs erstellten pdf-Dateien wie folgt in deinem Repository ab:

```
modelling/diagrams/<Szenario-Titel>_<Start bzw. Result>.pdf
```

1.3 Klassendiagramm ableiten

Leite aus deinen Objektdiagrammen ein Klassendiagramm ab. Lege die erstellte pdf-Datei wie folgt in deinem Repository ab:

```
modelling/diagrams/classdiagram.pdf
```

1.4 Wireframes

Erstelle Wireframes für einen Setup-/Ingame- und einen GameOverScreen deines Spiels. Lege die pdf-Dateien wie folgt in deinem Repository ab:

```
modelling/wireframes/Setup.pdf
modelling/wireframes/Ingame.pdf
modelling/wireframes/GameOver.pdf
```

Aufgabe 2. Programmierung

In dieser Aufgabe wird das „Uno"-Spiel vervollständigt. Dazu wird die Oberfläche erstellt und die Spiellogik implementiert. Entsprechend des MVC-Patterns sollen sie durch Controller miteinander verknüpft werden. Außerdem soll durch Tests sichergestellt werden, dass dein Programm funktioniert.

Die Anwendung muss unter Verwendung von JavaFX umgesetzt werden.

2.1 Datenmodell generieren

Verwende die bereits in deinem Projekt existierende Klasse GenModel, um dein Klassendiagramm aus Aufgabe 1.3 zu definieren sowie zu generieren.

2.2 FXML-Dateien

Erstelle anhand der in Aufgabe 1 erstellten Wireframes die entsprechenden FXML-Dateien mit dem Scenebuilder. Lege diese unter src/main/resources im Ordner de/uniks/pmws2223/uno/view ab.

2.3 MVC

Implementiere folgende Klassen:

- de.uniks.pmws2223.uno.service.GameService
  enthält benötigte Logik-Methoden für das Spiel
- de.uniks.pmws2223.uno.service.BotService
  enthält benötigte Logik-Methoden für die Bots
- de.uniks.pmws2223.uno.Constants
  als zentraler Ort für Konstanten
- de.uniks.pmws2223.uno.controller.SetupController
  als Controller für den Setup-Bildschirm
- de.uniks.pmws2223.uno.controller.IngameController
  als Controller für den Ingame-Bildschirm
- de.uniks.pmws2223.uno.controller.GameOverController
  als Controller für den GameOver-Bildschirm
- de.uniks.pmws2223.uno.controller.<THING>Controller
  als benötigte(r) Sub-Controller

Es können weitere Controller als auch Services erstellt werden, wenn dies nach eigener Einschätzung zu besser strukturiertem Code führt.

Danach sollte deine Anwendung **vollständig** ausführbar und spielbar sein. Starte sie mit der Klasse Main, um das Spiel auszuprobieren und zu spielen, bis ein Sieger feststeht.

2.4 BotService

Ein Bot muss eigenständig anhand der eigenen Karten und der aktuell ganz oben liegenden Karte entscheiden, welchen Spielzug er macht. Hierbei darf der Bot beliebig komplex gestaltet werden. Aber auch eine simple Reihe von Abfragen ist erlaubt.

Damit man die Spielzüge der Bots verfolgen kann, verwende einen Timer/TimerTask, welcher eine Verzögerung von minimal 500ms bis maximal 2000ms erzeugt.(Siehe Übung)

2.5 Random

Das Austeilen der Spielkarten und das Deck, von welchem Karten gezogen werden können, muss zufällig geschehen. Verwende hierfür die Klasse `Random`[2] aus `java.util`. (Siehe Übung)

2.6 Tests

- `GameServiceTest` - Karten (Zahl/Farbe, +2, Richtungswechsel, Farbe aussuchen) ausspielen testen
- `BotServiceTest` - Den Spielzug eines Bots testen (Auf welche Art und Weise entscheidet der Bot seinen Spielzug)

Erkläre in jeder Methode in Form von Kommentaren, welche Anforderung an das Verhalten der Logik-Methode geprüft wird. Lege den `GameServiceTest` und den `BotServiceTest` im Modul `src/test/java` im package de.uniks.pmws2223.uno.service ab.

Implementiere außerdem einen GUI-Test mit Namen `GameTest` im Modul `src/test/java` im package de.uniks.pmws2223.uno. Implementiere darin einen Test, der ein Spiel mit drei Bots (mit dem Spieler insgesamt 4 Spieler) startet und diese mindestens drei Runden spielen lässt. Verwende einen Seed für die zufällig ausgeteilten Karten und ggf. vorhandenen zufälligen Entscheidungen von Bots, um ein gleichbleibendes Spiel zu testen.

Es dürfen zusätzliche Tests implementiert werden.

[2] https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

Informationen zur Präsentation

- Dein Projekt muss pünktlich abgegeben werden (siehe Deadline).
- Im Laufe der zweiten Bearbeitungswoche wird ein Doodle gepostet (Ankündigung über Discord), über das sich die Studierenden einen Zeitslot auswählen müssen, in dem sie ihr Projekt präsentieren. Diese Präsentationen werden erst nach der Abgabe stattfinden.
- Die Präsentation wird über Discord stattfinden. Hierfür ist eine Webcam erforderlich, da wir deine Identität prüfen müssen. Halte hierfür deinen Personalausweis und deinen Studierendenausweis bereit.
- Es soll **keine** PowerPoint-Präsentation vorbereitet werden. Die Ergebnisse des Projektes werden am laufenden Programm, am Programmcode und anderen Projektdateien erklärt.
- Innerhalb von ca. 5 Minuten sollen die Ergebnisse von Aufgabe 2 als Demo präsentiert werden.
- Innerhalb von ca. 10 Minuten soll der Programmcode präsentiert werden.
- Die Prüfenden können jederzeit Fragen dabei stellen.
- Die Auswertung wird einige Zeit dauern, daher findet die Notenvergabe erst entsprechend später statt.