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/ws1920/programming-methodologies/` zu berücksichtigen.

Abgabefrist ist der 19.12.2019 - 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 Korrekteure sichtbar. Für diese und voraussichtlich alle zukünftigen Abgaben wird kein neues Repository benötigt. Nutzt weiterhin jenes, welches mit folgendem Link angelegt wurde:

`https://classroom.github.com/a/X9QUkSjx`

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!

Commit Message Konventionen

In dieser und zukünftigen Hausaufgaben führen wir eine Konvention für die Commit Messages ein.

Solche Commits, welche als Abgabe einer Hausaufgabe dienen, werden mit der Nachricht: ”Final HA<Number>” versehen. Für die Abgabe von Teilaufgaben (wie in Aufgabe 1 dieses Blattes), wird folgende Konvention eingeführt ”HA<Number> finished Task<TaskNumber>”.

Falls in Folge der Abgabe-Commits ein Fehler auffällt, welchen ihr zeitlich gesehen noch korrigieren könnt, so verwendet für den Folge-Commit wieder die Message ”Final HA<Number>”. Der aktuellste Commit mit dieser Nachricht, vor der Deadline, wird als Abgabe gewertet.

Das Ignorieren der Konventionen wird mit 0 Punkten bewertet!

Scenebuilder

Ab sofort werden alle Elemente der GUI mit dem Scenebuilder gebaut. Das bedeutet, dass nur in ausdrücklichen Ausnahmefällen innerhalb einer Controller-Klasse ein GUI-Element instanziiert werden darf. Der Scenebuilder speichert seine Dateien im fxml-Format. Diese müssen im Resources-Verzeichnis des Gradle-Projektes unter der selben Packagestruktur abgelegt werden wie diejenige Klasse, die für das Laden der Datei zuständig ist. Detaillierte Erläuterungen sind der Übung zu entnehmen.

Projekte deren GUI nicht mit FXML-Dateien umgesetzt sind, werden mit 0 Punkten bewertet!

Projekt-Frühjahrsputz

Bevor wir mit der eigentlichen Implementierung des Projektes beginnen, sollten alle Altlasten der vergangenen Hausaufgaben beseitigt werden, welche nicht länger zur Entwicklung benötigt werden. Dies sind hauptsächlich die Datenmodelle aus HA4 Task 1 und HA6 Task 1, sowie deren Scenarios und Tests.

Die Übersichtlichkeit des Projektes sollte durch das Aufräumen gesteigert werden. Diese Anforderung ist nicht Pflicht, sie beugen jedoch späteren Fehlerquellen vor.

Aufgabe 1 - Model (25P)

```
package de.uniks.pmws1920.builder;
public class ModelBuilder {

    private PropertyChangeSupport listeners = null;
    private ArrayList<Player> setupPlayer = new ArrayList<>();

    // Get, Create or Update
    public Player buildPlayer(String name, String color) {
        // see if person with name is already in list
        // If person exists, update
        // Else create person and add to list
        // fire property change
        // return person
    }

    // Remove
    public void removePlayer(Player player) {
        // Delete player if in list
        // fire property change
    }

    // ================================================================================
    // This is the Code for PropertyChange-Handling
    // ================================================================================

    private boolean firePropertyChange(String propertyName, Object oldValue,
                                       Object newValue) {
        if (listeners != null) {
            listeners.firePropertyChange(propertyName, oldValue, newValue);
            return true;
        }
        return false;
    }

    public boolean addPropertyChangeListener(PropertyChangeListener listener) {
        if (listeners == null) {
            listeners = new PropertyChangeSupport(this);
        }
        listeners.addPropertyChangeListener("setupPlayer", listener);
        return true;
    }

    public boolean removePropertyChangeListener(PropertyChangeListener listener) {
        if (listeners != null) {
            listeners.removePropertyChangeListener("setupPlayer", listener);
        }
        return true;
    }
}
```

In dieser Aufgabe soll die in der Vorlesung vorgestellte Model-Klasse erstellt werden. Wir nennen diese Klasse ModelBuilder.java. Der oben gezeigte Code darf als Starthilfe kopiert werden. Speichere die Klasse unter src/main/java/de/uniks/pmws1920/builder.

Alle schreibenden Zugriffe auf das Datenmodel oder ein Element daraus müssen über den Modelbuilder geschehen.

Füge die Implementierung der leeren Methodenrümpfe hinzu. Die buildPlayer-Methode soll einen Spieler entweder anlegen oder aktualisieren und anschließend zurückgeben. Der Name des Spielers dient dabei als Identifizierungsmerkmal. Es kann somit keine zwei Spieler geben, die den selben Namen haben. Die removePlayer-Methode entfernt den Player sowohl aus der Liste, als auch von sämtlichen anderen instanziierten Objekten.

Bei der Bewertung wird vor allem auf die Funktionsweise des Modelbuilders geachtet.

Markiert den letzten Commit für diese Teilaufgabe der Konvention entsprechend.

Aufgabe 2 - View (25P)

In dieser Hausaufgabe setzen wir den aus Hausaufgabe 5 bekannten Setup Screen mithilfe des Scenebuilders um. Hierzu erstellen wir zunächst eine View-Datei, die erst in der Folgeaufgabe mit einer Controller-Klasse verbunden wird. Nach diesem Model-View-Controller-Pattern werden im Folgenden alle Ansichten des Programmes sowie deren Subelemente organisiert.

2.1 SetupView

Abbildung 1: Mockup Setup Screen

Setzt das in Abbildung 1 gezeigte Mockup mit dem Scenenbuilder um. Speichere die fxml-Datei unter src/main/resources/de/uniks/pmws1920 mit dem Namen SetupView.fxml ab.

Es wird empfohlen als Root-Element der View eine VBox zu wählen. Der Bereich zwischen Eingabe und Startbutton kann ebenfalls durch eine weitere VBox dargestellt werden. Dieses Vorgehen vereinfacht die Arbeit mit der folgenden Component.

2.2 PlayerRowComponent

Wir bezeichnen Subelemente einer View mit dem Begriff Component. Spezielle Subelemente wie die PlayerRowComponent, werden häufig kopiert oder gelöscht, daher macht es Sinn, diese in eine eigene fxml-Datei auszulagern.

Abbildung 2: Mockup für das PlayerRow-Element

Setzt das in Abbildung 2 gezeigte Mockup mit dem Scenenbuilder um. Speichere die fxml-Datei unter src/main/resources/de/uniks/pmws1920/controller/setup unter dem Namen PlayerRowComponent.fxml.

Es wird empfohlen, als Root-Element der Component eine HBox zu wählen. Das Mockup zeigt einen Kreis, der die Farbe des Players indizieren soll. Dieses Design ist lediglich ein Vorschlag. Es steht jedem frei, eine eigene Darstellungsform der PlayerRowComponent zu erstellen. Diese muss lediglich den Namen und die Farbe des Spielers anzeigen und eine Möglichkeit bieten, den Spieler aus der Liste zu entfernen.

2.3 InGameView

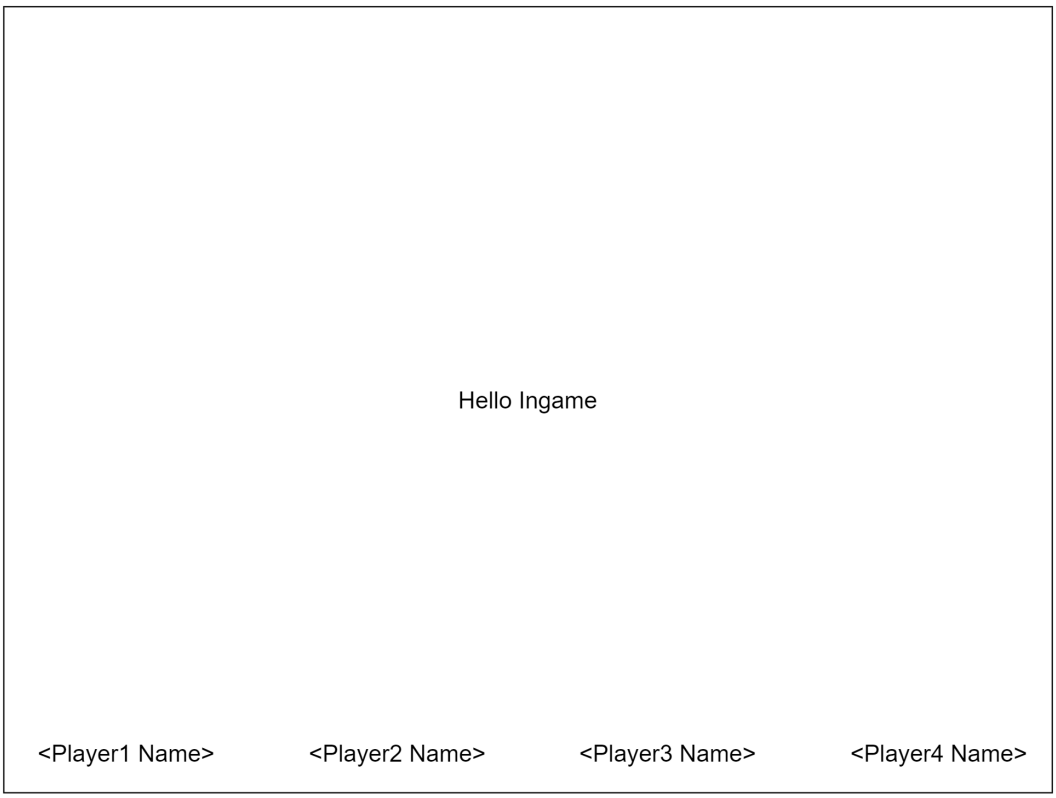

Abbildung 3: Mockup Ingame Screen

Setze das in Abbildung 3 gezeigte Mockup mit dem Scenenbuilder um. Speichere die fxml-Datei unter src/main/resources/de/uniks/pmws1920 mit dem Namen IngameView.fxml ab.

Die Anzeige der Spieler am unteren Rand der View richtet sich nach der Anzahl der Spieler, welche im Setup Screen erstellt wurden. An dieser Stelle ist es erlaubt, diese durch das Hinzufügen von Labeln im Code der zugehörigen Controller-Klasse zu realisieren.

Aufgabe 3 - Controller (25P)

Die Unteraufgaben dieser Aufgabe dürfen in beliebiger Reihenfolge abgearbeitet werden, da die Controller stark zusammenhängen. Es dürfen während dieser Aufgabe ebenfalls Änderungen/Anpassungen an den zuvor erstellten fxml-Dateien vorgenommen werden, um diese mit den Controllern und Feldern zu verknüpfen.

3.1 Umbenennen

Benennt die Setupscreen-Klasse aus Hausaufgabe 6 zu ShroomWars um. Geht im Nachhinein sicher, dass auch nach der Umbenennung euer Projekt noch ausführbar ist.

```
package de.uniks.pmws1920;

public class ShroomWars {

    private ModelBuilder builder;
    private Stage stage;
    private Scene scene;
    // it's not allowed to assign any controller to a global variable here

    @Override
    public void start(Stage primaryStage) throws Exception {
        // some code
        // call showSetup()
    }

    public void showSetup() {
        // show Setup Screen
    }

    public void showIngame() {
        // show Ingame Screen
    }
}
```

Der oben gezeigte Code darf als Starthilfe kopiert werden.

Die ShroomWars-Klasse wird im folgenden die Organisation der verschieden Ansichten gewährleisten. Jede Ansicht wird dabei über eine eigene Methode innerhalb der Klasse aufgerufen und ersetzt die vorangegangene Ansicht (wenn vorhanden). Dabei darf der jeweilige Controller der Ansicht **nicht** auf globaler Ebene in der ShroomWars-Klasse referenziert werden. Der Wechsel der Ansichten soll über die Runnable-Variablen der SubController realisiert werden.

3.2 SetupController

```
public class SetupController {

    @FXML
    public VBox playerBox;

    @FXML
    public ColorPicker colorPicker;

    @FXML
    public TextField textField;

    // Callback that can be triggered by change.run()
    private Runnable change; // TODO Generate setter and getter yourself

    private ArrayList<Player> players = new ArrayList<>(); // No setter or getter

    // ================================================================================
    // You must use these two methods
    // ================================================================================

    public void setBuilder(ModelBuilder builder) {
        this.builder = builder;
        this.getBuilder().addPropertyChangeListener(this::handlePlayerListChange);
    }

    private void handlePlayerListChange(PropertyChangeEvent event) {
        // Get new player list out of event
        // Set new list to global field
        // Render everything linked to the player list again
    }

    // ================================================================================
    // Linked Methods
    // ================================================================================

    @FXML
    public void addPlayer() {
        // call builder
    }

    @FXML
    public void startGame() {
        // change view
    }
}
```

Der oben gezeigte Code darf als Starthilfe kopiert werden. Speichere die Klasse unter src/main/java/de/uniks/pmws1920/controller/setup.

Es soll möglich sein, durch die Eingabe von Daten in die referenzierten Felder und das anschließende Klicken auf den "Add Player"-Button einen Spieler zum Model mithilfe der ModelBuilder-Klasse hinzuzufügen. Durch das Klicken auf den "Start Game"-Button soll die View ausgetauscht werden, wenn mindestens 2 Spieler angelegt wurden.

3.3 PlayerRowController

```
package de.uniks.pmws1920.controller.setup;

public class  PlayerRowController {
    @FXML
    public HBox root;
    @FXML
    public Pane playerColorPane;
    @FXML
    public Label playerNameLabel;

    private  Player player;

    // ========================================================================================
    // You must use these three methods
    // ========================================================================================

    public void setBuilder(ModelBuilder builder) {
        this.builder = builder;
    }

    public void setPlayer(Player player)  {
        this.player = player;
        this.renderComponent(this.player);
    }

    private  void renderComponent(Player player) {
        // Put code here
    }

    // ========================================================================================
    // Linked Methods
    // ========================================================================================

    @FXML
    public void removeRow() {
        // remove player of this  controller  from the model
    }
}
```

Der oben gezeigte Code darf als Starthilfe kopiert werden. Speichere die Klasse unter src/main/java/de/uniks/pmws1920/controller/setup.

Durch das Klicken auf den "Remove"-Button einer Zeile soll der Player aus dem Model entfernt werden und somit nicht mehr in der View angezeigt werden.

3.4 IngameController

```
public class SetupController {

    private ArrayList<Player> players = new ArrayList<>(); // No setter or getter

    // ================================================================================
    // You must use this method
    // ================================================================================

    public void setBuilder(ModelBuilder builder) {
        this.builder = builder;
        // Get player list
        // render view with labels
    }

}
```

Der oben gezeigte Code darf als Starthilfe kopiert werden. Speichere die Klasse unter src/main/java/de/uniks/pmws1920/ingame.

Bei der Bewertung wird vor allem auf die klare Abgrenzung der Funktionalitäten in Kombination mit deren View-Elementen geachtet.

Markiert den letzten Commit für diese Teilaufgabe der Konvention entsprechend.

Aufgabe 4 - Test (25P)

Als Abschluss für diese Hausaufgabe soll die Gesamtheit der implementierten Teile der vergangenen Teilaufgaben getestet werden. Dies geschieht wie in der Vorlesung gezeigt mithilfe der TestFx-Library. Füge die Abhängigkeit des Projektes zu der Library in die build.gradle-Datei hinzu.

```
dependencies {
    ...
    testCompile "org.testfx:testfx-junit:4.0.16-alpha"
}
```

4.1 SetupScreenTest

```
public class SetupScreenTest extends ApplicationTest {

    @Override
    public void start(Stage stage) {
        ShroomWars wars = new ShroomWars();
        wars.start(stage);
    }

    @Test
    public void addTwoPlayersTest() {
        // Add two players
        // Assert
    }

    @Test
    public void addTooManyPlayersTest() {
        // Only 4 players max.
        // Assert
    }

    @Test
    public void updateOnePlayerTest() {
        // Add player
        // Assert
        // Update player
        // Assert
    }

    @Test
    public void removeOnePlayerTest() {
        // Add player
        // Assert
        // Remove player
        // Assert
    }
}
```

Der oben gezeigte Code darf als Starthilfe kopiert werden. Speichere die Klasse unter src/test/java/de/uniks/pmws1920/controller/setup. Füge dann die Implementierung der leeren Methodenrümpfe hinzu. Schließe jeden Test mit mindestens einem sinnvollen Assert ab.

4.2 ChangeViewTest

```
public class ChangeViewTest extends ApplicationTest {

    @Override
    public void start(Stage stage) {
        ShroomWars wars = new ShroomWars();
        wars.start(stage);
    }

    @Test
    public void changeViewWithTwoPlayersTest() {
        // Add two players
        // Click start game
        // Assert in new view
    }
}
```

Der oben gezeigte Code darf als Starthilfe kopiert werden. Speichere die Klasse unter src/test/java/de/uniks/pmws1920/setup. Füge dann die Implementierung der leeren Methodenrümpfe hinzu. Schließe jeden Test mit mindestens einem sinnvollen Assert ab.

Bei der Bewertung wird vor allem auf das durchlaufen der Tests und die Asserts geachtet.

Markiert den letzten Commit für diese Hausaufgabe der Konvention entsprechend.

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.

JavaFX

- Scenebuilder: `https://gluonhq.com/products/scene-builder/`