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 23.01.2020 - 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!

Aufgabe 1 - ModelBuilder (33P)

In der vergangenen Hausaufgabe wurde der Name eines Players genutzt, um diesen zu identifizieren. Da wir den ModelBuilder in der folgenden Aufgabe um die Verwaltung von Shrooms und Houses erweitern, ist diese Art der Zuordnung nicht mehr ausreichend, weshalb IDs eingeführt werden sollen.

- Erweitert die Klassen Player und GameObject um ein ID Attribut. Der Datentyp steht euch frei. Wählt beispielsweise einen Integer, der bei jedem Erstellen einer Entity hochgezählt wird. (Oder setzt euch alternativ mit den UUIDs aus dem Anhang auseinander.)
- Setzt alle build<Entity> und remove<Entity> wie in Hausaufgabe 7 um, mit dem Unterschied, dass die IDs für die build<Entity> übergeben werden. Der unten gezeigte Code darf als Starthilfe kopiert werden. (Die ArrayLists für die Entities dürfen nach eigenem Ermessen durch andere Collection-Typen ersetzt werden)
- Überlegt euch eigenständig wie ihr die build<Entity> so aufbaut, dass sie alle 3 Funktionalitäten (Create, Get, Update) erfüllen. Ihr könntet beispielsweise bei dem übergeben einer ID ohne weitere Parameter nur die Entity zurückgeben, und so ein Get simulieren.
- Implementiert die initGame()-Methode wie im Codesnippet angegeben (Das Mockup in Abbildung 1 zeigt die Initialisierung mit 4 Playern). Das zurückgegebene War-Objekt soll im IngameController abgelegt werden.

```java
package de.uniks.pmws1920.builder;

public class ModelBuilder {

    private PropertyChangeSupport listeners = null;

    private War war = new War();
    private ArrayList<Player> playerList = new ArrayList<>();
    private ArrayList<House> houseList = new ArrayList<>();
    private ArrayList<Shroom> shroomList = new ArrayList<>();

    // ========================================================================
    // Game Rule Methods
    // ========================================================================

    public War initGame() {
        // take player list
        // give each player a house (hp = 10, capacity = 10) they own
        // place 4 shrooms (hp = 10, atk = 2) for each player at their house
        // add the same amount of unconquered houses as players in the game e.g. 4 Players = 4 Unconquered Houses
        // Only use the build<Class>-Methods below to create Entities
        return this.war;
    }

    // ========================================================================
    // Player
    // ========================================================================

    // Get, Create or Update
    public Player buildPlayer(int id, String name, String color) {
    // Update, Create, Return and fire property change
    }

    // Remove
    public void removePlayer(Player player) {
        // Remove and fire property change
    }

    // ========================================================================
    // Shroom
    // ========================================================================

    // Get, Create or Update
    public Shroom buildShroom(int id, int ownerId, int targetId, int hp, int attackValue) {
    // Update, Create, Return and fire property change
    }

    // Remove
    public void removeShroom(Shroom shroom) {
        // Remove and fire property change
    }

    // ========================================================================
```

```
    // House
    // ================================================================================

    // Get, Create or Update
    public House buildHouse(int id, int ownerId, int hp, int capacity) {
        // Update, Create, Return and fire property change
    }

    // Remove
    public void removeHouse(House house) {
        // Remove and fire property change
    }

    // ================================================================================
    // PropertyChange-Handling
    // ================================================================================

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

Aufgabe 2 - Ingame (33P)

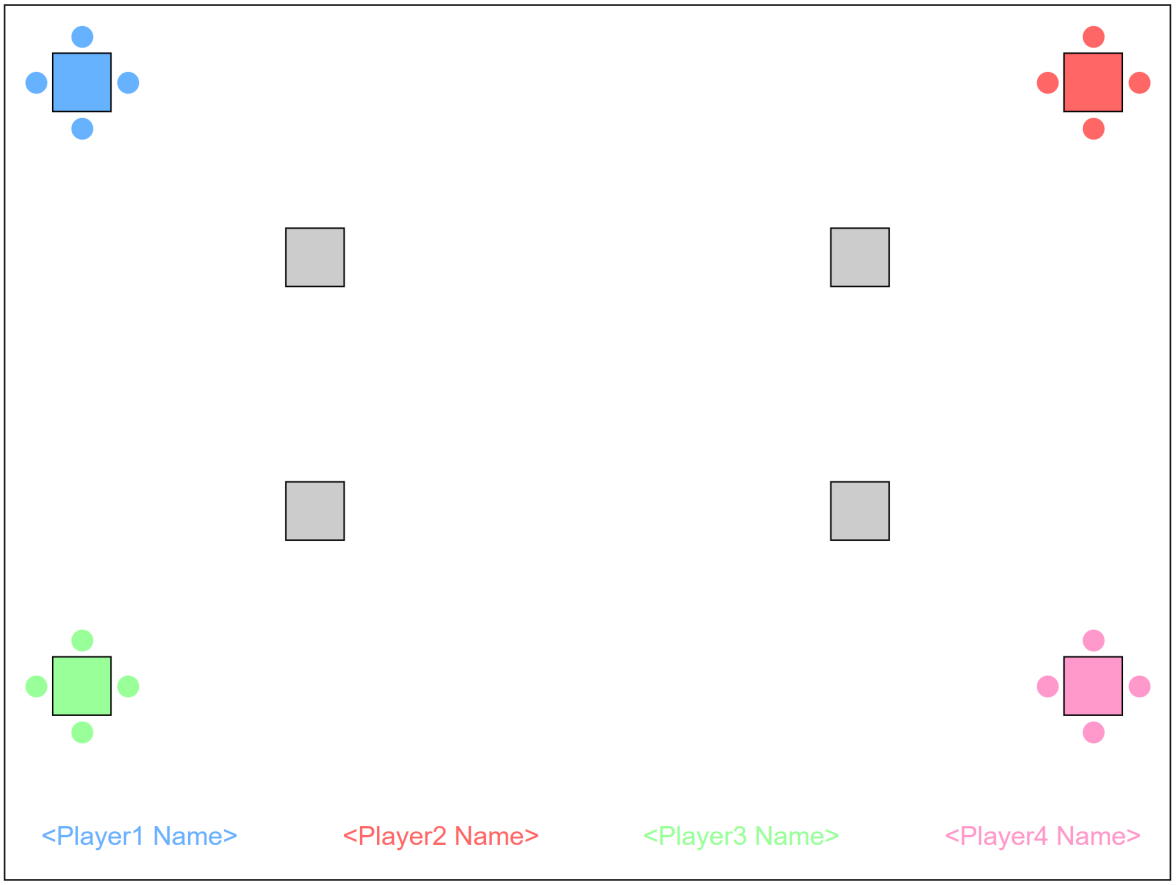

Abbildung 1: Beispiel einer Startsituation

In dieser Aufgabe soll sich mit der, in Zukunft maßgeblich benutzten, (Anchor)Pane von JavaFX auseinandergesetzt werden. Eine Pane kann genutzt werden um darauf zu ”malen”. Wie in Abbildung 1 dargestellt, können darauf geometrische Formen abgebildet werden, welche als Darstellung des Spielgeschehens genutzt werden können.

Die Aufgabe besteht nun darin, die abgebildete ”Zeichnung” zu replizieren.

Für diese Hausaufgabe gilt:
Die dargestellten Formen müssen weder mit einem eigenen Controller, noch mit dem Datenmodell verknüpft sein. Lediglich die Farben sollen aus der Spielerliste entnommen sein.

- Beginne damit, die initGame()-Methode aus Aufgabe 1 in der setBuilder()-Methode des IngameControllers (aus Hausaufgabe 7) aufzurufen.
- Rufe danach die placeShapes()-Methode aus dem Codesnippet unten auf.
- Füge darin den Code ein, der die Abbildung 1 repliziert. Die geometrischen Formen dürfen dabei gerne abweichen, sollten jedoch die Unterscheidbarkeit zwischen besetzten/nicht besetzten Houses sowie Shrooms gewährleisten.
- Denke daran, die Farben aus den Player-Objekten für die besetzten Houses zu verwenden.
- Färbe auch die Player-Label entsprechend der Farben im Datenmodell.

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

public class  IngameController {

    @FXML
    public AnchorPane pane;

    public void placeShapes() {
        // put code here
    }
}
```

Anmerkung:
Der abgebildete IngameController wurde in der letzten Hausaufgabe im bereitgestellten Codesnippet fälschlicherweise als SetupController angegeben. In dieser Hausaufgabe soll die IngameView mit dem IngameController verknüpft sein.

Aufgabe 3 - Test (34P)

Als Abschluss für diese Hausaufgabe soll die Nutzung von TestFx noch einmal wiederholt und verinnerlicht werden. Die unten gezeigten Codesnippets dürfen als Starthilfe kopiert werden.

```
public class IngameTest extends ApplicationTest {

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

    @Test
    public void playerLabelColorTest() {
        // Add some player
        // Change to the IngameView
        // Assert player label color
    }
}
```

```
public class ModelBuilderTest {

    @Test
    public void initModelTest() {
        // put code here
    }

    // ================================================

    @Test
    public void buildPlayerCreateTest() {
        // create a player via modelbuilder
        // Assert
    }

    @Test
    public void buildPlayerGetTest() {
        // create a player via modelbuilder
        // try to create the same player again
        // Assert
    }

    @Test
    public void buildPlayerUpdateTest() {
        // create a player via modelbuilder
        // update a player via modelbuilder
```

```
        // Assert
    }

    // ================================================

    @Test
    public void buildHouseCreateTest() {
        // create a house via modelbuilder
        // Assert
    }

    @Test
    public void buildHouseGetTest() {
        // create a house via modelbuilder
        // try to create the same house again
        // Assert
    }

    @Test
    public void buildHouseUpdateTest() {
        // create a house via modelbuilder
        // update a house via modelbuilder
        // Assert
    }

    // ================================================

    @Test
    public void buildShroomCreateTest() {
        // create a shroom via modelbuilder
        // Assert
    }

    @Test
    public void buildShroomGetTest() {
        // create a shroom via modelbuilder
        // try to create the same shroom again
        // Assert
    }

    @Test
    public void buildShroomUpdateTest() {
        // create a shroom via modelbuilder
        // update a shroom via modelbuilder
        // Assert
    }
}
```

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

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.

UUID

- Was ist eine UUID?
  `https://de.wikipedia.org/wiki/Universally_Unique_Identifier`
- UUID Java Dokumentation
  `https://docs.oracle.com/javase/7/docs/api/java/util/UUID.html`