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 06.02.2020 - 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.

Aufgabe 1 - Save and Load (45P)

Für die letzte Aufgabe dieser Hausaufgabe:

```
dependencies {
    ...
    compile group: 'org.fulib', name: 'fulibYaml', version: '1.0.+'
    ...
}
```

Jede build<Entity>-Methode soll beim Aufruf eine repräsentative Nachricht als String erstellen und diese sollen wie in der Vorlesung gezeigt, in eine .yaml-Datei abgelegt werden. Diese Datei soll unter dem Ordner savegames abgelegt werden, und den Namen save<YOUR_MATRICULAR_NUMBER>.yaml haben. Somit ergibt sich der Pfad ./savegames/save<YOUR_MATRICULAR_NUMBER>.yaml

Bei dem Test ist es egal, ob das existierende Savegame gelöscht wird.

Abbildung 1: Setup mit Load-Button

Beachtet, dass je nachdem welcher Button (load oder save) geklickt wird, eine andere Methode im ModelBuilder ausgeführt werden muss, um das Spielgeschehen aufzubauen. Da die Stelle, an der die initGame()-Methode gerufen wird, in den bisherigen Abgaben teils stark unterscheidet, muss je nach Situation eine Möglichkeit gefunden werden, die jeweils richtige Methode auszuführen.

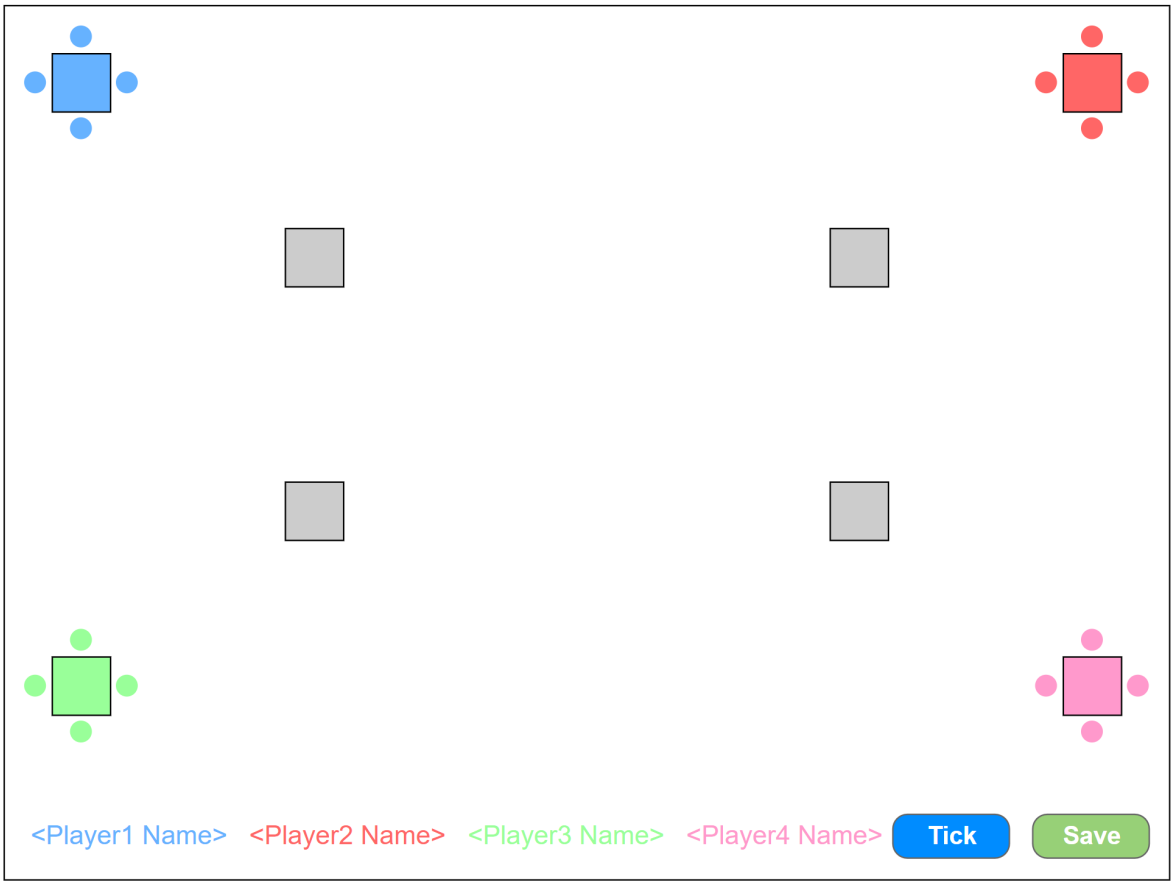

Abbildung 2: Ingame mit Save-Button

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

public class ModelBuilder {

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

    public War loadHistory(){
        // load date in savefile as shown in the lecture
        // return the war object (that is not persisted) to make the initGame method
        // interchangeable with this method
    }

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

    public Player persistPlayer(int id, String name, String color){
        // this must be called on every buildPlayer call
        // put code for Player persisting here
    }

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

    public Shroom persistShroom(int id, int ownerId, int targetId, int hp, int attackValue){
        // this must be called on every buildShroom call
        // put code for Shroom persisting here
    }

    // ========================================================================
    // House
    // ========================================================================
    public House persistHouse(int id, int ownerId, int hp, int capacity){
        // this must be called on every buildHouse call
        // put code for House persisting here
    }
}
```

Abbildung 3: IngameController

Aufgabe 2 - Move (45P)

Nach der Implementierung der reinforce()-Methode der letzten Hausaufgabe, soll in dieser Woche die Bewegung der Einheiten umgesetzt werden. Da für die Realisierung sowohl das Modell, als auch die Controller gehandhabt werden müssen, wird eine Hilfsklasse benötigt, die wir MovementHandler nennen. Wie in Codesnippet 6 zu sehen ist, ermöglicht dieser durch die Referenzierung zweier HouseController, sohwohl die Manipulation der View, als auch die des Modells.

Das Ziel der Aufgabe ist es, bei dem Nacheinander-Auswählen zweier Houses zwischen diesen eine Bewegungsroute einzurichten. Damit ist gemeint, dass nach dem Erstellen einer dieser Verbindungen zweier Häuser durch einen MovementHandler bei jedem Tick eine bestimmte Anzahl an Shrooms von einem Haus zu dem Zielhaus wechseln. Dies geschieht so lange, bis diese Bewegungsroute / Verbindung manuell entfernt wird. Diese Routen dienen also als kontinuierliche Angriffsrouten.

Um dies zu bewerkstelligen, muss der IngameController eine Liste über die bereits angelegten Angriffsrouten (MovementHandler) führen und das Anlegen und Entfernen eben jener ermöglichen. Fügt die unten stehenden Methoden/ Kommentare zu eurem Projekt hinzu und implementiert deren Methodenrümpfe.

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

public class IngameController {

    private <ANY_LIST_TYPE_YOU_WANT> movementList;

    @FXML
    public void nextTick() {
        // Reinforce Houses
        this.getBuilder().reinforce();

        // call handleMovement() for every MovementHandler
        // Remove MovementHandler if canBeRemoved() return true;
    }

    // ========================================================================
    // Logic
    // ========================================================================

    private void handleHouseSelect(HouseController houseController) {
        // save first selected as source house

        // if next selected house is different than the sourceHouse

        // if no MovementHandler for these two houseController exist
        // create a MovementHandler and hand in the two houseController plus the pane
        // else MovementHandler for these two houseController already exist, call markToBeRemoved


        // deselect houses and reset references for new selection
    }

}
```

Abbildung 4: IngameController

Desweiteren muss der IngameController beim Klick auf ein House benachrichtigt werden, um im Folgenden die Orchestrierung der MovementHandler zu bewerkstelligen. Wie unten zu sehen ist, nutzen wir für diese, als Callback implementierte Benachrichtigung des IngameControllers eine Klasse mit dem Namen Consumer. Diese stellt eine Erweiterung des bereits bekannten Runnables dar, mit dem Unterschied, dass ein Consumer ein Objekt als Parameter übergeben bekommt.

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

public class HouseController {

    private Consumer<HouseController> houseSelectedCallback;

    // ================================================================================
    // Init
    // ================================================================================
    public void setContent(ModelBuilder builder, AnchorPane pane, House house, double xPos, double yPos,
                           Consumer<HouseController> houseSelectedCallback) {
        // this is new
        this.houseSelectedCallback = houseSelectedCallback;
    }

    // ================================================================================
    // Logic
    // ================================================================================

    public ShroomController extractOneShroom() {
        // return first shroomController out if the list if list is not empty
    }

    public void handleEnemyArrives(ArrayList<ShroomController> shrooms) {
        // Add all shroomController to list
    }


    private void handleSelected() {
        this.houseSelectedCallback.accept(this);
    }


    @FXML
    public void onClicked() {
        // here should be some code where you highlight the houseController
        // handle selected should be called after the highlighting. Otherwise some strange side cases will happen
        this.handleSelected();
    }

    // I recommend to write some public methods one that, when called, reset the visual selection of the house and
    // one that sets the visual selection. This makes the manipulation build up in the handleHouseSelect of the
    // IngameController easier

}
```

Abbildung 5: HouseController

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

public class MovementHandler {

    private <YOUR_SHAPE> <YOUR_SHAPE_VARIABLE>;

    private int amount;
    private AnchorPane pane;
    private HouseController sourceHouse;
    private HouseController targetHouse;

    // ============================================================================
    // Init
    // ============================================================================

    // this constructor should be used for this homework
    // this constructor is built to call the constructor below itself.
    // Is is used to lock the number of shrooms that will be moved by the constant number passed to the constructor
    // below
    public MovementHandler(HouseController sourceHouse, HouseController targetHouse, AnchorPane pane) {
        this(sourceHouse, targetHouse, pane, <FIXED_NUMBER_OF_SHROOMS_TO_MOVE_PER_TICK>);
    }

    public MovementHandler(HouseController sourceHouse, HouseController targetHouse, AnchorPane pane,
                           int amount) {
        this.sourceHouse = sourceHouse;
        this.targetHouse = targetHouse;
        this.pane = pane;
        this.amount = amount;

        this.setup();
    }


    private void setup() {
        this.placeSelf();
    }


    // ============================================================================
    // Init View
    // ============================================================================

    private void placeSelf() {
        // choose a shape to visualize the set movement
        // place on pane
    }

    // ============================================================================
    // Logic
    // ============================================================================

    public void handleMovement() {
        // extract amount of shroomsController from sourceHouseController
        // add all shroomsController to targetHouseController

        // use ModelBuilder to set the new target of the shrooms in the extracted shroomController
        // When the model is altered after the controllers where shifted, the property change listener constructed in
        // the last homework will display the arrived shroom correctly.

        // if this Handler is marked to be removed, call removeYou here
    }
```

```
    public void markToBeRemoved() {
        // show visually that this Handler will be removed (in my case, paint the line red)
        // choose a way to mark this instance to be removed after the next handleMovement() call
    }

    public void removeYou() {
        // look at the fulib generated removeYou()-Methods
        // this method should cut of all references to other instances
        // don't forget to remove the used shape from the pane

    }

    public boolean canBeRemoved() {
        // return if marked as to be removed
    }

    // ========================================================================================
    // Helping Methods
    // ========================================================================================

    // feel free to write some helping methods
}
```

Abbildung 6: MovementHandler

Aufgabe 3 - GameLoop (10P)

Wir führen in dieser Hausaufgabe den seit Beginn der Veranstaltung beschriebenen Gameloop ein. Dieser wird durch die Timer-Klasse umgesetzt und führt die spezifizierte Methode alle, in unserem Fall, 3 Sekunden aus.

An dem unten stehenden Code muss nichts geändert werden, da er bereits voll funktionsfähig ist. Allerdings können durch die Automatisierungen neuartige Fehler auftreten. Diese beruhen auf der parallelen Ausführung von Oberflächen und restlicher Logik die von JavaFx vorgegeben ist. Um die auftretenden Fehler zu beheben, müssen die Anweisungen an die Oberfläche durch ein Platform.runLater() gekapselt werden.

Die Aufgabe gilt als korrekt gelöst, wenn beim Testen keine Exceptions auftreten.

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

public class  IngameController {

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

    // ================================================================================
    // Fake Ticks
    // ================================================================================

    public void startLoop() {
        Timer timer = new Timer();
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                nextTick();
            }
        }, 0, 3000);
    }
}
```

Abbildung 7: IngameController