Point‘n‘Click Adventure

Natalie Hußfeldt

Sandra Herbst

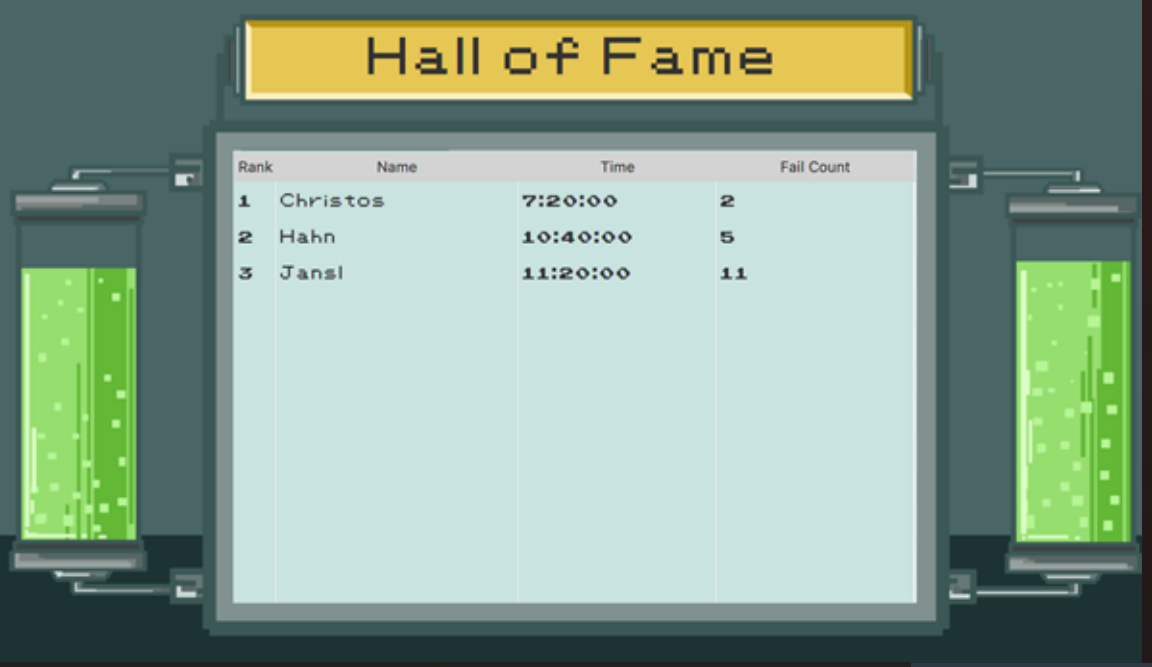
Jana Mößner

Anh Thu Bui

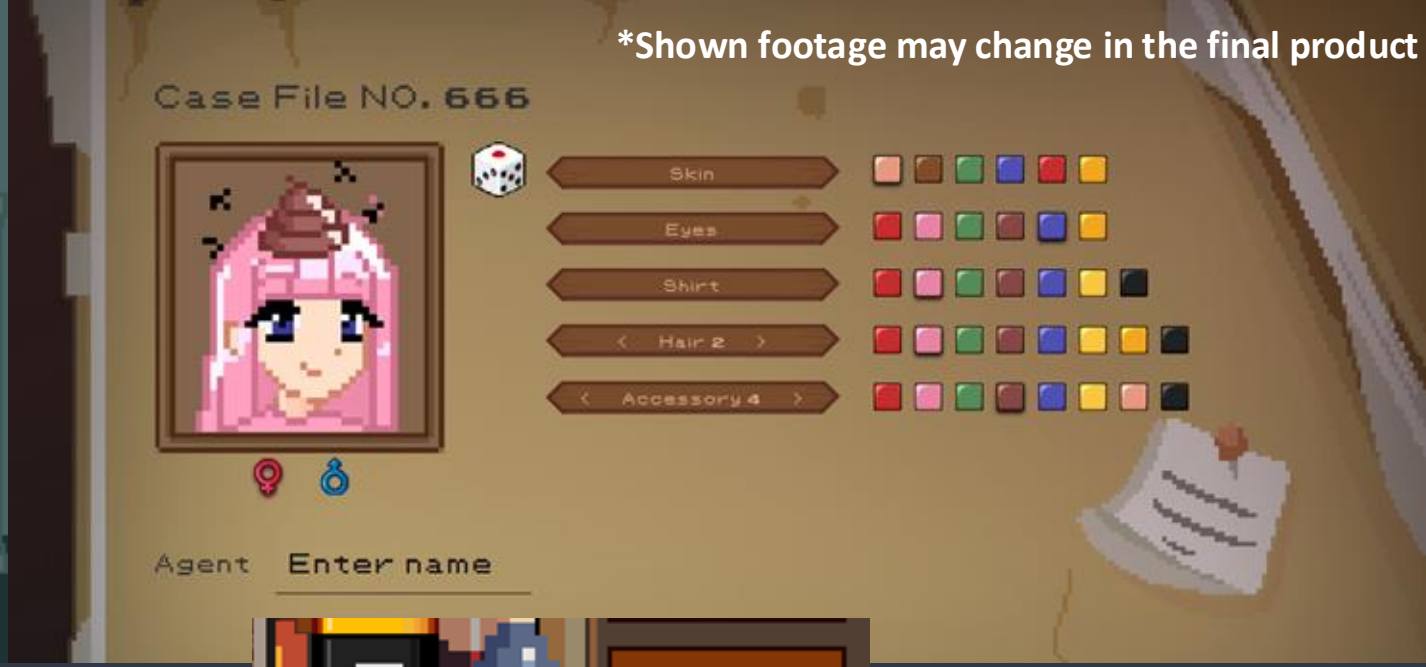
Lea Bretz

Strukturierung

- Dependency Injection
- Character Creation Tool
- Settings
- Highscore
- HTTP Response (Server)



*Shown footage may change in the final product



Dependency Injection

Ziel: Abhängigkeiten zwischen Klassen minimieren

Beispiel

Problem

- Klasse ist extrem statisch und **abhängig** von der Instanziierung des Hundes

Lösung:

- Instanziierung des Hundes **auslagern** und in Zielklasse *“injecten”*

  → Constructor Injection

```
public class Tierpfleger {
    private Hund hund;

    public Tierpfleger(){
        hund = new Hund("Bissi");
    }
};
```

```
public class Tierpfleger {
    private Hund hund;

    public Tierpfleger(Hund hund){
       this.hund = hund;
    }
};

class DependencyInjection{
    public static void main(String[] args) {
        Hund hund = new Hund("Bello");
        Tierpfleger Hugo = new Tierpfleger(hund);
    }
}
```

Dependency Injection - Spring Framework

- **Spring Framework** übernimmt die Instanziierung der Objekte
- Managed den gesamten Lebenszyklus der Objekte

→ **Inversion of Control**

- “Blueprints” werden in einer xml dokumentiert
- “inventoryBuilder” verwendet **Constructor Injection** und referenziert Objekte aus anderen Klassen

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>
    <bean id="inventoryManager" class="project.game.InventoryManager">
    </bean>
    <bean id="settingsManager" class="project.manage.SettingsManager">
    </bean>
    <bean id ="jsonTextManager" class="project.game.JSONTextManager">
    </bean>
    <bean id ="inventory" class="project.GUI.GridpaneBuilder">
    </bean>
    <bean id="inventoryBuilder" class="project.game.InventoryBuilder"
scope="prototype">
        <constructor-arg ref="inventory"/>
        <constructor-arg ref="player"/>
    </bean>

    <bean id="player" class="project.game.PlayerFactory" factory-
method="getPlayer">
        <constructor-arg type="java.lang.String" value="playerData"/>
    </bean>
```

Dependency Injection - Spring Framework

```
public void initPlayer(){
    context = new
ClassPathXmlApplicationContext("file:src/main/resources/spring/spring.xml")
;
    player = (Player) context.getBean("player");
}
```

- **ApplicationContext** ist wie eine Art “BeanFactory”, welches die Beans aus der xml erstellt
- dieser wird nur in der Main erstellt und den anderen Klassen übergeben, da sonst bei einer wiederholten Instanziierung die xml noch einmal gelesen wird
- anhand der angegebenen **ID** können spezifische Objekte extrahiert und verwendet werden

drawCharacter()

```
//BufferedImage to draw the new character.png
BufferedImage newImg = new BufferedImage( width: 200,  height: 250, BufferedImage.TYPE_INT_ARGB);
```

BufferedImage

TYPE_INT_ARGB :

•Repräsentiert ein Bild mit 8-bit RGBA (red,green,blue,alpha) Farbkomponenten in Integer Pixeln

Alphakanal:

•Angaben über Transparenz

```
//create graphics of the image so different graphics can be drawn onto it
Graphics2D g2d = newImg.createGraphics();
```

- Zeichnen in das BufferedImage mit Graphics-Objekt

drawCharacter()

- (x,y) linke obere Ecke im Koordinatenraum dieses Grafikkontexts gezeichnet

```
g2d.drawImage(convertFXToAWTImage(skin.getImage()), x: 0, y: 0, observer: null);
g2d.drawImage(convertFXToAWTImage(shirt.getImage()), x: 0, y: 0, observer: null);
g2d.drawImage(convertFXToAWTImage(hair.getImage()), x: 0, y: 0, observer: null);
g2d.drawImage(convertFXToAWTImage(eyes.getImage()), x: 0, y: 0, observer: null);
g2d.drawImage(convertFXToAWTImage(accessory.getImage()), x: 0, y: 0, observer: null);
g2d.dispose();
```

- saveImageToFile: erstellt neues Image von Typ png und speichert es in die angegebene Resource

```
//create the new character.png
saveImageToFile(newImg, type: "png", dst: "src/main/resources/character.png");
log.debug( s: "New character image has been created.");
```

getFilteredImage()

RGBImageFilter:
einfache Möglichkeit, einen ImageFilter zu erstellen, der die Pixel eines Bildes im Standard-RGB ColorModel ändert

z.B Int red (rgb & 0xff0000) >> 16;
bringt uns quasi an den Speicherbereich des Rotwerts

```
ImageFilter filter = (RGBImageFilter) (x, y, rgb) → {
        int alpha = (rgb & 0xff000000);
        int red = (rgb & 0xff0000) >> 16;
        int green = (rgb & 0x00ff00) >> 8;
        int blue = (rgb & 0x0000ff);

        // Avoids maximum of 255 (RGB Color values can never go over 255)
        // 0xff is 255 in Hex format, minimum is 0 (Range 0-255)
        // It will never filter black! (0,0,0)
        if (red != 0 && green != 0 && blue != 0){
            red = Math.max(0, Math.min(0xff, red + redIncrement));
            green = Math.max(0, Math.min(0xff, green + greenIncrement));
            blue = Math.max(0, Math.min(0xff, blue + blueIncrement));
        }

        return alpha | (red << 16) | (green << 8) | blue;
};
```

```java
/**
 * This method will convert a given java.awt.Image to javafx.scene.Image.
 * @param imgAWT that is being converted.
 * @return javafx Image.
 */
private Image convertAWTToFxImage(java.awt.Image imgAWT) {
    return SwingFXUtils.toFXImage((BufferedImage) imgAWT, writableImage: null);
}

/**
 * This method will convert a given javafx.scene.Image to java.awt.Image.
 * @param imgFX that is being converted.
 * @return javafx Image.
 */
private java.awt.Image convertFXToAWTImage(Image imgFX) { return SwingFXUtils.fromFXImage(imgFX, bufferedImage: null); }

//create awt Images of the chosen styles
public java.awt.Image getChosenImageAsAwtImage(String baseImage) throws IOException {
    return ImageIO.read(getClass().getResourceAsStream(baseImage));
}
```

ImageView kann nur ein JavaFX Image anzeigen lassen, die generierten BufferedImages und Images sind aber awt.Images

Lösung: **Konvertierungsmethoden**


Case File NO. 666
Skin
Eyes
Shirt
Hair 4
Accessory 1
Agent
Enter name

Case File NO. 666

Skin

Eyes

Shirt

< Hair 1 >

< Accessory 1 >

Agent Enter name

Back

Submit

Settings


Back
Settings
Language
EN
DE
Background Sound Volume
Sound Effect Volume
Theme
Dark
Light

Settings - Allgemeine Informationen

- Getter und Setter greifen ***nur auf Variablen des Players zu*** und werden auch von der Player Instanz gestellt
- In den Player Settern und Gettern ***werden die Getter und Setter der Manager Instanzen aufgerufen*** die er Implementiert
- Aktuelle Game Einstellungen sind sichtbar
- Beim Laden des Controllers wird die Aktuelle ***FXML URL*** mitgegeben um den Zugriff auf die Einstellungen vom Gesamten Spiel aus zu ermöglichen.

Settings - Switch Language And Theme

- Clicklistener auf ***Checkboxen***
- das momentane ***Stylesheet löschen*** und je nach Checkbox das neue hinzufügen
- Das momentane ***Resource Bundle austauschen*** (Szenenwechsel notwendig)

```
public void addChangeLanguageListeners(){
    checkEN.setOnMouseClicked(clickEvent -> {
        player.setLangSettings(Settings.language.ENGLISH);
        if(checkDE.isSelected()){
            checkDE.setSelected(false);
        }});

    checkDE.setOnMouseClicked(clickEvent -> {
        player.setLangSettings(Settings.language.GERMAN);
        if(checkEN.isSelected()){
            checkEN.setSelected(false);
        }});
}
```

Settings - Sound Settings

- Klasse SoundPlayer erstellt Objekt MediaPlayer
- addVolumePropertyListener auf bgSoundSlider und effectSoundSlider

- Getter und Setter ***kommen aus Klasse Player*** und setzen Werte in SettingsManager

```
public void updateSliderValues(){
    bgSoundSlider.setValue(player.getBGSoundVolume()*100);
    effectSoundSlider.setValue(player.getEffectSoundVolume()*100);
}

public void addVolumePropertyListener(){
    bgSoundSlider.valueProperty().addListener(observable -> {
        player.setBGSoundVolume(bgSoundSlider.getValue()/100);
        soundPlayer.updateVolumeSettings();
    });

    effectSoundSlider.valueProperty().addListener(observable -> {
        player.setEffectSoundVolume(effectSoundSlider.getValue()/100);
    });
}
```

Settings - Sound Settings

- SoundPlayer holt sich dann beim Aufrufen der Methoden playBackgroundSound und playSoundEffect die Werte von Player
- Diese Methoden erwarten nur eine URL zum Sound-File

```
public void updateVolumeSettings() { bgSound.setVolume(player.getBGSoundVolume()); }

public void playSoundEffect(String filename) {
    Media me = new Media(getClass().getResource( name: "/sound/" + filename + ".mp3").toString());
    MediaPlayer mp = new MediaPlayer(me);
    mp.setVolume(player.getEffectSoundVolume());
    mp.setAutoPlay(true);
    mp.play();
}
```

Highscore


Hall of Fame
Rank
Name
Time
Fail Count
1
Christos
7:20:00
2
2
Hahn
10:40:00
5
3
Jansl
11:20:00
11

Highscore - GET Request

```
public HttpResponse<String> httpGET() {
    try {
        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
                .GET()
                .header( name: "Authorization",  value: "Basic ZXNjYXBlOmphbnNs")
                .timeout(Duration.ofSeconds(10000))
                .header( name: "accept",  value: "application/json")
                .uri(URI.create(POSTS_API_URL))
                .build();
        return client.send(request, HttpResponse.BodyHandlers.ofString());
    } catch (InterruptedException | IOException e) {
        log.fatal( s: "httpGET Request has failed: \n" + e.getMessage() + "\n");
        e.printStackTrace();
    }
    return null;
}
```

HTTPBuilder
macht requests an Server mit GET, POST, DELETE
parsed Player objekte zu JSON Objekten

Highscore - GETRequest

HighscoreThread

- implementiert Runnable
- erstellt HTTPBuilder Objekt und ruft damit die httpGET Methode auf, die PlayerObjekte zurückgibt
- HighscoreThread wird in der Main class gestartet

```
public void apiCall(){
    try {
        builder = new HTTPBuilder
                ( POSTS_API_URL: "https://escaperoom-game.herokuapp.com/players");
        players = builder.httpGET();
        data.addAll(builder.JSONtoPlayer(players));
        log.debug( s: players.body() + " data received");

    }catch(IOException e){
        log.error( s: "API call failed: " + e.getMessage());
    }
}

@Override
public void run() { apiCall(); }
```

Highscore - Daten anzeigen

HighscoreController

- in fxml ist TableView mit 4 Columns
- den Columns werden in HighscoreController die richtigen Properties zugewiesen

```
private void initializeTableColumns() {
    columnName.setCellValueFactory(new PropertyValueFactory<>( s: "name"));
    columnTime.setCellValueFactory(new PropertyValueFactory<>( s: "sumPlayTime"));
    columnFailCount.setCellValueFactory(new PropertyValueFactory<>( s: "failCounter"));
    setRank();
}
```

Highscore - Daten anzeigen

HighscoreController

- erste Spalte rank number wird nicht vom server zurückgegeben sondern im controller eingefügt mit index der row + 1
- Daten werden vom gestartetet thread in der Main geholt und der tableView übergeben
- Daten aus der TimeColumn werden nochmal durchgegangen um sie für einen User lesbar zu formatieren
  „PT10H040M“ → „10:40:00“

Highscore - Daten anzeigen

```
private void setTimeColumnDataFormatted() {
    columnTime.setCellFactory(new Callback<>() {
        @Override
        public TableCell<Player, String> call(TableColumn<Player, String> playerStringTableColumn) {
            return updateItem(playerEntry, empty) → {
                    super.updateItem(playerEntry, empty);
                    if (this.getTableRow() != null && playerEntry != null) {
                        setText(formatDuration(Duration.parse(this.getItem())));
                    } else {
                        setText("");
                    }
            };
        }
    });
}

private String formatDuration(Duration duration) {
    long s = duration.getSeconds();
    return String.format("%d:%02d:%02d", s / 3600, (s % 3600) / 60, (s % 60));
}
```

HTTP Response - Server

Spring Boot Framework – REST (**GET, POST**)
PostgreSQL Datenbank + JDBC
Heroku.com als kostenlosen Webhost (mit diversen **Nachteilen**)

GET – Top 10 Spieler.
POST – Speichert Spieler in der Datenbank des Servers ein.

Request Authorization Header benötigt Schlüssel für **Username und Passwort.**
Verhindert (etwas, aber nicht vollständig) Requests von Dritten.

HTTP Response - POST Response

Player JSON POST-Request vom Client, wenn Spiel beendet wurde:

```
{
  "name":"Christos",
  "endDate":"2019-11-09, 05:50:30",
  "startDate":"2019-10-05, 04:10:10",
  "sumPlayTime":"PT7H020M",
  "failCounter":2
}
```

LocalDateTime Objekt als String: 7 Stunden, 20 Minuten

Server Datenbank **speichert** Player in seine Tabelle ein

Millisekunden berechnet sich aus **sumPlayTime**, dient zur **Sortierung** der Bestspielzeit.

name	failCounter	sumPlayTime	startDate	endDate	totalMillisec
Christos	2	PT7H020M	2019-10-05, 04:10:10	2019-11-09, 05:50:30	26400000

Server – GET Response

```java
@GetMapping("/players")
public String player() throws SQLException {
    ResultSet rs = null;
    try {
        PostgreSQL.connect();
        rs = PostgreSQL.getAllPlayers( table: "player");
```

```java
public static ResultSet getAllPlayers(String table) {
    return onQuery(String.format("SELECT * FROM %s ORDER BY totalMillisec ASC LIMIT 10", table));
}
```

ResultSet fungiert als ***Cursor***, der durch alle Daten in der Tabelle looped, gibt hier **MAX 10** Zeilen aus.

Rank 1


```
[
{
  "endDate": "2020-12-10, 10:10:30",
  "name": "Christos",
  "sumPlayTime": "PT3H020M",
  "failCounter": 11,
  "startDate": "2010-10-02, 16:30:30"
},
```


Rank 2


```
{
  "endDate": "2020-11-05, 02:09:33",
  "name": "Hahn",
  "sumPlayTime": "PT10H040M",
  "failCounter": 11,
  "startDate": "2010-10-05, 10:39:35"
}
```


Player werden im Query nach der Spielzeit geordnet, weshalb die Top 10 Spieler bereits geordnet als Response beim Client ankommen.