

# **Releasebundle 2**

Präsentation zum 2. Review

# Gliederung

1. **Gesetzte Ziele für Release 2**
2. **Projektstruktur**
3. **GUI**
  1. Allgemein
  2. Aufteilung in Komponenten
4. **Neue Funktionen**
  1. Menüleiste
  2. Videoplayer
  3. Loganzeige

# 1. Gesetzte Ziele für Release 2

## **Kurz zum Vorprojekt:**

- Nichtfunktionale GUI steht
- Projekt-/Klassenstruktur erkennbar
- Gerüst fürs Handbuch angelegt

→ Auf Vorprojekt aufbauend nun erste Funktionalität







## **Gesetzte Ziele per Releaseplan:**

- Dialog für Log-/Videoauswahl
- Videos können abgespielt/pausiert werden
- Logdateien können geparkt und angezeigt werden

# 2. Projektstruktur

- Von Release 1 stark abgeändert

## Unterteilt in Pakete:

-  **config**: (Vom Nutzer) einstellbare Werte
  -  **scene**: Startpunkt der Anwendung; Init. der JavaFX-scene
  -  **menu**: Funktionalität der Menüleiste
  -  **videoplayer**: Funktionalität des Videoplayers
  -  **track**: setzen/bearbeiten/verarbeiten von Markierungen im Trackpane
  -  **log**: parsen/anzeigen/verarbeiten von Logeinträgen
- verbesserte Übersichtlichkeit; schnelleres Einlesen
- korrespondiert mit GUI-Komponenten

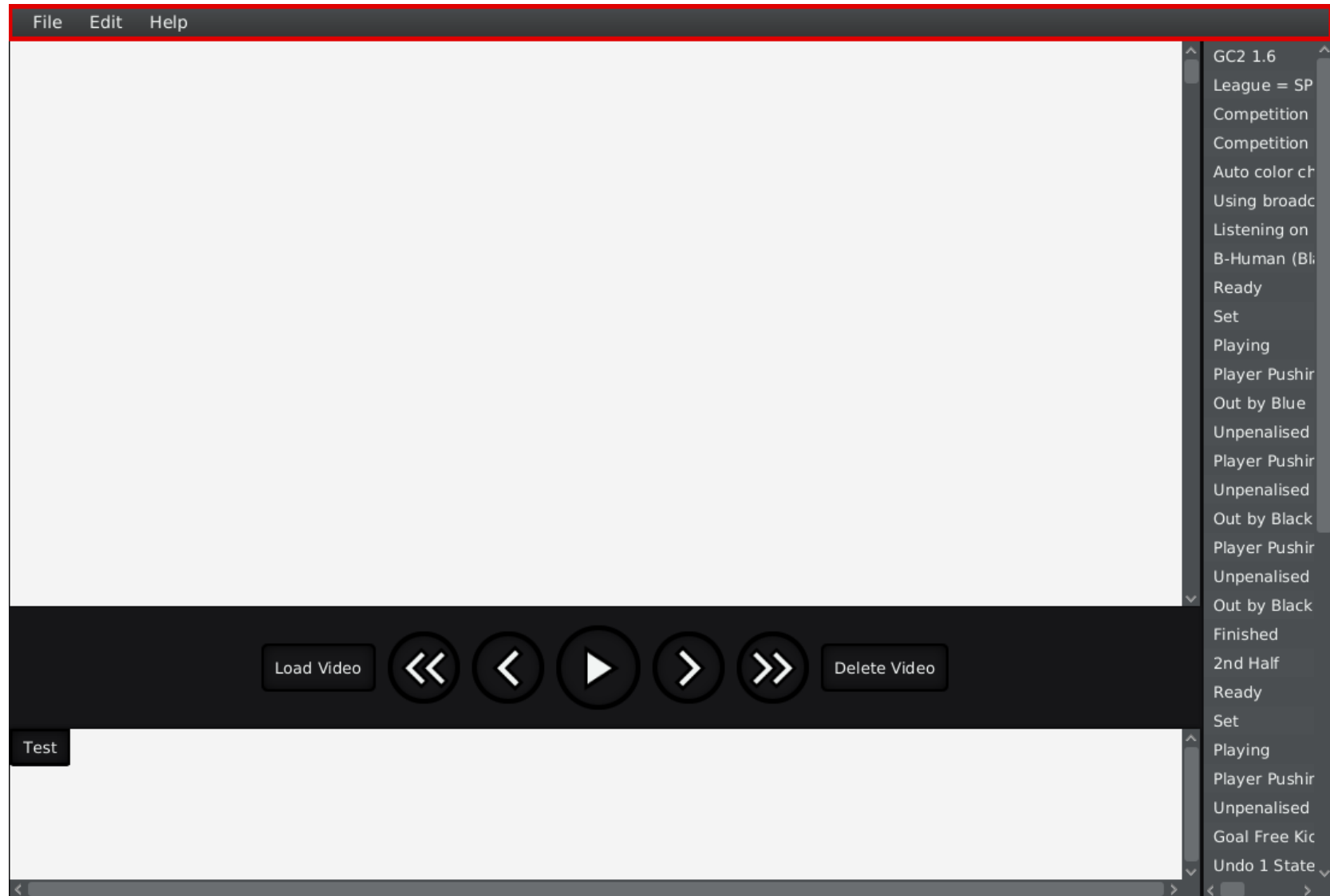
# 3.1 GUI - Allgemein

- realisiert mit *JavaFX*
- eigener Controller/FXML für jede Komponente  
→ übersichtliche Umsetzung von Funktionalität/Layout
- Komponenten werden in einer JFX-Scene zusammengefasst
- zusätzliche Einbindung von .css-Datei für ansprechenderes Design

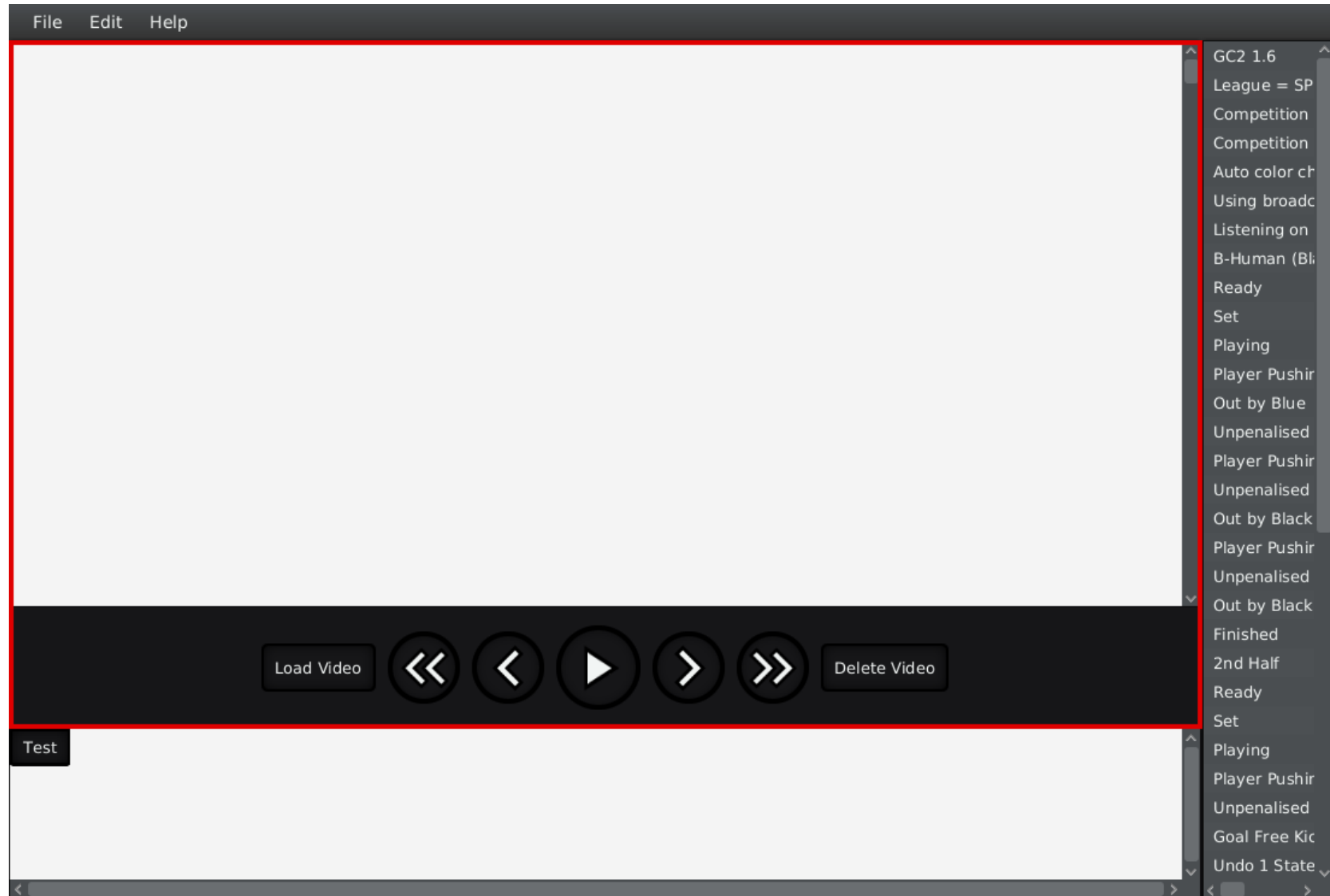
## 3.2 GUI – Aufteilung in Komponenten



## 3.2 GUI – Menüleiste

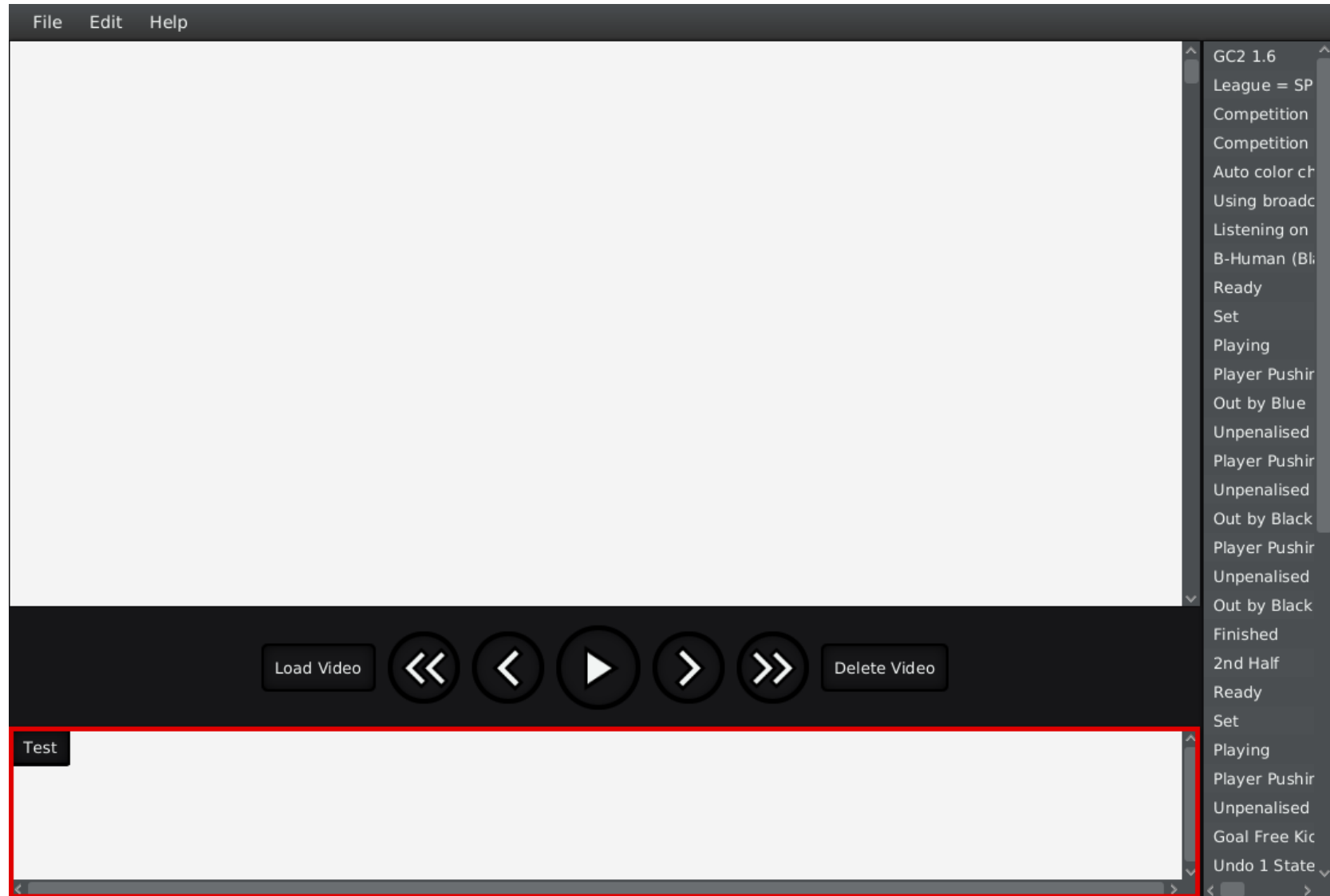


## 3.2 GUI – Videoplayer





## 3.2 GUI – Trackpane



## 3.2 GUI – Loganzeige



## 4. Neue Funktionen

- **Menüleiste:** Dialog zum Auswählen von Video-/Bild-/Logdatei
- **Videoplayer:** Abspielen/Pausieren von Videos
- **Loganzeige:** Anzeigen von Einträgen einer geparsten Logdatei

## 4. Neue Funktionen

**<Live Demo>**

# 4.1 Neue Funktionen - Menüleiste

- umgesetzt mit Filechooser:

```
private File openFile(String title, String fileType, List<String> extensions) {  
    FileChooser chooser = new FileChooser();  
    chooser.setTitle(title);  
  
    extensions.forEach(x -> chooser.getExtensionFilters()  
        .addAll(new FileChooser.ExtensionFilter(fileType, x)));  
  
    return chooser.showOpenDialog(menuBar.getScene().getWindow());  
}
```

- für jeweiligen Dateityp aufgerufen:

```
public void loadVideo() {  
    File chosenFile = openFile(title: "Select Video", fileType: "Video Files", videoExtensions);  
}
```

→ unterstützte Formate nachträglich änderbar

## 4.2 Neue Funktionen - Videoplayer

- umgesetzt mit VLCJ – Framework  
→ setzt VLC 2.2+ - Installation voraus

## 4.2 Videoplayer – Laden von Video

```
public void onClickLoad() {  
    VideoPlayer videoPlayer = new VideoPlayer(videoScrollPane, videoHolder);  
    videoPlayer.getVideoPlayer().startMedia(Config.PATH_VIDEO);  
    videoPlayer.getVideoPlayer().pause();  
    videos.add(videoPlayer);  
}
```

## 4.2 Videoplayer – Laden von Video

```
public void onClickLoad() {  
    VideoPlayer videoPlayer = new VideoPlayer(videoScrollPane, videoHolder);  
    videoPlayer.getVideoPlayer().startMedia(Config.PATH_VIDEO);  
    videoPlayer.getVideoPlayer().pause();  
    videos.add(videoPlayer);  
}
```

- VideoPlayer – Objekt wird erstellt:
  - erstellt internen (vlc-)MediaPlayer und setzt ImageView in den Videobereich
  - über WritableImage/PixelWriter wird MediaPlayer – Inhalt an ImageView gegeben
  - ImageView immer genau Hälfte der Breite des Videobereichs und skaliert mit Fenstergröße



## 4.2 Videoplayer – Laden von Video

```
public void onClickLoad() {  
    VideoPlayer videoPlayer = new VideoPlayer(videoScrollPane, videoHolder);  
    videoPlayer.getVideoPlayer().startMedia(Config.PATH_VIDEO);  
    videoPlayer.getVideoPlayer().pause();  
    videos.add(videoPlayer);  
}
```

- Video bei *PATH\_VIDEO* wird in MediaPlayer geladen und sofort pausiert

## 4.2 Videoplayer – Laden von Video

```
public void onClickLoad() {  
    VideoPlayer videoPlayer = new VideoPlayer(videoScrollPane, videoHolder);  
    videoPlayer.getMediaPlayer().startMedia(Config.PATH_VIDEO);  
    videoPlayer.getMediaPlayer().pause();  
    videos.add(videoPlayer);  
}
```

- Video bei *PATH\_VIDEO* wird in MediaPlayer geladen und sofort pausiert
- VideoPlayer wird in Liste der geladenen Videos hinzugefügt

## 4.2 Videoplayer – Play/Pause

```
public void onClickPlayPause() {  
    isPlaying = !isPlaying;  
    if (isPlaying) {  
        videos.stream().filter(x -> x.getVideoPlayer().isPlayable()).forEach(x -> x.getVideoPlayer().play());  
        buttonPlayPauseIcon.setImage(pauseIcon);  
    } else {  
        videos.stream().filter(x -> x.getVideoPlayer().isPlaying()).forEach(x -> x.getVideoPlayer().pause());  
        buttonPlayPauseIcon.setImage(playIcon);  
    }  
}
```

## 4.2 Videoplayer – Play/Pause

```
public void onClickPlayPause() {  
    isPlaying = !isPlaying;  
    if (isPlaying) {  
        videos.stream().filter(x -> x.getVideoPlayer().isPlayable()).forEach(x -> x.getVideoPlayer().play());  
        buttonPlayPauseIcon.setImage(pauseIcon);  
    } else {  
        videos.stream().filter(x -> x.getVideoPlayer().isPlaying()).forEach(x -> x.getVideoPlayer().pause());  
        buttonPlayPauseIcon.setImage(playIcon);  
    }  
}
```

- Zustands – Flag wird umgeschaltet (init. false)

## 4.2 Videoplayer – Play/Pause

```
public void onClickPlayPause() {  
    isPlaying = !isPlaying;  
    if (isPlaying) {  
        videos.stream().filter(x -> x.getVideoPlayer().isPlayable()).forEach(x -> x.getVideoPlayer().play());  
        buttonPlayPauseIcon.setImage(pauseIcon);  
    } else {  
        videos.stream().filter(x -> x.getVideoPlayer().isPlaying()).forEach(x -> x.getVideoPlayer().pause());  
        buttonPlayPauseIcon.setImage(playIcon);  
    }  
}
```

- Zustands – Flag wird umgeschaltet (init. false)
- Flag wird abgefragt
- falls Zustand auf true/spielend steht:
  - Liste der geladenen Videos wird gefiltert und pausiert
  - Icon des Play/Pause – Buttons wird gewechselt
- analog bei Zustand auf false

## 4.3 Neue Funktionen - Loganzeige

- Logeinträge werden in LogEntry – Objekten gespeichert:

```
public LogEntry(String dateTime, String entryText) {  
    this.dateTime = LocalDateTime.parse(dateTime, TIME_STAMP_FORMATTER);  
    this.message = entryText;  
}
```

- Zeitstempel(-String) wird zu LocalDateTime konvertiert und gespeichert  
→ leichte ermittlung zeitlicher Abstände
- Text der Logzeile separat in String *message*

## 4.3 Neue Funktionen - Loganzeige

- Parsen einer Logdatei:

```
public static List<LogEntry> parse(String path) throws IOException {  
    try (Stream<String> lines = Files.lines(Paths.get(path))) {  
        return lines.map(LOG_PATTERN::matcher).filter(Matcher::matches).map(LogParser::toLogEntry).collect(Collectors.toList());  
    }  
}
```

## 4.3 Neue Funktionen - Loganzeige

- Parsen einer Logdatei:

```
public static List<LogEntry> parse(String path) throws IOException {  
    try {  
        Stream<String> lines = Files.lines(Paths.get(path));  
        return lines.map(LOG_PATTERN::matcher).filter(Matcher::matches).map(LogParser::toLogEntry).collect(Collectors.toList());  
    }  
}
```

- Zeilen der Datei bei *path* werden in Stream geladen



## 4.3 Neue Funktionen - Loganzeige

- Parsen einer Logdatei:

```
public static List<LogEntry> parse(String path) throws IOException {  
    try (Stream<String> lines = Files.lines(Paths.get(path))) {  
        return lines.map(LOG_PATTERN::matcher).filter(Matcher::matches).map(LogParser::toLogEntry).collect(Collectors.toList());  
    }  
}
```

- Zeilen der Datei bei *path* werden in Stream geladen
- Aus Stream werden matchende Zeilen in LogEntry – Objekte konvertiert und als Liste gespeichert

## 4.3 Neue Funktionen - Loganzeige

- Parsen einer Logdatei:

```
public static List<LogEntry> parse(String path) throws IOException {  
    try (Stream<String> lines = Files.lines(Paths.get(path))) {  
        return lines.map(LOG_PATTERN::matcher).filter(Matcher::matches).map(LogParser::toLogEntry).collect(Collectors.toList());  
    }  
}
```

- Zeilen der Datei bei *path* werden in Stream geladen
- Aus Stream werden matchende Zeilen in LogEntry – Objekte konvertiert und als Liste gespeichert

→ *LOG\_PATTERN* ist definiert wie folgt:

## 4.3 Neue Funktionen - Loganzeige

- wird aus RegEx - String kompiliert:

```
LOG_PATTERN = Pattern.compile("(\\d{4}\\.\\d{1,2}\\.\\d{1,2}-\\d{2}\\.\\d{2}\\.\\d{2})\\:\\s(.*)");
```

- definiert zwei CapturingGroups:

## 4.3 Neue Funktionen - Loganzeige

- wird aus RegEx - String kompiliert:

```
LOG_PATTERN = Pattern.compile("(\\d{4}\\.\\d{1,2}\\.\\d{1,2}-\\d{2}\\.\\d{2}\\.\\d{2})\\:\\s(.*)");
```

- definiert zwei CapturingGroups

→ matched: **2018.6.21-13.14.24**: Playing

## 4.3 Neue Funktionen - Loganzeige

- wird aus RegEx - String kompiliert:

```
LOG_PATTERN = Pattern.compile("(\\d{4}\\.\\d{1,2}\\d{1,2}-\\d{2}\\d{2}\\d{2})\\:\\s(.*)");
```

- definiert zwei CapturingGroups

→ matched: 2018.6.21-13.14.24: **Playing**

(jeglichen Character 0+ mal, bis auf \n)

## 4.3 Neue Funktionen - Loganzeige

- Hier eingesetzt Funktion *toLogEntry* nutzt dies:


```
public static List<LogEntry> parse(String path) throws IOException {  
    try (Stream<String> lines = Files.lines(Paths.get(path))) {  
        return lines.map(LOG_PATTERN::matcher).filter(Matcher::matches).map(LogParser::toLogEntry).collect(Collectors.toList());  
    }  
}
```

## 4.3 Neue Funktionen - Loganzeige

- Hier eingesetzt Funktion *toLogEntry* nutzt dies:

```
public static List<LogEntry> parse(String path) throws IOException {  
    try (Stream<String> lines = Files.lines(Paths.get(path))) {  
        return lines.map(LOG_PATTERN::matcher).filter(Matcher::matches).map(LogParser::toLogEntry).collect(Collectors.toList());  
    }  
}
```

```
private static LogEntry toLogEntry(Matcher m) {  
    return new LogEntry(m.group(1), m.group(2));  
}
```



2018.6.21-13.14.24: Playing

## 4.3 Neue Funktionen - Loganzeige

- Hier eingesetzt Funktion *toLogEntry* nutzt dies:

```
public static List<LogEntry> parse(String path) throws IOException {  
    try (Stream<String> lines = Files.lines(Paths.get(path))) {  
        return lines.map(LOG_PATTERN::matcher).filter(Matcher::matches).map(LogParser::toLogEntry).collect(Collectors.toList());  
    }  
}
```

```
private static LogEntry toLogEntry(Matcher m) {  
    return new LogEntry(m.group(1), m.group(2));  
}
```

↔ 2018.6.21-13.14.24: Playing



```
public LogEntry(String dateTime, String entryText) {  
    this.dateTime = LocalDateTime.parse(dateTime, TIME_STAMP_FORMATTER);  
    this.message = entryText;  
}
```



## 4.3 Neue Funktionen - Loganzeige

- erhaltene LogEntry – Liste wird in ObservableList überführt
  - erleichtert Reaktion auf Änderungen (Observer Pattern)
  - direkte Einspeisung in JFX-ListView zum anzeigen der Einträge in der Loganzeige

**Vielen Dank für die  
Aufmerksamkeit!**

**Fragen?**