Aufbau interaktiver 3D-Engines

Universität Osnabrück
Fachbereich Mathematik / Informatik

4. Übung

30.04.2013

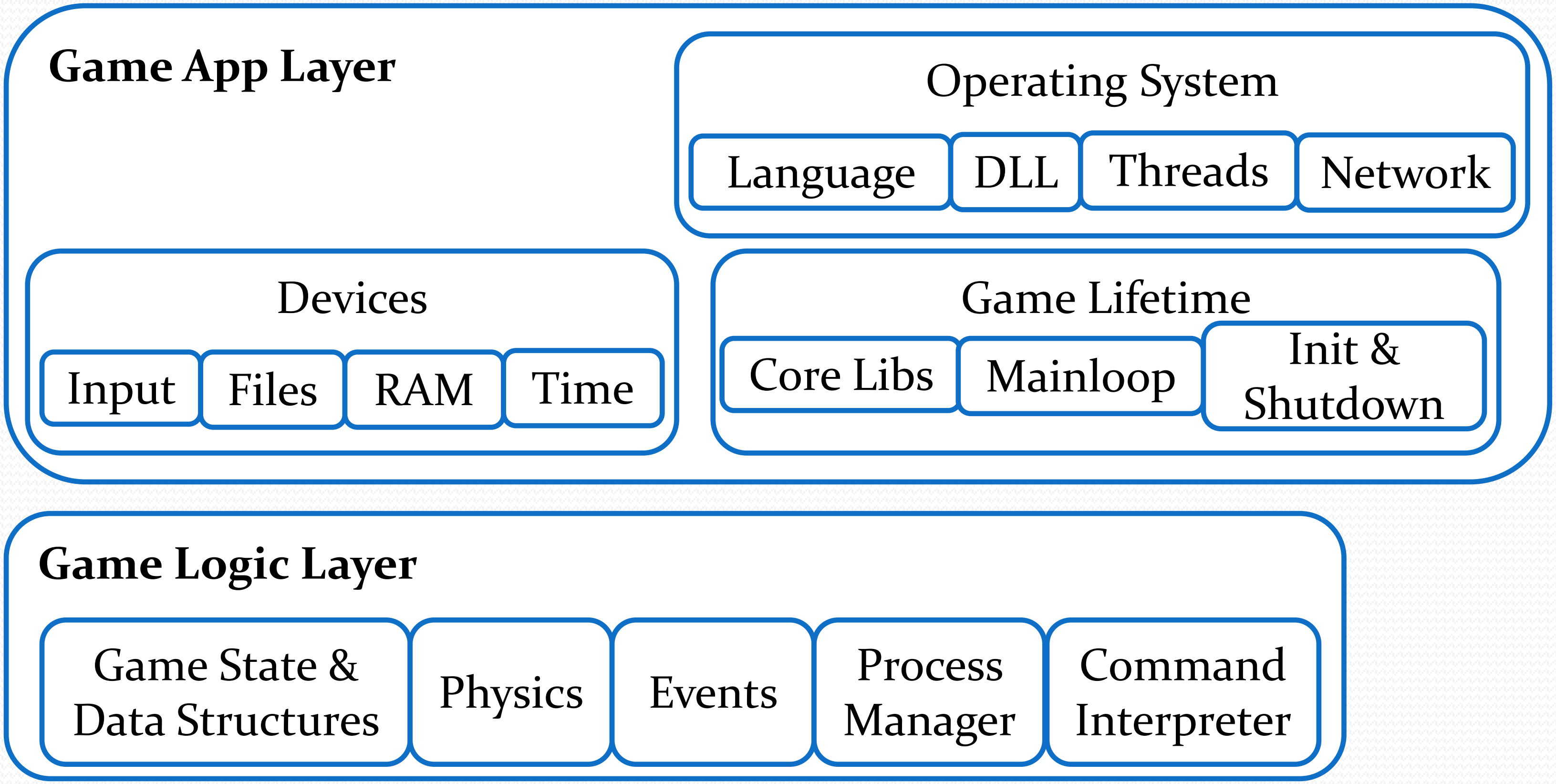
Prof. Dr. rer. nat. Oliver Vornberger
Nico Marniok, B. Sc.
Erik Wittkorn, B. Sc.

Übersicht

1. Fragen zum aktuellen Übungsblatt
2. Rückblick: Übersicht
3. Implementationsdetails
   1. Eventsystem
   2. Prozesssystem
   3. Input
4. Mathematische Grundlagen

Fragen zum aktuellen Übungsblatt

Rückblick: Übersicht



Game App Layer

Operating System

Language | DLL | Threads | Network

Devices

Input | Files | RAM | Time

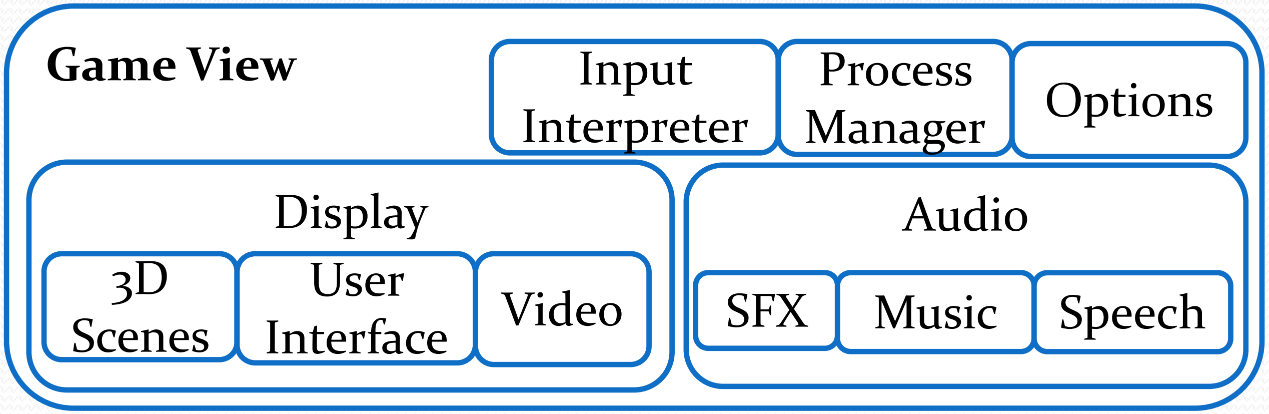
Game Lifetime

Core Libs | Mainloop | Init & Shutdown

Game Logic Layer

Game State & Data Structures | Physics | Events | Process Manager | Command Interpreter

Rückblick: Übersicht



Quelle: Game Coding Complete

Implementation: EventSystem

- Eventmanager
  - Verwaltung der Event Queues und der Listener
  - Events werden gesammelt und verarbeitet
- Eventlistener
  - Implementieren die trigger(Eventdata data) Funktion
- Events
  - Haben eine eindeutige ID
  - Transportieren spezielle Informationen

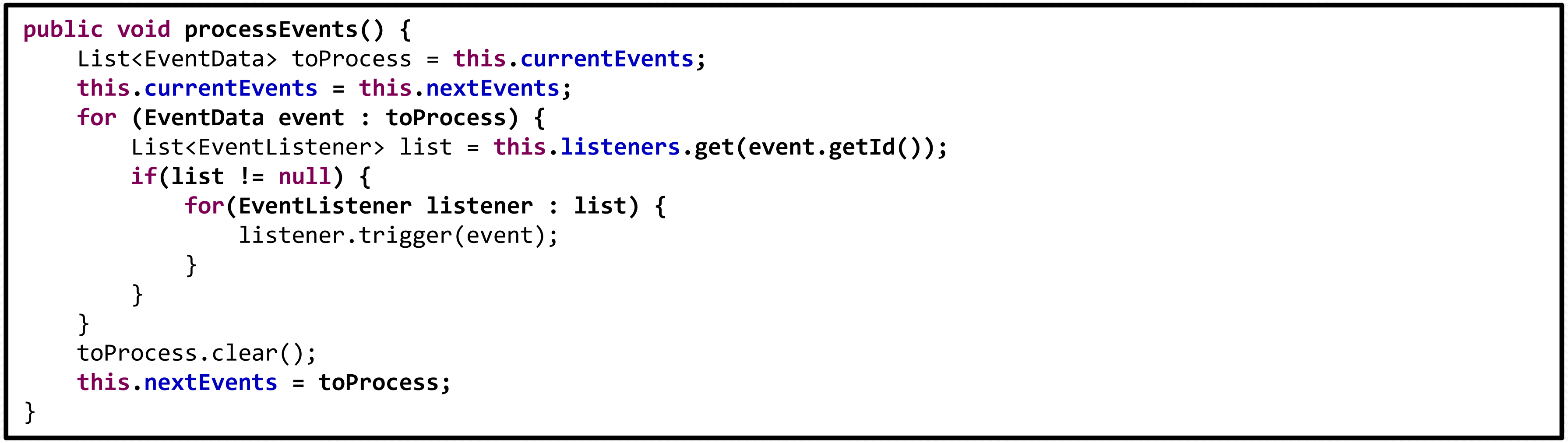
Implementation: EventManager

```
private List<EventData> currentEvents = new LinkedList<>();
private List<EventData> nextEvents = new LinkedList<>();
private final Map<Integer, List<EventListener>> listeners = new HashMap<>();
```

```
public boolean register(EventListener listener, int... types) {
    boolean success = true;
    for(int type : types) {
        List<EventListener> list = this.listeners.get(type);
        if(list == null) {
            this.listeners.put(type, list = new LinkedList<>());
        }
        if(list.contains(listener)) {
            Logger.INSTANCE.warning("Don't register listener twice.");
            success = false;
        } else {
            list.add(listener);
        }
    }
    return success;
}
```

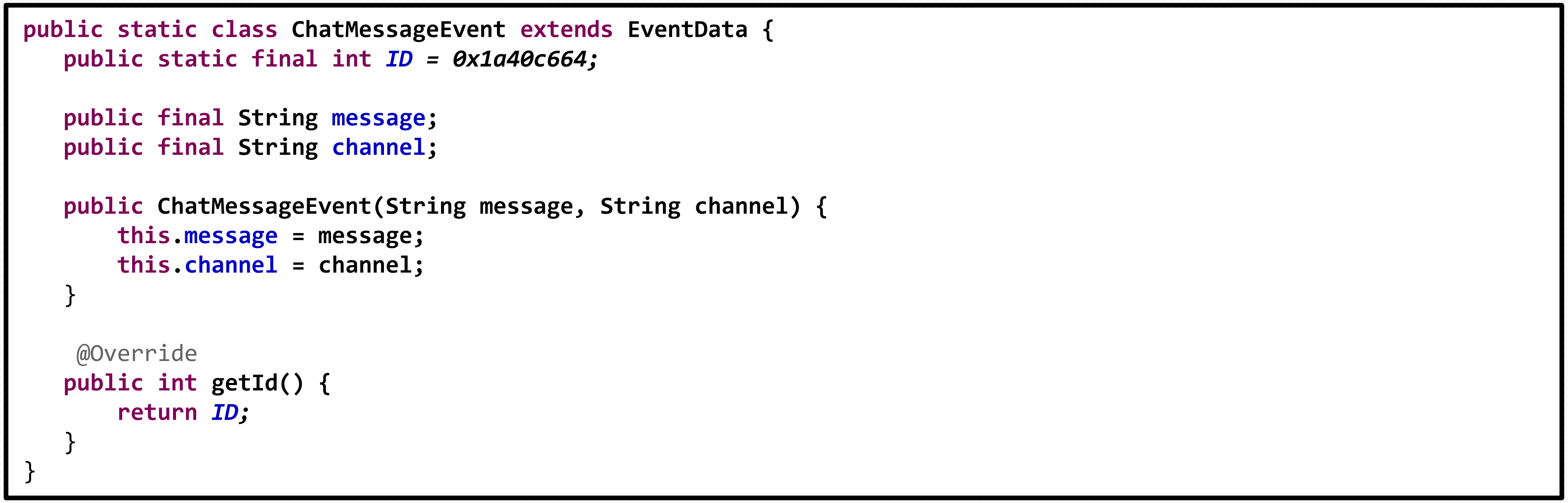
```
public void queueEvent(EventData event) {
    event.timeStamp = GameApp.getRealTimer().getTimeStamp();
    this.currentEvents.add(event);
}
```

Implementation: EventManager



```
public void processEvents() {
    List<EventData> toProcess = this.currentEvents;
    this.currentEvents = this.nextEvents;
    for (EventData event : toProcess) {
        List<EventListener> list = this.listeners.get(event.getId());
        if(list != null) {
            for(EventListener listener : list) {
                listener.trigger(event);
            }
        }
    }
    toProcess.clear();
    this.nextEvents = toProcess;
}
```

Implementation: Events und Listener

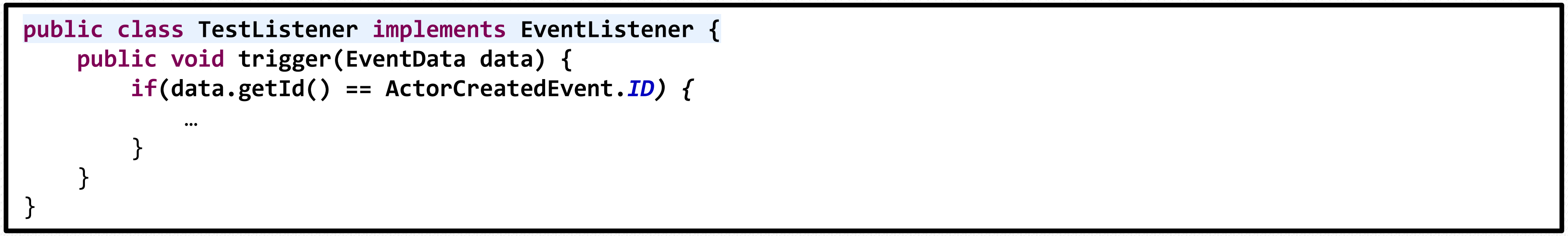


```java
public static class ChatMessageEvent extends EventData {
    public static final int ID = 0x1a40c664;

    public final String message;
    public final String channel;

    public ChatMessageEvent(String message, String channel) {
        this.message = message;
        this.channel = channel;
    }

     @Override
    public int getId() {
        return ID;
    }
}
```



```java
public class TestListener implements EventListener {
    public void trigger(EventData data) {
        if(data.getId() == ActorCreatedEvent.ID) {
            …
        }
    }
}
```

Implementation: Prozesssystem

- Prozess
  - Besitzt zeitliche Ausdehnung
  - Kann erfolgreich sein, oder fehlschlagen
  - Kann pausiert oder abgebrochen werden
- Prozessmanager
  - Verwaltet Prozesse
  - Prozesse führen über einen, oder nach einem Zeitraum Operationen durch

Implementation: Prozess

```
static enum State {
    UNINITIALIZED,
    REMOVED,
    RUNNING,
    PAUSED,
    SUCCEEDED,
    FAILED,
    ABORTED,
}

private State state = State.UNINITIALIZED;

private EngineProcess child = null;
```

```
protected abstract void onUpdate(long dMillis);

public final void succeed() {
    if(this.isAlive()) {
        this.state = State.SUCCEEDED;
    }
}

protected void onSuccess() {}
```

Implementation: ProzessManager

```java
private final LinkedList<EngineProcess> processes = new LinkedList<>();
private final List<EngineProcess> tempProcesses = new LinkedList<>();
```

```java
public int update(long deltaMillis) {
    this.processes.addAll(this.tempProcesses); this.tempProcesses.clear();
    ListIterator<EngineProcess> iter = this.processes.listIterator();
    for(EngineProcess process; iter.hasNext();) {
        process = iter.next();
        if(process.getState() == EngineProcess.State.UNINITIALIZED) {
            process.onInit();
        } else if(process.getState() == EngineProcess.State.RUNNING) {
            process.onUpdate(deltaMillis);
        } else if(process.isDead()) {
            switch(process.getState()) {
                case ABORTED:
                    process.onAbort();
                    iter.remove(); ++failed; break;
                case FAILED:
                    process.onFail();
                    iter.remove(); ++failed; break;
                case SUCCEEDED:
                    process.onSuccess();
                    if(process.getChild() != null) {
                       iter.set(process.getChild());
                    } else {
                        iter.remove();
                        ++succeeded; } break;
        }}}
        return (failed << 16) | succeeded;
    }
```

Implementation: Input

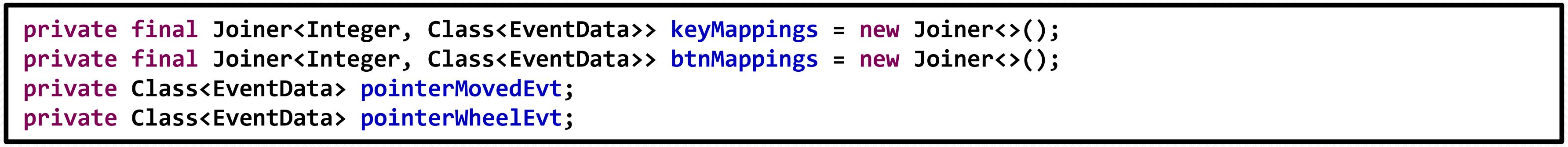
- InputHandler (GameApp)
  - Leitet Hardware-Input an registrierte Handler weiter
- KeyboardHandler
  - **`onKeyDown`**, **`onKeyUp`** und **`isProcessingRepeatEvents`** verarbeiten Tastatur-Input
- PointerHandler
  - **`pointerMoved`**, **`onButtonDown`** , **`onButtonUp`** und **`onWheelMoved`** verarbeiten Maus-Input
- PlayerControls (implements KeyboardHandler, PointerHandler)
  - Standard-InputHandler, der Eingaben in Events umwandelt

Implementation: InputHandler

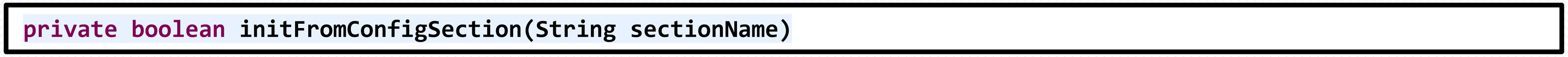
```java
private final LinkedList<KeyboardHandler> keyboardHandlers = new LinkedList<>();
private final LinkedList<PointerHandler> pointerHandlers = new LinkedList<>();
```

```java
public void processInput() {
    while(Keyboard.next()) {
        boolean processed = false; boolean down = Keyboard.getEventKeyState();
        Iterator<KeyboardHandler> iter = this.keyboardHandlers.iterator();
        while(!processed && iter.hasNext()) {
            KeyboardHandler handler = iter.next();
            if(down) {
                if(!Keyboard.isRepeatEvent() || handler.isProcessingRepeatEvents()) {
                    processed = handler.onKeyDown(Keyboard.getEventKey(), Keyboard.getEventCharacter());
                }
            } else {
                processed = handler.onKeyUp(Keyboard.getEventKey());}}}
    while(Mouse.next()) {
        boolean processed = false; int button = Mouse.getEventButton();
        boolean down = Mouse.getEventButtonState(); int wheel = Mouse.getEventDWheel();
        int posX = Mouse.getEventX(), posY = Mouse.getEventY(), dX = Mouse.getEventDX(), dY = Mouse.getEventDY();
        Iterator<PointerHandler> iter = this.pointerHandlers.iterator();
        while(!processed && iter.hasNext()) {
            if(button == -1 && (dX != 0 || dY != 0)) {
                processed = iter.next().pointerMoved(posX, posY, dX, dY);
            } else if(wheel != 0) {
                processed = iter.next().onWheelMoved(wheel);
            } else if(down) {
                processed = iter.next().onButtonDown(button);
            } else {
                processed = iter.next().onButtonUp(button);}}}}
```

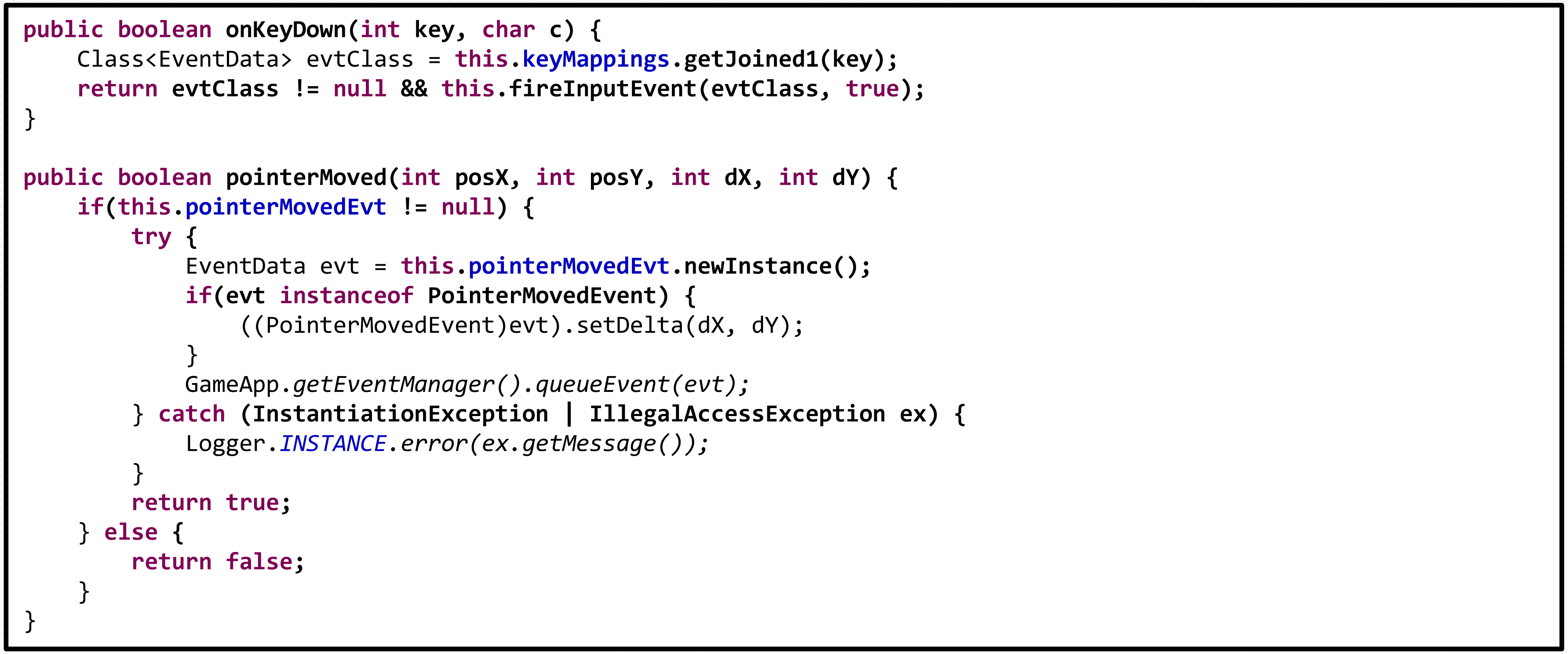
Implementation: PlayerControls



```
private final Joiner<Integer, Class<EventData>> keyMappings = new Joiner<>();
private final Joiner<Integer, Class<EventData>> btnMappings = new Joiner<>();
private Class<EventData> pointerMovedEvt;
private Class<EventData> pointerWheelEvt;
```



```
private boolean initFromConfigSection(String sectionName)
```



```
public boolean onKeyDown(int key, char c) {
    Class<EventData> evtClass = this.keyMappings.getJoined1(key);
    return evtClass != null && this.fireInputEvent(evtClass, true);
}

public boolean pointerMoved(int posX, int posY, int dX, int dY) {
    if(this.pointerMovedEvt != null) {
        try {
            EventData evt = this.pointerMovedEvt.newInstance();
            if(evt instanceof PointerMovedEvent) {
                ((PointerMovedEvent)evt).setDelta(dX, dY);
            }
            GameApp.getEventManager().queueEvent(evt);
        } catch (InstantiationException | IllegalAccessException ex) {
            Logger.INSTANCE.error(ex.getMessage());
        }
        return true;
    } else {
        return false;
    }
}
```

Mathematische Grundlagen

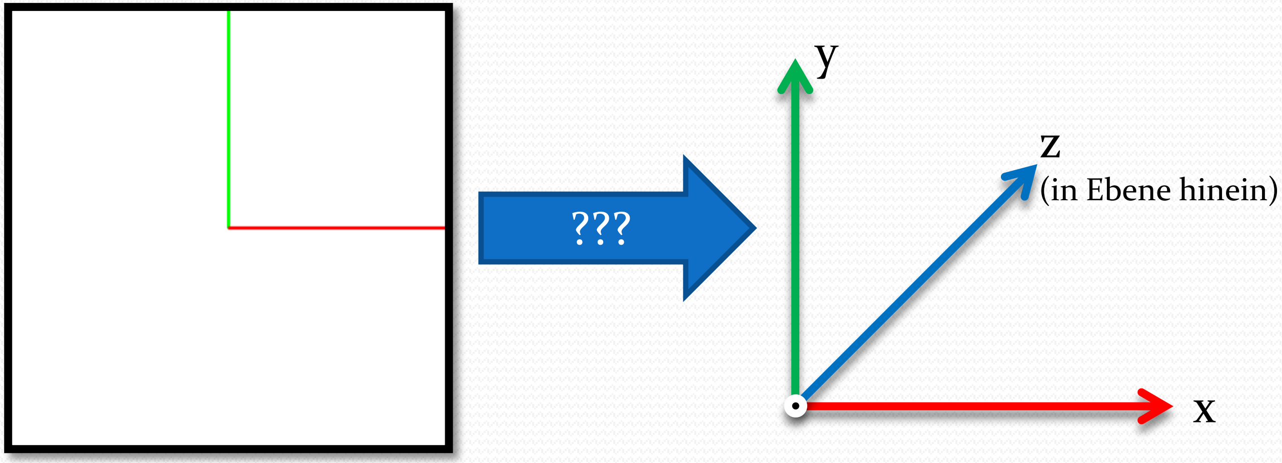
- $S\left(\frac{1}{2},\frac{1}{2},\frac{1}{2}\right) = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- $T_x(0{,}5) = \begin{pmatrix} 1 & 0 & 0 & 0{,}5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Mathematische Grundlagen

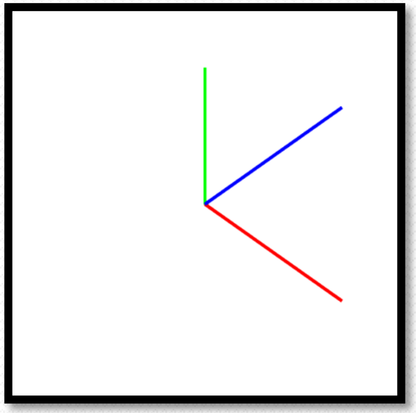
- $R_x\left(\pm\frac{\pi}{4}\right) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\pm\frac{\pi}{4} & -\sin\pm\frac{\pi}{4} & 0 \\ 0 & \sin\pm\frac{\pi}{4} & \cos\pm\frac{\pi}{4} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- $R_y\left(\pm\frac{\pi}{4}\right) = \begin{pmatrix} \cos\pm\frac{\pi}{4} & 0 & \sin\pm\frac{\pi}{4} & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\pm\frac{\pi}{4} & 0 & cos\pm\frac{\pi}{4} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- $R_z\left(\pm\frac{\pi}{4}\right) = \begin{pmatrix} \cos\pm\frac{\pi}{4} & -\sin\pm\frac{\pi}{4} & 0 & 0 \\ \sin\pm\frac{\pi}{4} & \cos\pm\frac{\pi}{4} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
- $\sin\left(\pm\frac{\pi}{4}\right) = \pm\frac{1}{\sqrt{2}} = cos\left(\pm\frac{\pi}{4}\right)$

Mathematische Grundlagen

- Ausgangslage



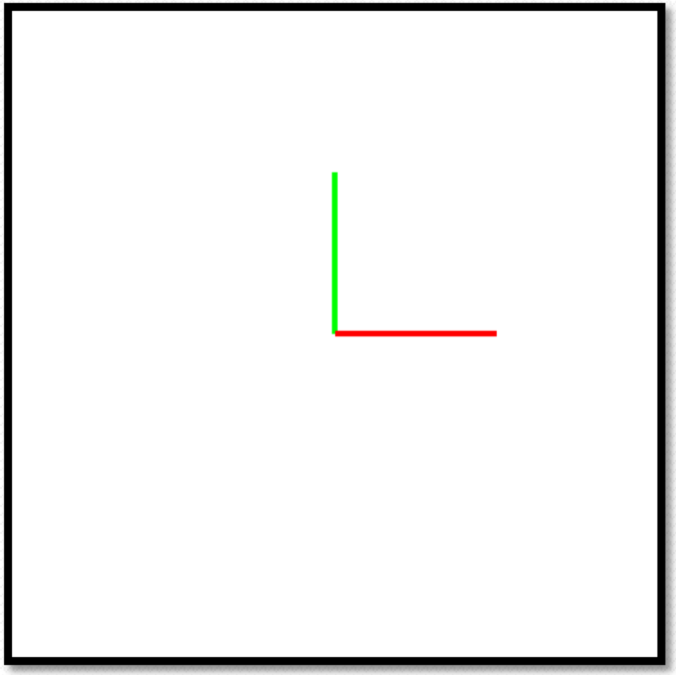
$$M = R_x\left(-\frac{\pi}{4}\right) * R_y\left(+\frac{\pi}{4}\right)$$



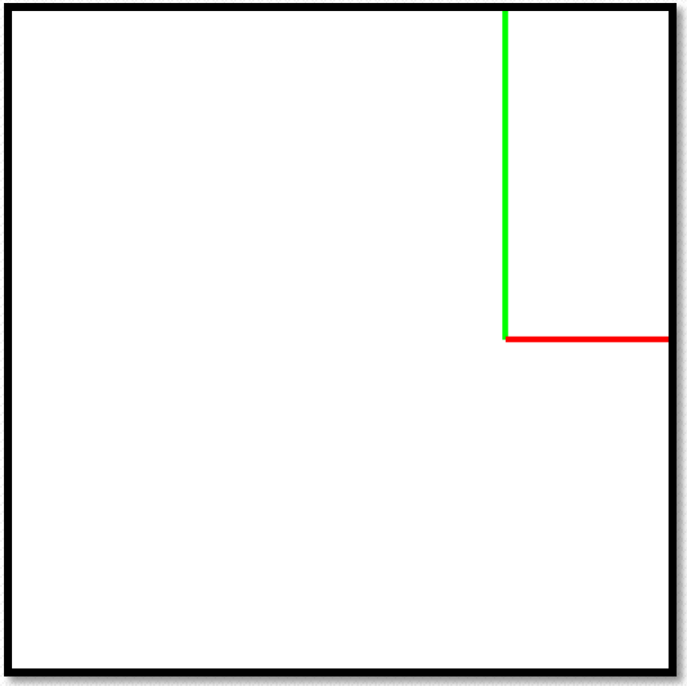
Merke:
Positive Rotationen gehen gegen den Uhrzeigersinn, wenn man entlang der Achse sieht

Mathematische Grundlagen

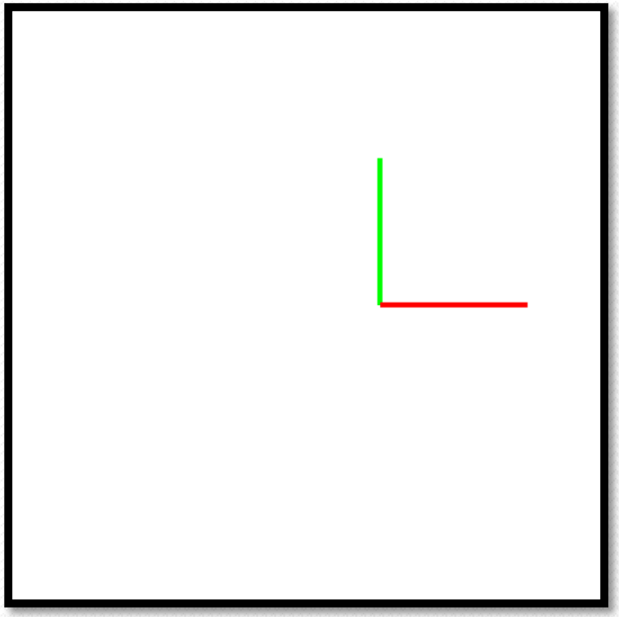
$$M = S\left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$$

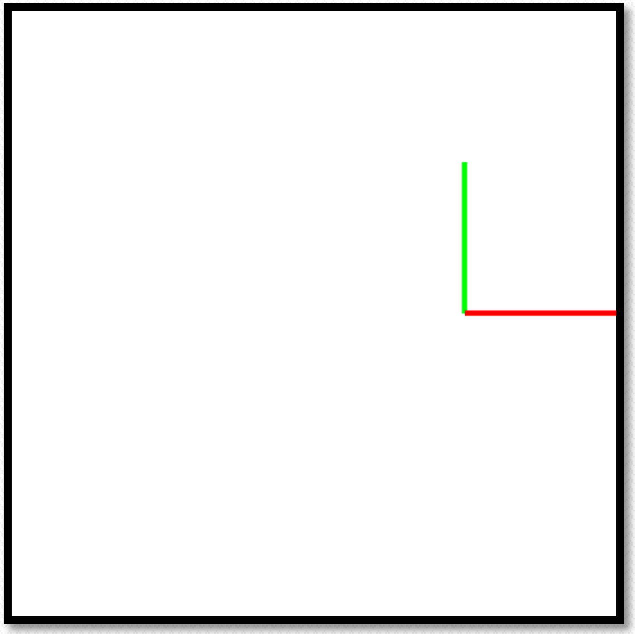


$$M = T_x\left(\frac{1}{2}\right)$$



Mathematische Grundlagen



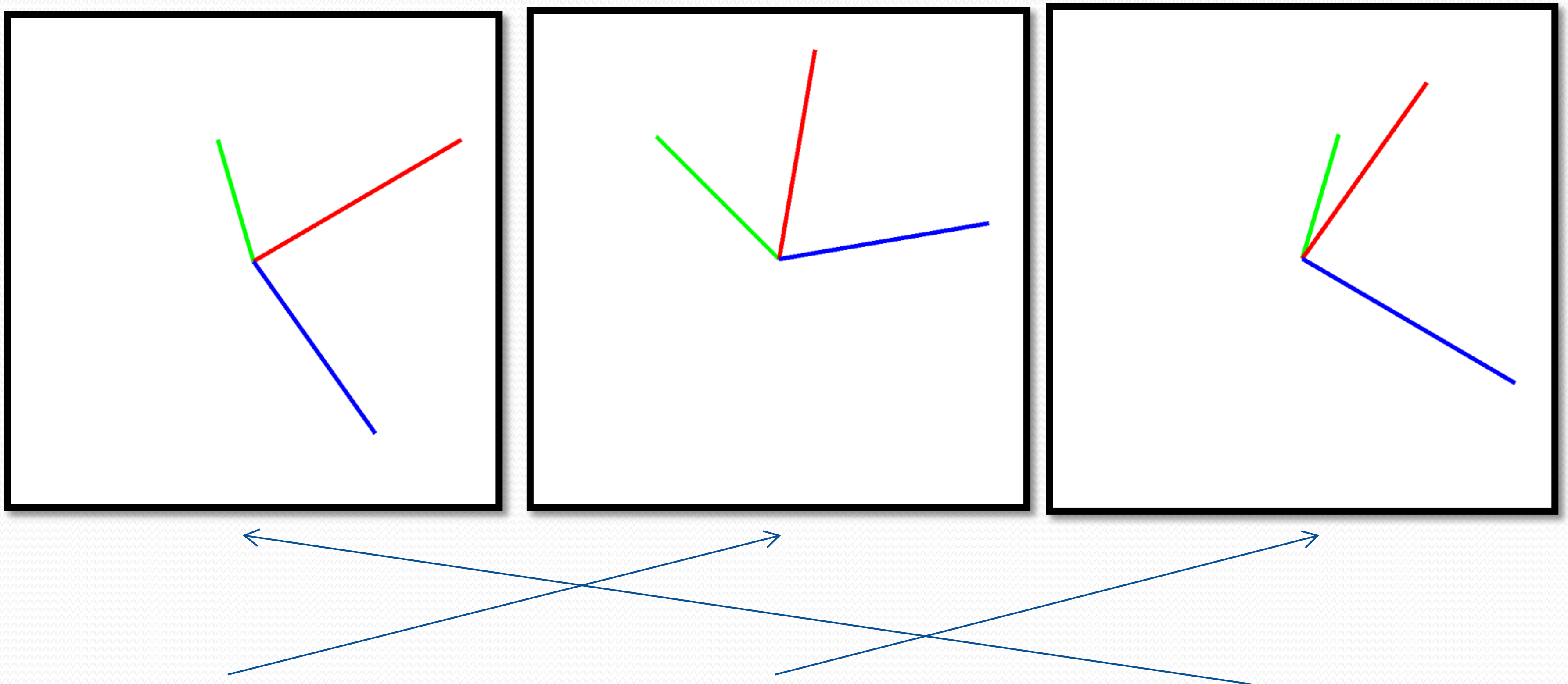


$$M = S\left(\frac{1}{2},\frac{1}{2},\frac{1}{2}\right) * T_x\left(\frac{1}{2}\right)$$

$$M = T_x\left(\frac{1}{2}\right) * S\left(\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)$$

Merke:
Skalierungen wirken global

Mathematische Grundlagen



$M = R_x\left(\frac{\pi}{4}\right) * R_z\left(\frac{\pi}{4}\right) * R_y\left(\frac{\pi}{4}\right)$

$M = R_y\left(\frac{\pi}{4}\right) *R_z\left(\frac{\pi}{4}\right) * R_x\left(\frac{\pi}{4}\right)$

$M = R_y\left(\frac{\pi}{4}\right) * R_x\left(\frac{\pi}{4}\right) *R_z\left(\frac{\pi}{4}\right)$

Merke:
*Yaw (Y) * Pitch (X) * Roll (Z)*

Mathematische Grundlagen

Merke:
*M = Translation * Rotation * Skalierung*

Nächste Woche:

- Abgabe des aktuellen Übungsblattes
- Ausgabe des nächsten Übungsblattes

Vielen Dank für die Aufmerksamkeit ☺