Das modifizierte Slick-Framework eea (entity event action)

1 Slick2D und eea

Slick2D[1] ist ein einfaches, aber mächtiges 2D-Rendering-Framework. Es eignet sich hervorragend für das Erstellen von einfachen 2D-Spielen. Es stellt die Darstellung von grafischen Objekten mit OpenGL und das Abspielen von Tönen bereit. Das speziell für das GdI1-Projekt entwickelte Framework *eea* basiert auf Slick2D und erweitert das Framework um Entitäten (`Entity`), diverse Komponenten (`Event`, `RenderComponent`) und Aktionen (`Action`).

2 Überblick über das Framework eea

Um das *eea*-Framework richtig ausnutzen zu können, ist es unbedingt nötig, die wichtigsten Module zu kennen. Einen Überblick über *eea* gibt Abbildung 1.

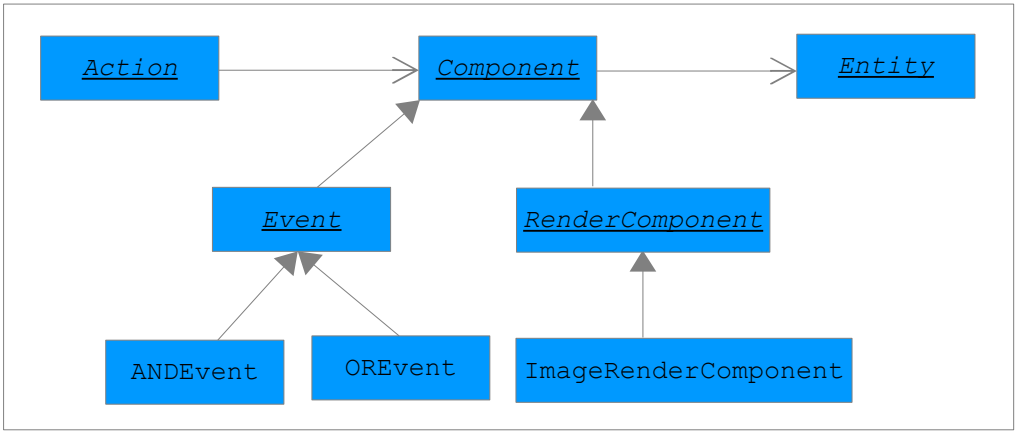


Abbildung 1: Überblick über *eea*

Eine `Entity` ist wie der Name schon nahelegt eine beliebige Entität. Im Framework wird diese durch eine abstrakte Klasse mit gleichem Namen modelliert. Eine Entität kann im Fenster dargestellt werden, muss es aber nicht. Das Objekt lässt sich bereits skalieren, rotieren und positionieren. Es kann außerdem festgelegt werden, ob sich die `Entity` mit anderen `Entity`-Objekten überlagern lässt.

Eine `Entity` ist üblicherweise interessiert an eintreffenden Ereignissen. Diese werden einer `Entity` in Form von `Event`s hinzugefügt. `Event`s erben von der Klasse `Component`. So möchte ein Objekt beispielsweise wissen, ob eine Tastatureingabe stattfand, damit es sich dann z.B. fortbewegen kann.

Die Fortbewegung selbst wird durch eine `Action` definiert. Im Falle einer Vorwärtsbewegung könnte hier dem `KeyPressedEvent` für eine ausgewählte Taste eine `MoveForwardAction` hinzugefügt werden.

Mehrere Ereignisse lassen sich zu einem `MultiEvent` zusammenfassen. Im Konstruktor lassen sich hier beliebig viele `Event`-Objekte als Parameter übergeben. Treten alle Ereignisse gleichzeitig ein (Konjunktion [logisches Und] über `Event`-Objekte, repräsentiert durch einen `ANDEvent`) oder soll nur mindestens ein Ergebnis eintreten (entsprechend durch einen `OREvent` codiert), so werden alle auf dem jeweiligen `ANDEvent` bzw. `OREvent` registrierten `Action`s ausgeführt.

1 Mehr Informationen zu Slick 2D unter http://slick.cokeandcode.com/static.php?page=about. Unter http://slick.cokeandcode.com/wiki/doku.php?id=tutorials liegen sehr gute Tutorials für die Verwendung von Slick 2D. Dies geht jedoch weit über unsere Aufgabe hinaus.

Weiterhin erbt auch `RenderComponent` von `Component`. Einer `Entity` kann ein `RenderComponent`-Objekt hinzugefügt werden, damit es im Bildschirm gezeichnet werden kann. Wird einem `Entity`-Objekt ein Bild hinzugefügt, so wird das Zeichnen des Bilds über ein `ImageRenderComponent`-Objekt angestoßen. Das Zeichnen erfolgt immer vor einem Frame (Einzelbild des Programmfensters).

3 Erstellen eines ersten Programms

Nach dieser knappen Einführung folgt nun ein kleines Tutorial. Das Ziel ist das Erstellen eines sehr einfachen Spiels, welches erklärt, wie Entitäten sich bewegen können und das Zusammenspiel mit den anderen zuvor erwähnten Objekten funktioniert.

Es soll zwei Fensteransichten geben. In einem Menü kann man die Einträge „Spiel starten“ und „Beenden“ anklicken, in einem zweiten Fenster findet das tatsächliche Spiel statt.

Das Spiel enthält einen Wassertropfen, der durch Drücken der Maustaste an genau dieser Stelle instanziert wird. Anschließend fällt er nach unten bis zum Ende des Fensterrands und wird bei Berühren des Bodens zerstört. Danach wird wieder zurück in das Hauptmenü gewechselt. Wird während des Spiels die Escape-Taste gedrückt, so soll ebenfalls ins Hauptmenü zurück gewechselt werden.

3.1 Anlegen der Basisklassen

Zunächt brauchen wir eine Launcher-Klasse zum Starten des Programms. Wir möchten mehrere Fensteransichten (States) haben, also erbt `Launch` von der vordefinierten Klasse `StateBasedGame`. Wir möchten zwei States haben. Dazu werden zwei Integer-Konstanten `MAINMENU_STATE` und `GAMEPLAY_STATE` zur Identifikation der jeweiligen States angelegt. Über die überschriebene Methode `initStatesList` (wird automatisch bei Initialisierung aufgerufen) fügen wir die beiden States hinzu und übergeben sie auch dem `StateBasedEntityManager`.

Ein `StateBasedGame` wird wie auch andere Slick-Spiele in einem `AppGameContainer` gestartet. Dies geschieht selbstverständlich in der `main`-Methode. Abhängig vom Betriebssystem benötigt `eea` einen anderen Bibliothekenpfad. Dieser wird zu Beginn einmal gesetzt; falls sich die Bibliotheken bei Ihnen in einem anderen Verzeichnis befinden, ist diese Angabe entsprechend anzupassen!

```
public class Launch extends StateBasedGame {
    // Jeder State wird durch einen Integer-Wert gekennzeichnet
    public static final int MAINMENU_STATE = 0;
    public static final int GAMEPLAY_STATE = 1;
    public Launch() {
        super("Drop of Water"); // Name des Spiels
    }
    public static void main(String[] args) throws SlickException {
       // Setze den Bibliothekenpfad abhaengig vom Betriebssystem
       if (System.getProperty("os.name").toLowerCase().contains("windows")) {
           System.setProperty("org.lwjgl.librarypath",
           System.getProperty("user.dir") + "/native/windows");
       } else if {
           System.getProperty("os.name").toLowerCase().startsWith("mac")) {
           System.setProperty("org.lwjgl.librarypath",
           System.getProperty("user.dir") + "/native/macosx");
       } else {
```

```
            System.setProperty("org.lwjgl.librarypath",
            System.getProperty("user.dir") + "/native/"
    +System.getProperty("os.name").toLowerCase());
        }

        // Setze dieses StateBasedGame in einen App Container (oder Fenster)
        AppGameContainer app = new AppGameContainer(new Launch());

        // Lege die Einstellungen des Fensters fest und starte das Fenster
        // (aber nicht im Vollbildmodus)
        app.setDisplayMode(800, 600, false);
        app.start();
    }
    @Override
    public void initStatesList(GameContainer arg0) throws SlickException {

        // Fuege dem StateBasedGame die States hinzu
        // (der zuerst hinzugefuegte State wird als erster State gestartet)
        addState(new MainMenuState(MAINMENU_STATE));
        addState(new GameplayState(GAMEPLAY_STATE));

        // Fuege dem StateBasedEntityManager die States hinzu
        StateBasedEntityManager.getInstance().addState(MAINMENU_STATE);
        StateBasedEntityManager.getInstance().addState(GAMEPLAY_STATE);
    }
}
```

Damit etwas angezeigt werden kann, benötigen wir noch die beiden State-Klassen. Beide erben von `BasicGameState`, kennen ihren `StateBasedEntityManager`, der sie und ihre Entitäten verwaltet, sowie ihren Identifier `stateID`.

```
public class GameplayState extends BasicGameState {
    private int stateID; // Identifier dieses BasicGameState
    private StateBasedEntityManager entityManager; // zugehoeriger entityManager
    GameplayState(int sid) {
        stateID = sid;
        entityManager = StateBasedEntityManager.getInstance();
    }
    /**
     * Wird vor dem (erstmaligen) Starten dieses States ausgefuehrt
     */
    @Override
    public void init(GameContainer container, StateBasedGame game)
            throws SlickException {
        // hier ist (noch) nichts zu tun
    }
    /**
     * Wird vor dem Frame ausgefuehrt
     */
    @Override
    public void update(GameContainer container, StateBasedGame game, int delta)
            throws SlickException {
        // StatedBasedEntityManager soll alle Entities aktualisieren
        entityManager.updateEntities(container, game, delta);
    }
    /**
     * Wird mit dem Frame ausgefuehrt
     */
```

```
    @Override
    public void render(GameContainer container, StateBasedGame game, Graphics g)
            throws SlickException {
        // StatedBasedEntityManager soll alle Entities rendern
        entityManager.renderEntities(container, game, g);
    }
    @Override
    public int getID() {
        return stateID;
    }
}
```

Aus der Vererbung von `BasicGameState` werden die Methoden `init`, `render` und `update` überschrieben. Die Methode `init` wird einmalig bei der Initialisierung der Ansicht aufgerufen. `update` ermöglicht den Objekten, ihre Änderungen vorzunehmen. Die Methode wird vor dem Zeichnen pro Frame aufgerufen. `render` wird zusammen mit jedem Frame aufgerufen.

`MainMenuState` unterscheidet sich bisher nur durch den Namen von `GameplayState`. Es sollte sich nun ein schwarzes Fenster öffnen. Aus diesem Fenster soll später ein Menü werden.

3.2 Entitäten hinzufügen

Nun sollen Entitäten hinzugefügt und angezeigt werden. Die Initialisierung erfolgt natürlich in der `init`-Methode. Im folgenden ist zu sehen, wie ein Hintergrundbild geladen werden kann. Dabei ist zu beachten, dass im Ordner `assets` tatsächlich auch ein Bild `menu.png` enthalten ist.

```
    @Override
    public void init(GameContainer container, StateBasedGame game)
            throws SlickException {
        // Entität für Hintergrund
        Entity background = new Entity("menu");

        // Startposition des Hintergrunds
        background.setPosition(new Vector2f(400,300));

        // Bildkomponente zur Entität hinzufügen
        background.addComponent(new ImageRenderComponent(
                new Image("/assets/menu.png")));

        // Hintergrund-Entität an StateBasedEntityManager übergeben
        entityManager.addEntity(stateID, background);
    }
```

Auf die gleiche Weise lassen sich auch die anderen Entitäten hinzufügen. Wir erstellen nun noch zwei „Buttons“ zum Starten des Spiels und zum Beenden. Hier folgt der Code zum Wechseln in `GAMEPLAY_STATE`.

```
        /* Neues Spiel starten-Entitaet */
        String newGame = "Neues Spiel starten";
        Entity newGameEntity = new Entity(newGame);

        // Setze Position und Bildkomponente
        newGameEntity.setPosition(new Vector2f(218, 190));
        newGameEntity.setScale(0.28f);
        newGameEntity.addComponent(new ImageRenderComponent(
                new Image("assets/entry.png")));

        // Erstelle das Ausloese-Event und die zugehoerige Action
```

```
        MultiEvent mainEvents = new MultiEvent(new MouseEnteredEvent(),
                new MouseClickedEvent());
        Action newGameAction = new ChangeStateInitAction(Launch.GAMEPLAYSTATE);
        mainEvents.addAction(newGameAction);
        newGameEntity.addComponent(mainEvents);

        // Fuege die Entity zum StateBasedEntityManager hinzu
        entityManager.addEntity(this.stateID, newGameEntity);
```

Der Code zum Beenden des Spiels ist fast der gleiche. Hier wird lediglich eine andere Startposition und ein eine andere Action eingetragen. (Bei mehreren Buttons würde es sich lohnen, einen Blick auf das Factory Design Pattern[2] zu werfen.)

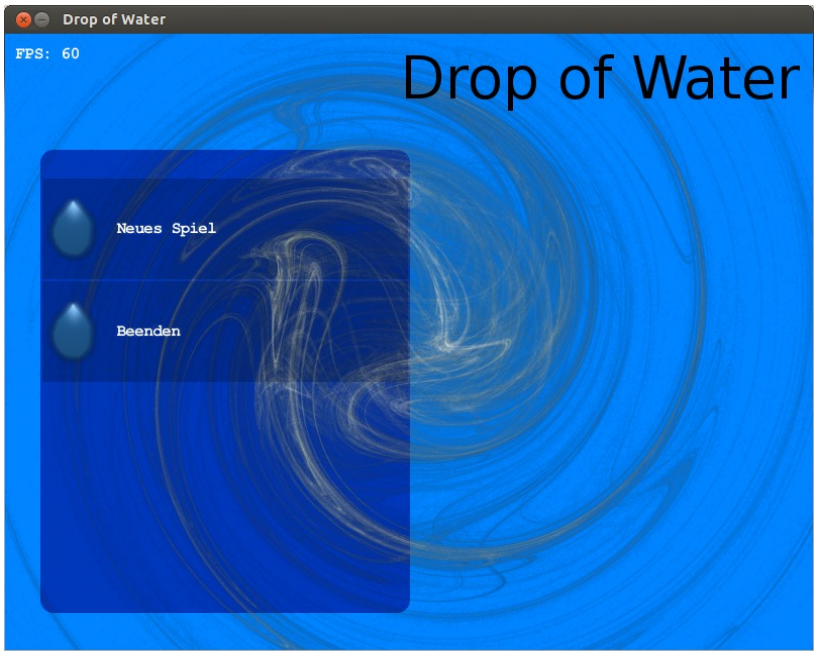
```
        Action quitAction = new QuitAction();
```

Nun haben wir allerdings noch Buttons ohne Beschriftung. Die Eingabe von Texten erfolgt in der `render`-Methode des jeweiligen State's. Wichtig ist, dass die Texte über die Bilder gelegt werden. Der Aufruf von `renderEntities` muss also vorher kommen.

```
    @Override
    public void render(GameContainer container, StateBasedGame game,
                                        Graphics g) throws SlickException {
        entityManager.renderEntities(container, game, g);

        int counter = 0;
        g.drawString("Neues Spiel", 110, startPosition + counter * distance);
        counter++;
        g.drawString("Beenden", 110, startPosition + counter * distance);
    }
```

Nun haben wir ein Menü erstellt und die zugehörigen Ereignissen mit ausgeführten Aktionen. Das Menü sollte damit wie folgt aussehen:



2 Unter http://www.philipphauer.de/study/se/design-pattern/factory-method.php befindet sich eine gute Beschreibung zu diesem Pattern. Generell kann sich der Einsatz von Patterns als sehr nützlich erweisen. In Ihrem weiteren Studium lernen Sie mehr über Patterns in der Kanonik *Einführung in Software Engineering*.

Im Spielfenster sollte bereits das Hintergrundbild mit Namen `background.jpg` hinzufügt werden (siehe oben). Nun fügen wir die `Action` „Zurück ins Hauptmenü“ bei „Drücken der Escape-Taste“ hinzu. Dies geschieht ebenfalls in der `init`-Methode. Wir erzeugen dazu erstmal eine Entität, deren einzige Aufgabe es ist, auf das Drücken der Escape-Taste zu horchen.

```
// Bei Drücken der ESC-Taste zurueck ins Hauptmenue wechseln
Entity escListener = new Entity("ESC_Listener");
KeyPressedEvent escPressed = new KeyPressedEvent(Input.KEY_ESCAPE);
escPressed.addAction(new ChangeStateAction(Launch.MAINMENUSTATE));
escListener.addComponent(escPressed);
entityManager.addEntity(stateID, escListener);
```

3.3 Entitäten zur Laufzeit hinzufügen und bewegen

Wir möchten nun den Wassertropfen zur Laufzeit hinzufügen. Anschließend soll dieser nach unten „fallen“.

Wir sorgen zunächst dafür, dass an der Mausposition der Wassertropfen erscheint. Wenn also ein Mausklick im Spielfenster erfolgt, so soll eine selbst definierte Action ausgeführt werden, die den Wassertropfen erzeugt.

```
// Bei Mausklick soll Wassertropfen erscheinen
Entity mouseClickedListener = new Entity("Mouse_Clicked_Listener");
MouseClickedEvent mouseClicked = new MouseClickedEvent();
mouseClicked.addAction(new Action() {
    @Override
    public void update(GameContainer gc, StateBasedGame sb, int delta,
            Component event) {
        // Wassertropfen wird erzeugt
        Entity drop = new Entity("drop of water");
        drop.setPosition(new Vector2f(
                gc.getInput().getMouseX(),gc.getInput().getMouseY()));

        try {
            // Bild laden und zuweisen
            drop.addComponent(new ImageRenderComponent(
                    new Image("assets/drop.png")));
        } catch (SlickException e) {
            System.err.println("Cannot find file assets/drop.png!");
        }

        entityManager.addEntity(stateID, drop);
    }
});
mouseClickedListener.addComponent(mouseClicked);
entityManager.addEntity(stateID, mouseClickedListener);
```

Nun soll der Wassertropfen nach unten fallen. Dies erfolgt durch folgenden Code in der `update`-Methode der selbst angelegten Action. Das `LoopEvent` wird mit jedem Frame ausgeführt.

```
LoopEvent loop = new LoopEvent("loop");
loop.addAction(new MoveDownAction(0.5f));
drop.addComponent(loop);
```

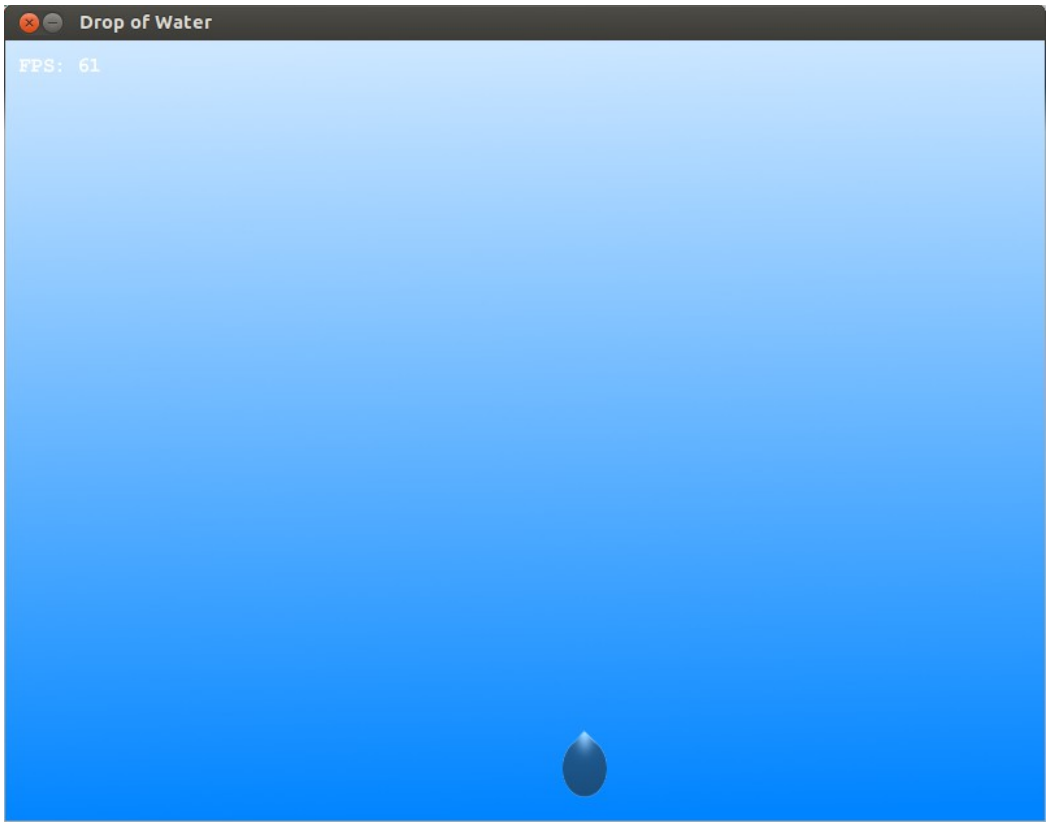
Jetzt fällt der Wassertropfen nach unten; wir reagieren jedoch noch nicht auf das Verlassen des Bildschirms. Dies geschieht immer noch in der selbst angelegten Action:

```
// Wenn der Bildschirm verlassen wird, dann ...
LeavingScreenEvent lse = new LeavingcreenEvent();

// ... zerstoere den Wassertropfen
lse.addAction(new DestroyEntityAction());
// ... und wechsle ins Hauptmenue
lse.addAction(new ChangeStateAction(Launch.MAINMENU_STATE));

drop.addComponent(lse);
```

Damit sind alle gewünschten Features mit `eea` realisiert. Das Spielfenster sollte nun wie folgt aussehen:



Der komplette Quellcode zu „Drop of water“ ist ebenfalls verfügbar als ZIP-Datei.

Voraussetzungen zum Starten des Spiels

Die genaue Installation von Slick 2D und eea wird im Lernportal in einem ausführlichen Dokument beschrieben. Beachten Sie bitte die dortigen Hinweise!