Frameworks: How-To Dungeon

Carsten Gips (HSBI)

How-To Dungeon

In diesem Semester werden Sie im Praktikum schrittweise Erweiterungen in verschiedenen “fertigen” Rogue-like Computerspielen programmieren und dabei (hoffentlich) die Methoden aus der Vorlesung einsetzen können.

Das Projekt “PM-Dungeon” stellt wichtige Bausteine für das Spiel bereit, beispielsweise eine Game-Loop und eine API für das Generieren und Benutzen von Leveln und vieles andere mehr. Im Hintergrund werkelt das etablierte Open-Source-Spieleframework libGDX.

Wir werden uns in diesem How-To einen Überblick verschaffen und einen ersten Einstieg versuchen: Wir programmieren einen einfachen Helden.

Projekt PM-Dungeon

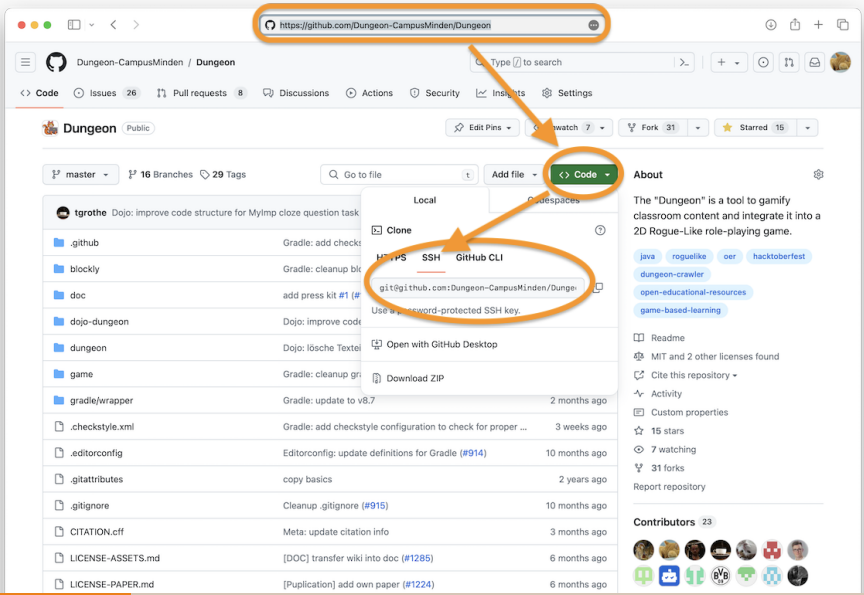
Das Projekt PM-Dungeon entstand in verschiedenen Forschungsprojekten und wurde (und wird) aktiv von Studierenden und wissenschaftlichen Mitarbeitern am Campus Minden entwickelt.

Zuletzt lief das Forschungsprojekt "Dungeon", gefördert durch die Stiftung für Innovation in der Hochschullehre im "Freiraum 2022". Dabei sollten diesmal nicht die Studierenden selbst Code schreiben, sondern die Lehrenden sollen Aufgaben in einer speziellen (von uns entwickelten) Programmiersprache schreiben (können), woraus dann ein fertiges Dungeon-Spiel generiert wird (mit der Aufgabe als Quest o.ä. im Dungeon eingebettet) und die Studierenden können durch das Spielen die Aufgaben lösen.

Sie werden merken, dass trotz klarer Richtlinien und Ideen die Entwicklung in der Praxis doch nicht so einfach ist und dass viele Dinge immer wieder geübt und erinnert werden müssen: Namen von Klassen und Methoden, sinnvolles Javadoc, Dokumentation jenseits des Javadoc, aber auch Commit-Messages und PR-Summaries.

Installation des Frameworks

Sie finden das Projekt auf GitHub: github.com/Dungeon-CampusMinden/Dungeon.



Java: Java SE 21 (LTS)

Wir benutzen im Dungeon-Projekt die aktuelle LTS-Version des JDK, d.h. **Java SE 21 (LTS)**. Sie können sich das JDK bei Oracle herunterladen oder Alternativen ausprobieren. Bitte unbedingt die jeweilige 64-bit Version nutzen!

In der Konsole sollte

```
java -version
```

ungefähr diese Ausgabe erzeugen (ignorieren Sie die Minor-Version, wichtig ist Major-Version: 21 bzw. "LTS"):

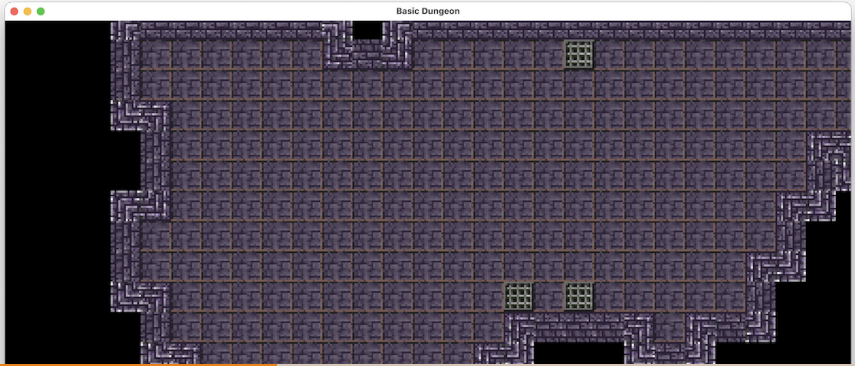
```
java version "21.0.3" 2024-04-16 LTS
Java(TM) SE Runtime Environment (build 21.0.3+7-LTS-152)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.3+7-LTS-152, mixed mode, sharing)
```

Erster Test

Für einen ersten Test gehen Sie in der Konsole in den vorhin erzeugten neuen Ordner `pm-dungeon/` und führen Sie dort den Befehl

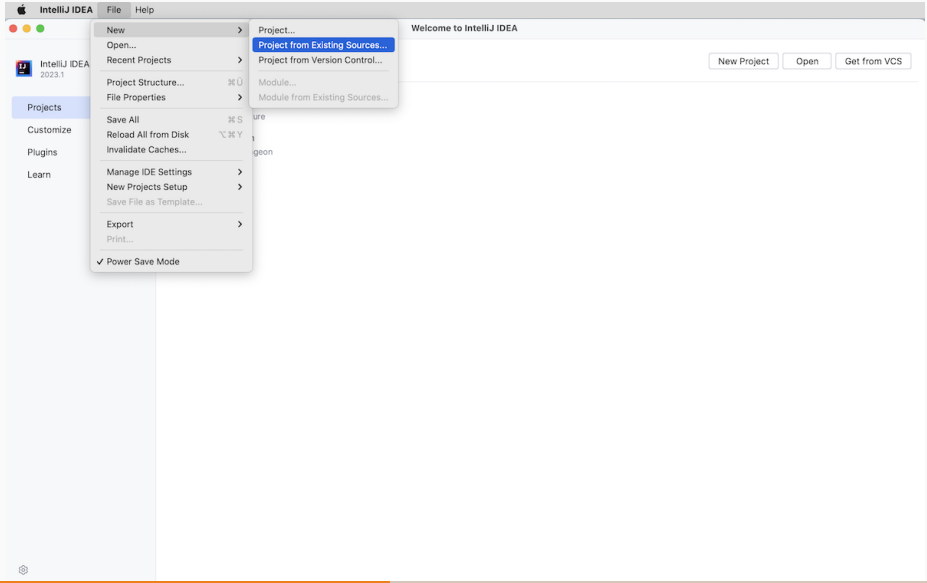
```
./gradlew game:runBasicStarter
```

aus. Dabei sollte das (mitgelieferte) Build-Tool Gradle starten und die benötigten Java-Bibliotheken herunterladen und schließlich das Spiel in einer Minimalversion starten - Sie sollten also ein Level sehen.



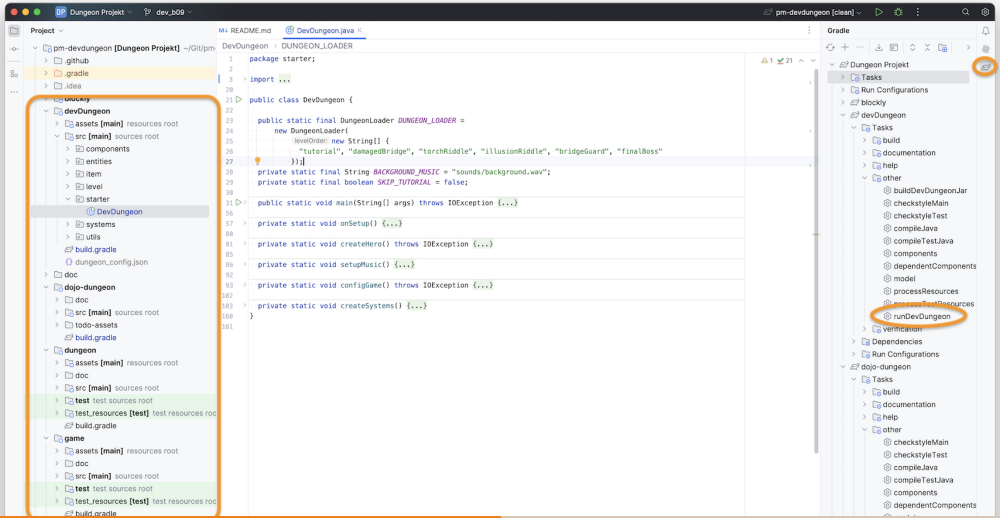
Import in der IDE

Importieren Sie das Projekt als Gradle-basiertes Projekt, dann übernimmt die IDE die Konfiguration für Sie.

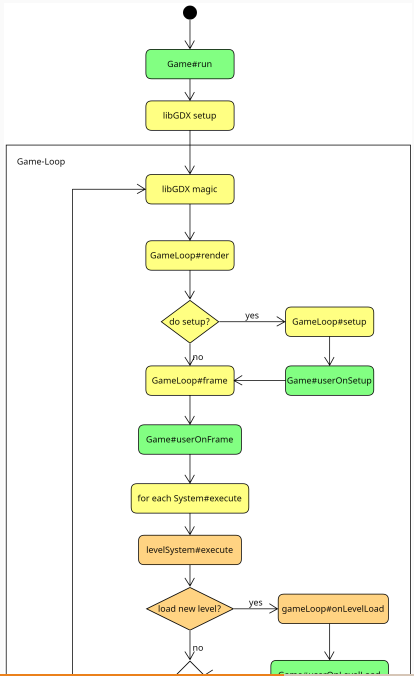


Überblick über die (Sub-) Projekte

Sie finden im Package-Explorer eine Reihe von Unterprojekten (Gradle-Subprojekte). Für PR2 sind eigentlich nur die Subprojekte “dojo-dungeon” und “devDungeon” relevant sowie die Dokumentation in den verschiedenen `doc/`-Ordnern (die derzeit leider noch eine ziemliche Baustelle ist).



Überblick über die Java-Strukturen



Mein Held

Um einen besseren Blick in das System zu bekommen, erstellen wir schrittweise einen eigenen einfachen Helden.

Legen Sie sich im `starter`-Package eine neue Klasse an, mit der Sie das Spiel konfigurieren und starten können:

```
package starter;
import core.Game;

public class Main {
    public static void main(String... args) {
        // Start the game loop
        Game.run();
    }
}
```

In IntelliJ können Sie nun die `main()`-Funktion direkt ausführen, dazu wird im Hintergrund die vorhandene Gradle-Konfiguration genutzt. Mit anderen IDEs funktioniert das vielleicht nicht direkt, dann erweitern Sie einfach die Gradle-Konfiguration um einen entsprechenden Task:

Einschub: ECS oder Entities, Components und Systems

Der Held ist ein Element im Spiel. Diese Struktur muss geeignet modelliert werden.

Unser Dungeon implementiert dabei eine Variante eines Entity Component System (*ECS*) und folgt damit "großen Vorbildern" wie beispielsweise Unity.

Neben verschiedenen Hilfsstrukturen gibt es dabei nur **Entitäten**, **Komponenten** und **Systeme**. Hier werden sämtliche Informationen und Verhalten modelliert.

Entity

Die Idee dahinter ist: Alle Elemente im Spiel werden als *Entität* realisiert, d.h. der Held und die Monster und die Items, die man so finden kann, sind alles Entitäten. Sogar Feuerbälle sind letztlich Entitäten. (Im Prinzip könnten sogar die Boden- und Wandkacheln Entitäten sein - sind es aus Effizienzgründen aktuell aber nicht.)

Eine Entität an sich kann erst einmal nichts und dient nur als Container für *Components*.

Das Spiel kennt alle zu einem Zeitpunkt vorhandenen Entitäten, diese müssen per `Game#add` registriert werden. Man kann die Entitäten über die API abrufen (`Game#allEntities`, `Game#find` und `Game#hero`).

Unsere Basisklasse für Entitäten ist aktuell `core.Entity`.

Nun aber Helden!

Ein Held ist eine Entität

Also legen wir nun endlich einen neuen Helden als Instanz von `core.Entity` an und registrieren diese Entität im Spiel:

```
public class Main {
    public static void main(String... args) {

        // Add some one-time configuration
        Game.userOnSetup(
                () -> {
                    Entity hero = new Entity("Hero");
                    Game.add(hero);
                });

        // Start the game loop
        Game.run();
    }
}
```

Walking mit System

Neue Monster

Wie kann ich ein Monster beim Laden des Levels erzeugen?

Beim Laden eines Levels wird der mit `Game#userOnLevelLoad` registrierte Lambda-Ausdruck ausgeführt. Hier kann man beispielsweise ein neues Monster erzeugen (lassen):

```
public class Main {
    public static void main(String... args) {

        // Add some one-time configuration
        Game.userOnSetup( ... );



        // Create a new monster in every new level
        Game.userOnLevelLoad(first -> {
            Entity fb = new Entity("HUGO");

            fb.add(new PositionComponent(Game.hero().get().fetch(PositionComponent.class).g

            try {
```

Kämpfe wie ein NPC

Wir haben beim Hero über das `PlayerComponent` eine Reaktion auf Tastatureingaben implementiert. Hier könnte man einer Taste auch den Start einer neuen Entität zuordnen, die sich dann automatisch bewegt. Man könnte also Feuerbälle schleudern ...

```
public class Main {
    public static void main(String... args) {


        // Add some one-time configuration
        Game.userOnSetup( () -> {
            Entity hero = new Entity("Hero");

            ...

            PlayerComponent pc = new PlayerComponent();
            pc.registerCallback(KeyboardConfig.FIRST_SKILL.value(), entity -> {
                Entity fb = new Entity("Fireball");

                fb.add(new PositionComponent(entity.fetch(PositionComponent.class).get().pc
```

Wrap-Up

In einem ECS haben wir Entities, Components und Systems.

- Die Entitäten sind nur Hüllen und gruppieren verschiedene Components.
- In diesen Components werden die Werte für die jeweiligen Zustände gehalten.
- Die Systems werden in jedem Durchlauf der Game-Loop aufgerufen und führen dabei ihre `execute()`-Methode aus. Typischerweise iterieren die Systeme dabei über alle Entitäten und verändern die Components der Entitäten.

Denken Sie daran, dass alles in einer Game-Loop läuft, die 30x oder 60x pro Sekunde aufgerufen wird. Sie können in der Regel keine direkte Interaktion zwischen verschiedenen Objekten realisieren, sondern müssen immer den Weg über die Systems gehen.

Schauen Sie gern in die vorhandenen Klassen und Packages und in die Dokumentation hinein:

- Klassen in `game/src/` und `dungeon/src`
- Dokumentation unter `game/doc/` und `dungeon/doc/`

Anregungen für **Spielideen**

- Shattered Pixel Dungeon Rogue Beginners Guide Playthrough
- Shattered Pixel Dungeon Duelist Update!

LICENSE