TDD FÜR DAS POINT&CLICK-ADVENTURE

Marco Kirchner, Philipp Schönert, Anton Krapp, Valentin Kächele, Dennis Dahlem

FH TRIER

INHALTSVERZEICHNIS

1 ÜBERBLICK

1.1 ENGINE

Es wurde entschieden, ein Entity-Component-System zu verwenden, um die Anforderungen eines Point&Click-Adventures damit abzubilden. Die Wahl fiel auf Ashley, da dieses System direkt zu **libGDX** gehört und demzufolge keine Kompatibilitätsschwierigkeiten zu erwarten sind. Eine Entity ist ein abstraktes Konstrukt das Komponenten enthält, welche im Grunde strukturierte Datenspeicher für die jeweiligen Entitäten darstellen. Man kann Entities mit Java-Objekten vergleichen, nur dass unterschiedliche Entities nicht immer die gleichen Components aufweisen.

1.1.1 VERWENDETE BIBLIOTHEKEN

Verwendet wurde **libGDX**, da alle Beteiligten bereits mit dieser Bibliothek gearbeitet hatten und Java als Programmiersprache war ebenfalls allen gut bekannt durch diverse Vorlesungen.

Eine weitere Überlegung war, dass die Bibliothek überschaubar, gut für Anfänger geeignet ist, zahlreiche Methoden für 2D-Programmierung bereitstellt. Außerdem ist sie aktuell und man kann mit ihr plattformübergreifend entwickeln.

1.1.2 BUILD-TOOL

Wir arbeiteten mit **Gradle**, da es standardmäßig von **libGDX** verwendet wird.

1.1.3 GIT

Zur Kollaboration und zur Aufrechterhaltung eines konsistenten Arbeitsstands wurde Git als Versionierungssystem verwendet.

1.1.4 KOTLIN

Es entstand zu Anfang ein fließender Übergang von Java zu Kotlin. Da der Java-Syntax eine gewisse Sperrigkeiteigen ist, wurde beschlossen, Kotlin als Hauptprogrammiersprache zu wechseln, da diese auch auf Java aufsetzt und in vielen Bereichen Vereinfachungen im Code herstellen kann. Ein Grund dafür ist, dass man nicht alles extensiv auf Nullpointer-Exceptions prüfen muss und deren Syntax einfacher zu verwenden ist.

Vereinzelt ist noch Java-Code zu finden (libGDX bedingt).

1.1.5 RENDERING-SYSTEM

Das Rendering-System durchläuft alle erstellten Entities, die eine Transform- und eine Texture-Component besitzen, weil nur diese renderbar sind.

1.1.6 CONTROL-SYSTEM

Dieses System beschäftigt sich mit der Steuerung des Spielercharakters, d.h. Hotspot-Erkennung, Maussteuerung.

1.1.7 MOVEMENT-SYSTEM

Bewegt Entities entlang von Wegpunkten, die in Move-Components der jeweiligen Entities gespeichert sind.

1.1.8 PATHPLANNING-SYSTEM

Berechnet an Hand von einem selbst definierbaren, begehbaren Bereich und einem Start- und Zielpunkt die Wegpunkte, die ein Charakter oder NPC nimmt.

1.1.9 CAMERA-SYSTEM

Das Camera-System ermöglicht das Scrollen eines Schauplatzes, wenn der Hauptcharakter den Bildschirmrand erreicht.

1.1.10 ANIMATION-SYSTEM

Spielt Animationen bei Charakteren und Gegenständen ab.

1.1.11 BLEND-SYSTEM

Dieses System dient visuellen Effekten bei Szenenwechseln oder in gewissen Spielsituationen.

1.1.12 DISTANCE-SCALE-SYSTEM

Damit werden Charaktere skaliert, wenn sie in einem Schauplatz nach hinten oder nach vorn laufen. Dadurch wird ein pseudorealistischer Effekt erreicht.

1.1.13 SCRIPTING-SYSTEM

Kümmert sich darum, dass Skripte korrekt geladen und ausgeführt werden.

1.1.14 SOUND-SYSTEM

Wie der Name schon sagt, verwendet dieses System Methoden zum Abspielen von Musik, Soundeffekten und gesprochenen Dialogen im Spiel.

1.1.15 TIMEOUT-SYSTEM

Hilfssystem für Skripte, um Skripte asynchron gestalten zu können.

1.1.16 USE-SYSTEM

Das Use-System sorgt dafür, dass die Hotspot-Interaktivität gewährleistet ist (take, look, speak, walk).

1.2 SCREENS

Ein Spiel besteht aus vielen unterschiedlichen Screens, die von der Situation abhängen. So ist ein Hauptmenü etwas anderes als ein Inventarbildschirm oder eine Spielszene.

1.2.1 BASE-SCREEN

Das ist dieSchnittstelle für einen Screen, der alle wichtigen Methoden zum Steuern eines Screens enthält, z.B. show, hide, render, update etc.Des Weiteren kann ein Screen einen InputProzessor festlegen, um Input zu verarbeiten (z.B. Tastatureingaben). Der Base-Screen dient als Interface für die anderen Screens, die lediglich ein Overlay darstellen.

1.2.1.1 GAME-SCREEN

Der Game-Screen rendert die jeweilig geladene Spielszene.

1.2.1.2 DIALOG-SCREEN

Der Dialog-Screen zeigt Dialogzeilen für Charaktere an.

1.2.1.3 MESSAGE-SCREEN

Er zeigt allgemeine Nachrichten in einem speziellen Nachrichtenfenster an (Popup).

1.2.1.4 MAINMENU-SCREEN

Zeigt das Hauptmenü an.

1.2.1.5 INVENTORY-SCREEN

Zeigt das Inventar (in Sackform) an.

1.2.1.6 CURSOR-SCREEN

Zeigt den interaktiven Cursor in einer Szene an.

1.3 SKRIPTING

Kein Adventure kommt ohne Intro oder Zwischensequenzen aus. Daher werden Skripte benötigt. Die Skripte werden über einen ClassLoader in den Speicher geladen, um sie zur Laufzeit neu laden zu können.

1.3.1 SKRIPTE

Skripte werden benutzt, um vordefiniertes Verhalten von Charakteren und Situationen zu erzeugen.

1.3.2 HOTSPOTS

Für alle Hotspots gibt es ein Interface **Use**, das die Interaktion mit dem jeweiligen Hotspot regelt. Davon abgeleitet kann man auf diverse Aktionen reagieren (schauen, reden, gehen, benutzen).

1.4 ASSETS

Jedes Spiel hat eine Menge Daten, die für den Ablauf des Spiels erforderlich sind.

1.4.1 GRAFIKEN/SPRITESHEETS

Die Spritesheets liegen im Format .png vor und gleichnamige .atlas-Dateien enthalten die Positionen und Größen der enthaltenen Sprites. Das ist das Grundformat, das wir für alle Grafiken nutzen.

1.4.1.1 GRAFIKASSET-MANAGER

Kümmert sich darum, dass die jeweiligen Spritesheets in den Speicher geladen werden und wieder entfernt werden, abhängig davon, welche Szene geladen wird. Das hat den Vorteil, dass man in der Szene die Assets zur Laufzeit neu laden kann.

1.4.1.2 SKIN

Der verwendete Skin beschreibt das User-Interface des Spiels. Das ist Schriftart, -größe, Buttons, Cursor und andere.

1.4.2 SOUND,MUSIK UND SPRACHAUSGABE

Liegen in den Formaten .ogg und .mp3 vor und werden zur Laufzeit geladen, wenn sie gebraucht werden.

1.4.3 TEXTE

Die Texte für die Dialoge und Ingame-Messages werden über eine .properties-Datei eingelesen.

1.4.4 DIALOGE

Dialogdateien, die im JSON-Format gespeichert werden. Dazu wird ein Dialog-Tool benutzt.

1.4.5 SZENEN

Diese Dateien sind im XML-Format und enthalten alle Entitäten und Komponenten einer Szene.

1.5 SCENELOADER &SCENEWRITER

Diese werden gebraucht, um Szenendaten zu serialisieren und zu deserialisieren. Damit ist gemeint, dass Klassen und Objekte in Text umgewandelt werden bzw. umgekehrt. Dies ist notwendig, damit die Szenendateien gespeichert und geladen werden können.

1.6 TOOLS

Für dieses Spiel wurden zahlreiche Hilfsprogramme geschrieben, die die Arbeit am Projekt wesentlich vereinfachten.

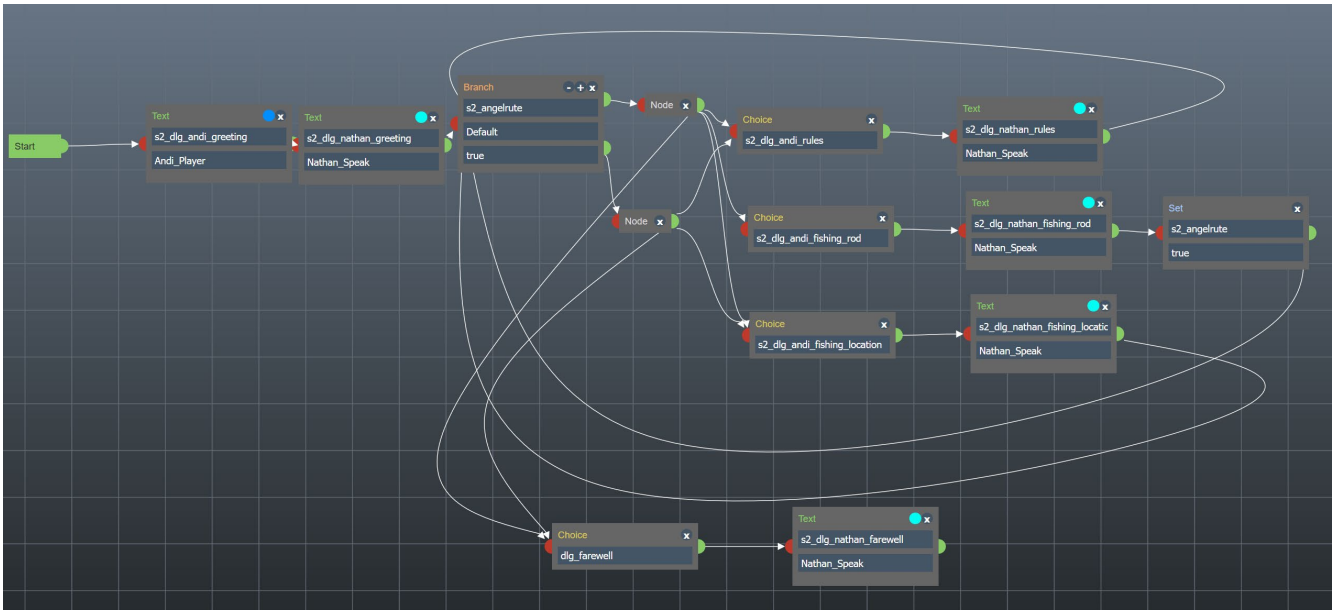
1.6.1 DIALOG-TOOL

Das Dialog-Tool ist in JavaScript geschrieben und kann in jedem modernen Browser ausgeführt werden.

Dialogverzweigungen erfordern eine differenzierte Behandlung von Kontrollflüssen innerhalb eines Dialogs. Zu diesem Zweck wurde der Dialogger von einem Open-Source-Spiel namens Lemma verwendet und für unsere Erfordernisse modifiziert.Dialoge beziehen sich entweder auf eine geskriptete Sequenz oder auf einen kompletten Dialog mit einem bestimmten NPC. Daher ist jeder Dialog in einer eigenen Datei gespeichert (siehe Dateiformate .dlz und .dl).

Features:

- Schriftfarbe im Spiel wird über die eingestellte Farbe von Texten im Tool bestimmt
- Kontrollflüsse für Dialoge können mit Verbindungen zwischen den Dialogtexten erzeugt werden
- Variablen können gesetzt werden, um den Spielverlauf zu beeinflussen
- Verzweigungen werden erzeugt, indem Variablen nach Bedingungenausgewertet werden
- Die Zuordnung der korrekten Entität mit einer bestimmen Dialogzeile erfolgt durch die Angabe des eindeutigen Namens der Entität und der eindeutigen ID der Dialogzeile

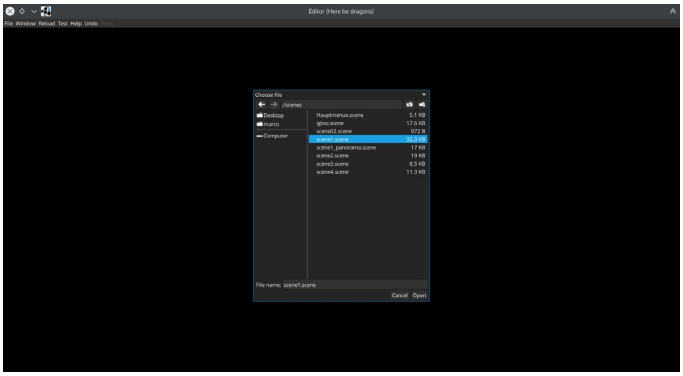


1.6.2 EDITOR

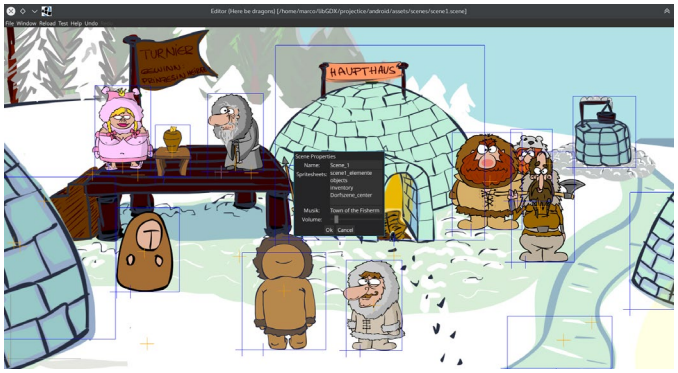
Der Editor setzt auf das Spiel auf und erlaubt das Erstellen und Bearbeiten von Entities und Components zur Laufzeit. Ermöglicht auch das Anpassen von Szeneneinstellungen. Speichern und Laden von Szenen gehört ebenfalls zum Funktionsumfang.
Zu den Funktionen gehören außerdem das Anzeigen und Verbergen der Werkzeuge zur Bearbeitung von Entitäten und Komponenten. Der Editor unterstützt das Neuladen der Grafiken, Sounds und Skripte.
Dies ermöglicht Änderungen an den Skripten, ohne den Editor schließen zu müssen. Dazu werden die Skripte mit Hilfe eines Gradle-Tasks außerhalb des Editors neu kompiliert.

Das Testen des Spiels im Editor wird durch die entsprechenden Funktionen im Menü ermöglicht. Des Weiteren gibt es eine Funktion zum Zurücksetzen des Spielstands, die Variablen werden gelöscht, wodurch die Skripte erneut von Anfang an ausgeführt werden.

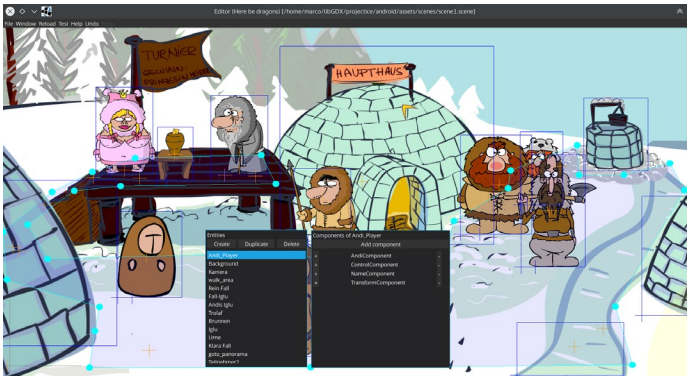
Bei **Undo** wird die letzte Aktion rückgängig gemacht. Das Gegenteil wird durch die **Redo**-Funktion ermöglicht.



Editorfenster mit Dateiexplorer zum Laden einer Szene.



Bearbeitung der Eigenschaften einer Szene
- Name der Szene
- Liste aller Spritesheets für die Szene
- Hintergrundmusik und deren Lautstärke



Typischer Aufbau einer Szene mit den Werkzeugen des Editors.

- Entitäten- und Komponentenfenster
- Editor für die begehbare Fläche mit Ziehpunkten
- Ebenen mit den Markierungen für Hotspots, Entitäten und Texturen

1.6.3 SPRITE-CONVERTER

Dieses Tool konvertiert die Spritesheets vom Starling-Format (.xml) in das von **libGDX**verwendet .atlas-Format. Dies ist notwendig, da das zur Grafikerstellung verwendete Tool keinen Export in das **libGDX**-Format unterstützt.

1.6.4 EXTRACTOR

Verwendet Apache POI, um in Excel-Format verwalteten Textzeilen in das .properties-Format zu konvertieren.

2 TECHNISCHE DOKUMENTATION

2.1 DATEIFORMATE

2.1.1 DIALOGFORMAT (.DLZ UND .DL)

Das .dl-Format wird vom Dialogtool benutzt und enthält neben den Textdaten auch die visuelle Repräsentation.

Das .dlz-Format enthält ausschließlich die Textdaten für das Spiel. Beide Dateiformate basieren auf dem JSON-Format. Grundstruktur des Formats ist ein Array von Objekten:

Attribute:

- id: eine eindeutige „id“ vom Typ String.
- type: einer der folgenden Typen vom Typ String:
    - Start: Einstiegspunkt des Dialogs
    - Node: Leerer Knoten, der übersprungen wird
    - Text: Gesprochener Text
        - name: ID des Strings
        - speaker: Entityname des Sprechers
        - color: Farbe des angezeigten Texts
    - Choice: Multiple-Choice-Knoten
    - Branch: Verzweigung an Hand einer Variable
        - variable: Name der Variablen, nach der verzweigt werden soll
        - branches: Wertpaare (Wert der Variable und ID des Zielknotens)
    - Set: Setzen einer Variable
        - variable: Name der Variablen, die gesetzt werden soll
        - value: Wert der Variablen
- next: die id des Nachfolgeknotens (optional)
- choices: Array von Auswahlmöglichkeiten für den Dialog (optional) mit folgendem Aufbau:
    - id: ID des Nachfolgeknotens
    - index: Index zur grafischen Sortierung der Auswahlmöglichkeiten

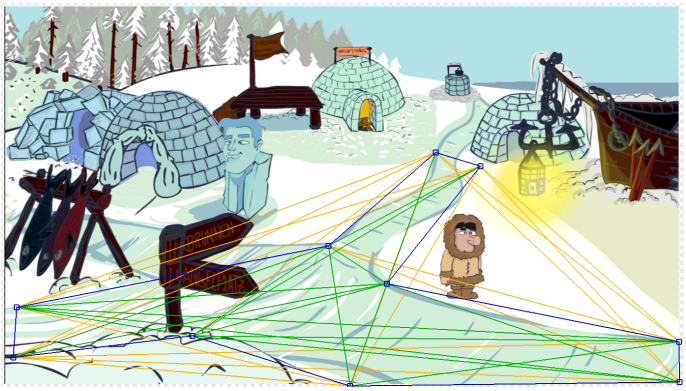
2.1.2 SZENENFORMAT(.SCENE)

Basiert auf dem XML-Format. Der Root-Knoten ist ein <scene>-Knoten und hat ein *name*-Attribut. Die weiteren untergeordneten Elemente sind <music>, <spritesheets>,<entities> ()

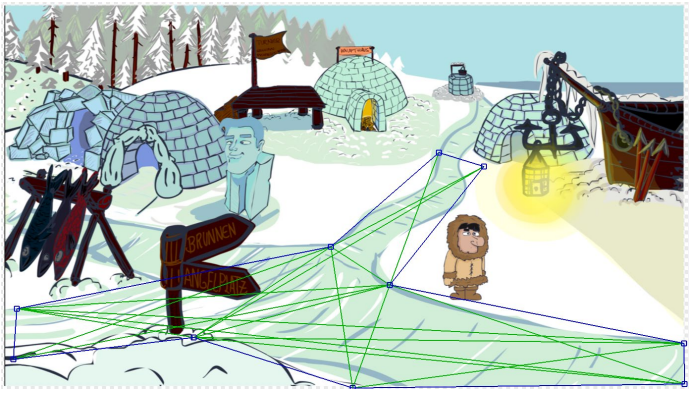
```
<scene name="Name der Szene">
	<music volume="1.0">Stueck 1</music>
	<spritesheets>
		<spritesheet>Spritesheet 1</spritesheet>
		<spritesheet>Spritesheet 2</spritesheet>
		<spritesheet>…</spritesheet>
	</spritesheets>
	<entities>
		<entity>
			<components>
				<TransformComponent>…</TransformComponent>
				<NameComponent>…</NameComponent>
				<TextureComponent>…</TextureComponent>
			</components>
		</entity>
	</entities>
</scene>
```

2.2 PATHFINDING

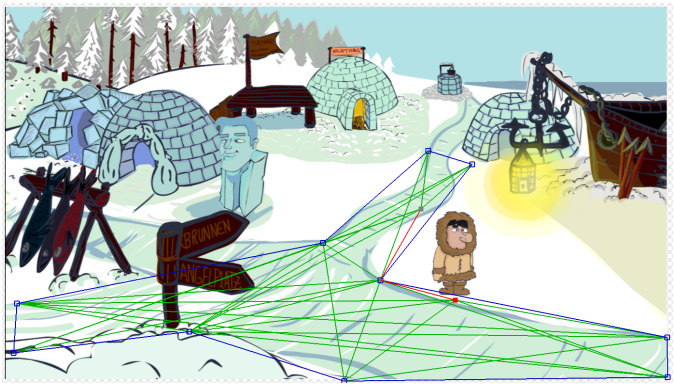
Zur Wegfindung wird ein Algorithmus verwendet, der im ersten Schritt aus den Eckpunkten der begehbaren Fläche einen vollständigen Graphen erstellt. Im nächsten Schritt werden die ungültigen Verbindungen (Kanten), die nicht komplett innerhalb der begehbaren Fläche liegen, entfernt.
Dieser neu erstellte Graph wird nun als Basis für die Wegfindung mit Hilfe des A*-Algorithmus benutzt.



vollständiger Graph mit ungültigen Verbindungen



Graph mit gültigen Verbindungen



Ergebnis der Pfadsuche mit Hilfe des A-Alg.*

Andi ist bereit zum Laufen

3 VERANSCHAULICHUNGEN DER DOKUMENTATION

3.1 ORDNERSTRUKTUR MIT ERKLÄRUNG

.git	Git-Repository		
android	nicht aktiver Android Quellcode (möglicher Android-Support)		
config	Spielkonfiguration: Meter zum Pixelwert		
core	Haupt-Quellcode (gesamtes Spiel)		
	annotations	Property Annotations	
		Property.kt	selbstdefinierte Property-Annotations
	dialog	Dialoge verarbeiten	
		Branch.kt	eine bedingte Dialog-Verzweigung
		Dialog.kt	liest die .dlz-Dateien ein
		Node.kt	Dialog-Knotenpunkt
	ecs	Entity-Component-System (Design-Pattern) wurde getrennt zwischen Daten(components) und Logik (system)	
		components	Inhalt aller Komponenten
			AndiComponent.kt — eine Marker-Komponente zum Generieren eines Andi-Spielobjekts
			AnimationComponent.kt — beinhaltet die Animation von Entitäten
			BlendComponent.kt — ermöglicht das Einblenden der Kamera
			BreathComponent.kt — Simulation von Atmung bei Entitäten
			CameraComponent.kt — beinhaltet Kameraeinstellungen
			ControlComponent.kt — ermöglicht Beschreibung der steuerbaren Entität
			DisabledComponent.kt — deaktiviert eine Entität (inkl. abh. Skripte)
			DistanceScaleComponent.kt — ermöglicht Skalierungvon Entitäten(3D-Effekt)
			HotspotComponent.kt — beschreibt die Eigenschaften eines Hotspots(Ausdehnung, Skript, Mausaktionen)
			IceComponent.kt — Interface aller Komponenten
			InvisibilityComponent.kt — blendet die Entität für den Spieler aus
			MoveComponent.kt — Bewegung einer Entität
			NameComponent.kt — String-ID bzw. Name der Komponente
			PathPlanningComponent.kt — ermöglicht die Generierung von Pfaden entlang der begehbaren Fläche
			ScriptComponent.kt — ermöglicht die Ausführung eines Skripts bei Interaktion mit einer Entität
			SpeakComponent.kt — zeigt Dialogtexte bzw. Hinweistexte direkt oberhalb der Entität an
			TextureComponent.kt — Entität-Textur
			TimeoutComponent.kt — Komponente zur zeitlichen Steuerung von Code zur Laufzeit
			TransformComponent.kt — Position/Rotation/Skalierung/ geometrischeDaten

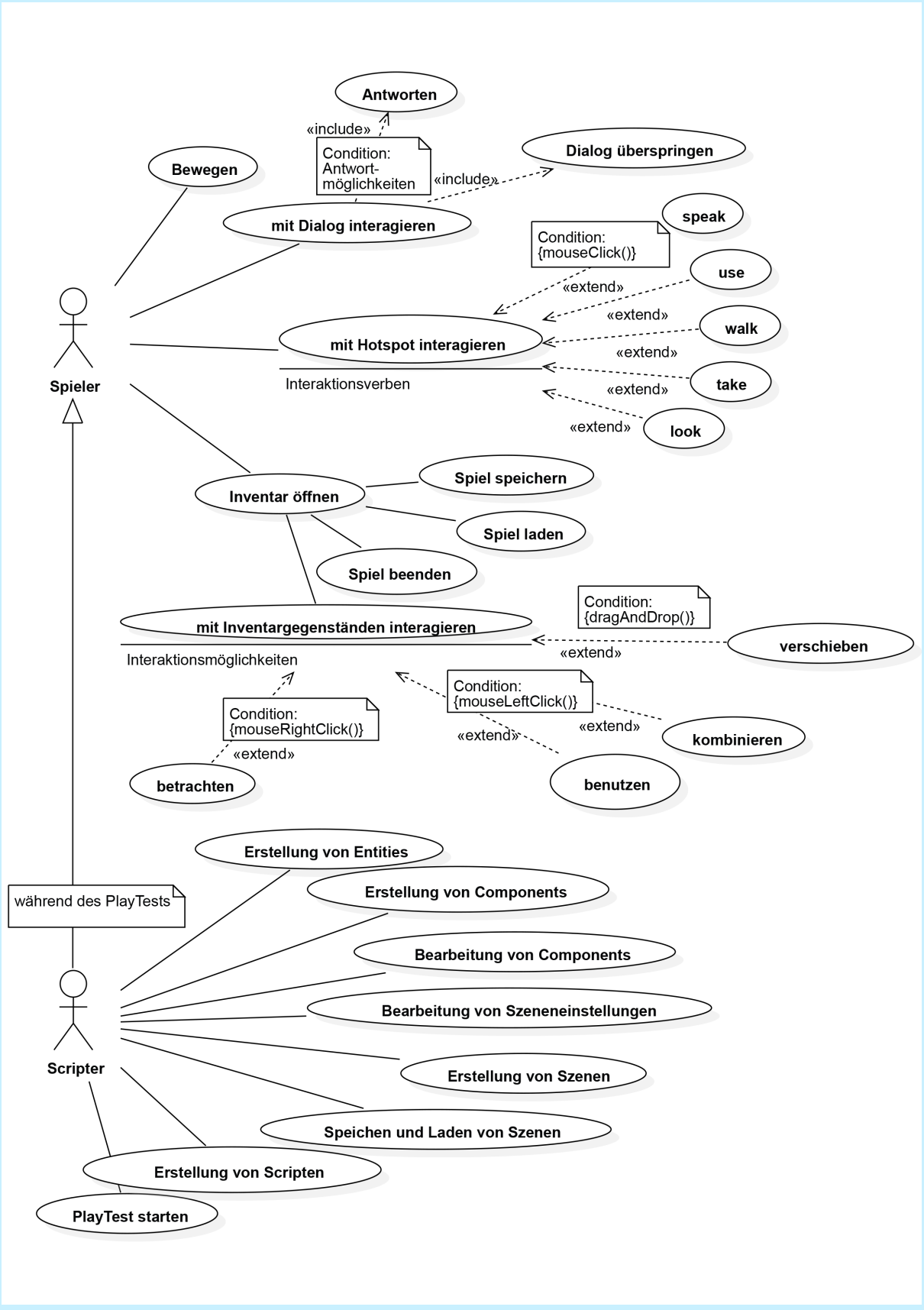
	UseComponent.kt	ermöglichen einer Entität das Benutzen eines Hotspots
	WalkAreaComponent.kt	begehbare Fläche
	WalkingComponent.kt	weist einer Entität eine Laufanimation zu
system		
	AndiSystem.kt	benutzt die Marker-Komponente zum Erstellen eines Andi-Spielerobjekts
	AnimationSystem.kt	animiert Entitätenüber Animations-Komponenten
	BlendSystem.kt	Auf- und Abblendeffekt des Fensters
	BreathSystem.kt	System führt Atmungseffekt an einer Entität aus
	CameraSystem.kt	verschiebt die Follower-Kamera
	ControlSystem.kt	steuert Entitäten mit der Maus
	DistanceScaleSystem.kt	Skaliert bewegliche Entitäten, um 3D-Effekt zu simulieren
	IceSystem.kt	Basisklasse für alle Systeme
	IntervalIceSystem.kt	Basisklasse für Systeme, die in regelmäßigen Intervallen aktualisiert werden
	IteratingIceSystem.kt	Basisklasse für Systeme, die über Entitäten iterieren
	MovementSystem.kt	System führt die Bewegung von Entitäten (meist NPCs) aus
	PathSystem.kt	generiert einen Pfad entlang der begehbaren Fläche
	ReloadAssetsSystem.kt	lädt die Assets neu, nachdem sie aktualisiert wurden
	RenderingSystem.kt	zeichnet die aktuelle Szene
	ScriptingSystem.kt	führt Skripte aus
	SortedIteratingIceSystem.kt	Basisklasse für Systeme, die sortiert über Entitäten iteriert
	SoundSystem.kt	spielt Musik und Soundeffekte ab
	SpeakSystem.kt	Sucht Entitäten, die eine Speak-Komponente haben
	TimeoutSystem.kt	führt Code zu einem späteren Zeitpunkt aus
	UseSystem.kt	führt das Benutzen eines Hotspots aus
Components.kt	Auflistung aller Komponenten	
Families.kt	Kategorisierung von Komponenten	
IceEngine.kt	ECS-Verwaltungseinheit(verwaltet alle Systeme)	
hotspots	Schnittstelle und Verwaltung von Hotspots	
	GotoScene.kt	Hotspot-Skript wechselt die Szene
	Hotspot	Hotspot-Verwaltung
	HotspotLoader	Klassen-Ladetool für Hotspots
	Use.kt	Basisklasse und Schnittstelle von Hotspots
inventory	Verwaltung des Inventars	
	Combinations.kt	Auflistung aller verfügbaren Gegenstand-Kombinationen
	Inventory.kt	Inventar-Verwaltung

	Items.kt	Auflistung aller verfügbaren Gegenstände
screens	alle Spielebenen, die im Spiel vorkommen	
	BaseScreen.kt	Schnittstelle für Ebenen
	BaseScreenAdapter.kt	Adapter für Ebenen (abstrakte Klasse/Designmuster)
	CursorScreen.kt	zeigt den Mauszeiger an
	DialogScreen.kt	zeigt die Texte der Dialoge an
	GameScreen.kt	zeichnet das Spiel
	InventoryScreen.kt	zeigt das Inventar-Fenster an
	MainMenuScreen.kt	zeigt das Hauptmenü an
	MessageScreen.kt	zeigt ein Popup-Fenster an
scripting	Schnittstelle und Verwaltung von Skripten	
	Script.kt	Basisklasse für Skripte
	ScriptLoader.kt	Klassenladetool für Skripte
	ScriptUtils.kt	Hilfsfunktionen für Skripte
utils	sonstige Hilfsfunktionen und sonstige Programmausführungen	
	Assets.kt	Verwaltung von Grafiken/Sprite-Sheets und Assets
	ColorDrawable.kt	Zeichenobjekt derlibGDX-Bibliothek
	CreateEntity.kt	Hilfsfunktion zum Erstellen und Bearbeiten einer Entität
	DefaultSkin.kt	Standardstil für die Benutzeroberfläche
	DelegatingBlockingInputProcessor.kt	Basisklasse delegierender, blockender Input-Prozessoren
	DelegatingInputProcessor.kt	Basisklasse delegierender, aber nicht blockender Input-Prozessoren
	DetachableInputProcessor.kt	Basisklasse für einen deaktivierbaren Input-Prozessor
	DialogListener.kt	ein Ereignisfilter, der aufgerufen wird, wenn ein Dialog beendet wurde
	FileClassLoader.kt	Klassenladetool, der Dateien laden kann und Neuladen der Klassen zur Laufzeit unterstützt
	FreetypeSkin.kt	ein Skin der das Laden von FreeType-Fonts (TTF,OTF) erlaubt
	RoundedRectangleDrawable.kt	Zeichenobjekt ausder libGDX-Bibliothek, das ein abgerundetes Rechteck zeichnet
	SceneLoader.kt	lädt eine Szenen-XML-Datei einer Szene
	SceneWriter.kt	speichert den Zustand einer Szene in eine XML-Datei
	Skin.kt	Definition der domänenspezifischen Sprache für den verwendetenBenutzeroberflächen-Skin
	StringJoiner.kt	Konkatenation von iterierbaren Objekten zu einem String mit Hilfe von einem definierbaren Trennzeichen
	VectorOperators.kt	definiert mathematische Operatoren(+-*) für die

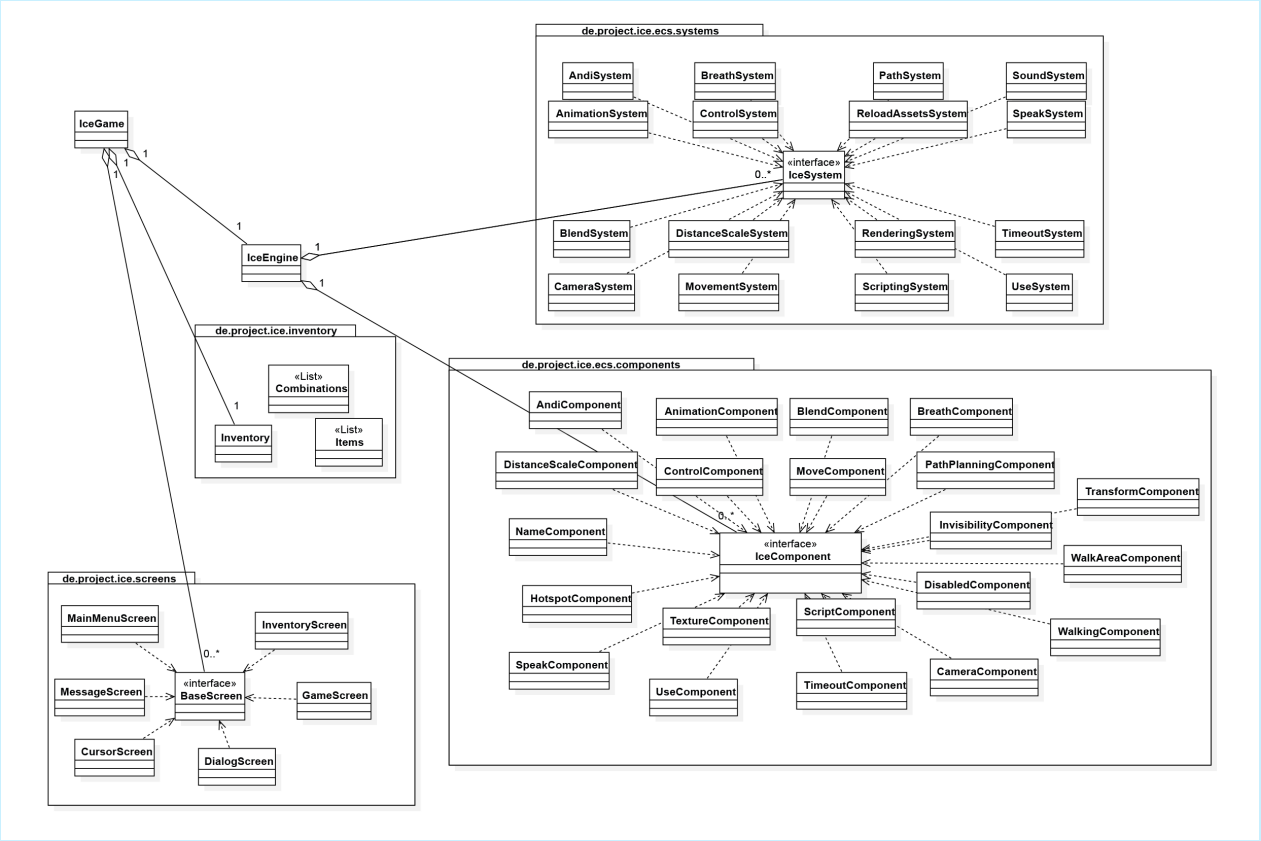
			Vektorklasse derlibGDX-Bibliothek
		IceGame.kt	Hauptanwendungsklasse Haupteinstiegspunkt für die libGDX-Bibliothek des Spiels; verwaltet Engine/Inventar/Ebenen uvm.
		Storage	schlüsselbasierte Speicher für Spieleinstellungen und Spielstände
desktop	Desktop-Launcher startet das Spiel im Windows-Fenster		
	DesktopLauncher.kt		startet die Anwendung im Desktop-Modus
DialogTool	visuelles Tool zum Erstellen der Dialog-Abhängigkeiten		
	lib		JavaScript-Bibliotheken,z.B. jQuery
		…	
		dialogger.js	
		index.css	
editor	Editor-Quellcode		
	editors		alle Editoren für Werte
		AnimationHolderEditor.kt	Editor für Animationszuordnung im Code
		BaseEditor.kt	Basisklasse für alle Editoren
		CursorEditor.kt	Editor für Cursor
		Editors.kt	Auflistung aller Editoren
		FollowTypeEditor.kt	Editor für FollowType(wie die Kamera dem Spieler folgt)
		HolderEditor.kt	Basisklasse für Editoren für Holder
		IntMapEditor.kt	Editor für IntMaps (Zuordnungen, die einen Integer-Wert als Schlüssel besitzen)
		NumberEditor.kt	Editor fürIntegers/Floats/Doubles/Longs
		OrthographicCameraEditor.kt	Editor für die orthographische Kamera
		StringEditor.kt	Editor für Strings
		TextureRegionHolderEditor.kt	Editor für Textur-Teilbereichzuordner im Code
		ValueEditor.kt	Basisklasse für Editoren mit Werten
		Vector2Editor.kt	Editor für 2D-Vektoren
		Vector3Editor.kt	Editor für 3D-Vektoren
	undoredo		
		ModifySceneAction.kt	Harte Kopie von einem Szenenzustand
		UndoableRedoableAction.kt	Schnittstelle für eine spezielle Aktion im Editor
		UndoRedoManager.kt	Verwaltung der Aktionen im Editor der stattgefundenen, aktuellen und zukünftigen Aktionen
	ComponentsWindow.kt		Fenster, das die Komponenteneiner Entität anzeigt
	ComponentTable.kt		Benutzeroberfläche-Tabelle, die die Komponenten einer Entität anzeigt
	EditGameScreen.kt		Ebene für Interaktionen mit der Spielwelt (z.B.Verschieben von Charakteren)
	Editor.kt		Haupt-Methode des Editors (JVM-Haupteinstiegspunkt)
	EditorApplication.kt		Hauptanwendungsklasse Haupteinstiegspunkt für libGDX-Bibliothek des Editors

	EditorScreen.kt	Ebene, die die Fenster des Editors anzeigt
	EntitiesWindow.kt	Fenster, das die Entitäten der Szene anzeigt
	EntityEntry.kt	Listeneintrag für eine Entität (Liste siehe EntitiesWindow.kt)
	FilledPolygonShapeRenderer.java	Modifizierter Form-Zeichner, der konkave Polygone gefüllt zeichnen kann
	PathScreen.kt	Anzeigen und Bearbeiten der begehbaren Fläche
	ScenePropertiesDialog.kt	Dialog zum Anzeigen und Bearbeiten von Szenen-Einstellungen
	ScreenshotFactory.kt	erstellt Screenshot im aktuellem Arbeitsverzeichnis
	TextureList.kt	Liste aller aktuellen und verfügbaren Texturen
extractor	Excel-Dateien werden zu .properties	
	Extractor.kt	Extrahiert die Textzeilen aus einer .xslx-Datei und schreibt sie in eine .properties-Datei
gradle	lokale Gradle-Einstellungen	
	wrapper	
		gradle-wrapper.jar komplette gradle-Distribution
		gradle-wrapper.properties Einstellung der gradle-Distribution
pathlib	Berechnung der Pfadfindung	
	Kalkulation der Pfadfindung	
	PathArea.java	Datenstruktur, die eine Fläche mit Löchern beschreibt
	PathCalculator.java	Klasse, die die Berechnung von Wegfindungen auf einer begehbaren Fläche (PathArea)übernimmt
	PathConnection.java	Datenstruktur, die eine Fläche mit Löchern beschreibt
	PathGraph.java	Datenstruktur, die einen Graphen darstellt
	PathHeuristic.java	Abschätzung von der Strecke zwischen zwei Knotenpunkten
	PathNode.java	Klasse, die einen Knotenpunkt auf einem Graphen repräsentiert
	Shape.java	Klasse,die konkave Polygone darstellen
scripts	Skripte und Hotspots-Skripte	
	hotspot.hotspots	Beinhaltet alle Hotspots-Skripte
		igloo Beinhaltet alle Skripte für die Iglu-Szene
		… Skripte zu den jeweiligen Hotspots
	scripting.scripts	Beinhaltet alle Skripte für Szenen (z.B.gehe dahin/rede mit jenem)
		scene1 Beinhaltet alle Skripte für Szene1
		… Skripte für weitere Szenen
	Projektdatei Informations-Einstellungen	
SpriteConverter	Konvertiert Starling-Spritesheets zu libGDX-Atlas	
	SpriteConverter.java	
build.gradle		
gradle.properties		

3.2 USE-CASE-DIAGRAMM



3.3 KLASSENDIAGRAMM CORE



3.4 KLASSENDIAGRAMM EDITOR