Direkte Signale (innerhalb einer Szene)

# Ziele

Unser Character ...

* sammelt Blumen (wird auf 150% seiner Größe gesetzt – einmalig, nicht stapelbar)
* trifft auf giftige Pilze (schrumpft um 10% - wiederholt sich)

Die Blumen verschwinden, die Pilze bleiben stehen.

**Start und Endzustand des Projektes**

Ein Bild, das Text, Screenshot, Schrift, Electric Blue (Farbe) enthält.

KI-generierte Inhalte können fehlerhaft sein.

# Einleitung: Starterpack

Das Starterpack (siehe Ordner ‚04\_Signals\_begin‘) beinhaltet wieder Character und Welt:

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein. Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein. Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

Sollten diese Elemente mal in einem anderen Spiel wiederverwendet werden, kann es gut sein das Charaktere und Objekte neu sortiert bzw. umbenannt werden müssen. In diesem Fall müsstet ihr eine Kopie des Originalordners erstellen, und dann alle Veränderungen (umbenennen, verschieben, löschen) **direkt im Godot Editor** vornehmen. Also nicht außerhalb, direkt im File Explorer die Änderungen vornehmen.

Für ‚World‘ gilt es ein paar Sachen zu beachten:

* **einzelne Tiles** können direkt aus der *Tile Section* in die TileMap gezeichnet werden
* aus der *Terrain S*ection können **automatisiert Gebiete** gezeichnet werden
  + Wasser kommt zuerst als Fläche (Wellen sind animiert – siehe Abschnitt ‚Ressourcen‘ für ein Beispiel Youtube)
  + Danach kommt Gras als Fläche (hier bietet sich das automatisierte Malen aus dem ‚Terrain‘ an)
  + Danach Pfade als Linie einzeichnen, geht (noch) nicht als Fläche ...
  + Beliebtes Fettnäpfchen: Wasser könnte mit einem ‚physical layer‘ und ‚collision shapes‘ ausgestattet werden, um zu verhindern das der Character über Wasser läuft. **Aber:** TileMapLayer-Instanzen überlappen sich und die Kollisionen des unteren Layers (Wasser) ist weiterhin aktiv– selbst wenn der darüberliegende Layer (Land) eigentlich begehbar sein sollte. Godot erkennt bei der Kollisionsprüfung beide Layer unabhängig von deren Sichtbarkeit oder Layer-Hierarchie.
  + Lösungen:
    - Neues Terrain ‚Rand‘ als Begrenzung verwenden oder
    - Über dem Wasser einen CollisionPolygon2D platzieren (siehe Screenshot)

Ein Bild, das Screenshot, Text, Karte, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Die ‚Game‘ Scene:

* 1. kombiniert die einzelne Spieleelemente (World, Character) und später auch Game\_UI oder Collectibles etc.

# Übung

## Lösche die bestehende Welt in den einzelenen TileMapLayern und entwerfe deine eigene Welt (≈ 15 min)

* Verwende sowohl automatisierte Terrains als auch einzelne Tiles

## Quallen / Energy Flames (separate Szene, AnimatedSprite2D)

* Ziel: Szene einer animierten Qualle / Flame, die beim Kontakt mit dem Spieler verschwinden und ein Signal (Trigger und / oder Daten) sendet
* Animierte Qualle: Benötigt ein AnimatedSprite2D, diesem kann das BigEnergyBall.png zugefügt werden (per Raster Icon). Nachdem die vertikale und horizontale Anzahl der Frames eingegeben wurde, wird die Größe einer einzelnen Kachel berechnet (z.B. 24 x 24). Das ‚Autoplay Icon‘ sorgt dann dafür, dass die Animation nach dem Laden des Spiels auch automatisch startet.

Ein Bild, das Screenshot, Diagramm, Grafiken, Pixel enthält.

Automatisch generierte Beschreibung

# Direkte Signalauswertung (Character ⬄Flower)

Das Godot-Signalsystem ist eine Umsetzung des ‚observer patterns‘ aus der Informatik. Ereignisse, die innerhalb des Spiels stattfinden, senden Signale aus, die von ‚beobachtenden Objekten‘ registriert werden. Beobachterobjekte reagieren auf die Ereignisse, indem sie ‚Handler-Code‘(bzw. die Signalfunktion) ausführen.

Zwischen dem Sender und dem Beobachter besteht eine Kopplung, z.B.:

* Trigger (ein mobiler Character)
* Sender / Akteur (eine Blume, wenn ein Character auf sie trifft)
* weitere Signal-Beobachter (eine UI Anzeige, die Aktionen in einem Spielstand anzeigt)

**Gesamtaufbau des Games (zur Orientierung)**:

Ein Bild, das Screenshot, Multimedia-Software, Software, Grafiksoftware enthält.

KI-generierte Inhalte können fehlerhaft sein.

**Aufbau ‚sendendes Objekt‘**:

* CollisionShapes können nur an ‚Bodys‘ oder ‚Areas‘ hängen, in dieser Situation hat der ‚Area2D‘ Node schon eine Menge vorimplementierter ‚Signale‘
* D.h. die Flower Szene bekommt
  1. ein Sprite2D für die grafische Darstellung
  2. ein Area2D für Signal Initiierung

Ein Bild, das Text, Screenshot, Multimedia-Software, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

**Signaleinstellung**:

* hier braucht es ein Script in der Szene (z.B. im Knoten Poison), um
  1. den emitierenden Node oder den Character zu verändern und/oder
  2. ein Signal für den Rest des Games zu versenden (EventController.emit\_signal (“flower\_met“)

Ein Bild, das Screenshot, Software, Multimedia-Software, Text enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Direkt Kopplung zwischen Signal und Wirkung

Bei Signalen gilt es deren Verknüpfung zu beachten:

* Im folgenden Beispiel befinden sich Signal und Handlercode in der selben Szene; d.h. das Signal kann alles in der der Player Szene (per ‚body‘) verändern und alles in der aktuellen Szene (per ‚self‘). Im aktuellen Beispiel ist das die ‚Flower‘-Szene.
* Der Aufbau ist fehleranfälliger, wenn sich z.B. der Name eines Knotens ändert. Spiele können mehrere hundert Knoten haben und nicht immer ist nachvollziehbar, wie oft ein Knoten mit einem obsoleten Namen noch referenziert wird.

Signalfunktion:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Die letzte Zeile zeigt an was vom Signal bewirkt wird. Folgende Tabelle zeigt verschiedene Wirkungen, die durch das Signal alternativ getriggert werden können, anstelle von body.scale = Vector2(1.5,1.5)

Zum Beispiel:

* self = Blume
* body = Player

|  |  |
| --- | --- |
| body.queue\_free() | **Existenz**: Der Player verschwindet; Achtung: Damit verschwindet auch die Kamera, die am Player hängt. |
| body.visible = false | **Sichtbarkeit**: Player wird unsichtbar; bewegt sich aber noch |
| body.scale \*= Vector2(1.5,1.5)  body.scale = Vector2(1.5,1.5) | **Größe**: Player vergrößert sich (wiederholt) bzw. vergrößert sich (einmalig) |
| body.speed \*= 0.5  body.speed = 0.5  body.speed = 400 | **Geschwindigkeit**: Player-Geschwindigkeit halbiert sich (wiederholt) bzw. halbiert sich (einmalig)  - wird auf 400 Pixel pro Sekunde gesetzt |
| self.queue\_free()oder queue\_free() | Die Blume verschwindet  bzw. alle vorherigen Anweisungen können wieder analog verwendet werden (außer ‚speed‘) |