Quelle: <https://www.youtube.com/watch?v=Rh_8UXjYTn4>

# Ziel

Player sammelt Energieflammen ein, daraufhin verschwindet die Flame und die UI erhöht die Anzahl der gesammelten Flammen (alternativ könnten hier auch Lebenspunkte angezeigt werden).

# Starterpack

Das Starterpack (siehe starter\_collectibles.zip) beinhaltet:

* ‚world‘ scene:
  + eine 2000 x 1000 ‚TileMapLayer‘ Fläche,
  + umgeben von einer 3000 x 1500 ColorRect Fläche
* ‚player‘ scene:
  + einen movable Sprite (mit collision shape und camera nodes)
  + Bewegung vom Sprite basiert auf dem Godot Template für ‚CharacterBody2D‘;   
    (dafür muss die Gravitation Einstellung unter Projekteinstellungen / Physik / 2D auf Null gesetzt werden)
  + add on feature: im player,gd script, Player / CharacterBody2D kann nicht die ‚Spielfläche‘ verlassen
    - velocity: beschreibt die Lageveränderung / Richtung (x,y Koordinaten) pro Sekunde;  
      wird von der physics engine beeinflusst (Gravitation, Trägheit etc.)
    - position: beschreibt die aktuelle Position eines Sprites in absoluten Koordinaten (relativ zum ‚parent node‘); die Position resultiert aus der Anwendung der Geschwindigkeit
    - Funktion clamp schränkt die ‚position‘ auf einen Min und Max Wert ein ...
* ‚game‘ scene:
  + Kombiniert die einzelne Spieleelemente (World, Player) und später auch Game\_UI oder Collectibles

# Schritte

## Energy Flames (neue Szene, animiertes Sprite)

* Ziel: Szene einer animierten Flame, die beim Kontakt mit dem Spieler verschwinden und ein Signal (Datenbotschaft) auslösen
* Aufbau:

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

Automatisch generierte Beschreibung

* Dem AnimatedSprite2D kann das BigEnergyBall.png zugefügt werden (siehe Raster Icon); wichtig ist es die Größe einzustellen (24 x 24) und Autoplay

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

Automatisch generierte Beschreibung

* Jetzt kommt das Signal ins Spiel, dafür braucht die ‚Energy\_Flame‘ Scene ein Script, in dem es den ‚Handler Code‘ (d.h. die Reaktion auf das Signal) aufnehmen kann.
* Fürs Erste kann hier eine simple Ausgabe erfolgen ‚Player made contact‘

## Signals

Das Godot-Signalsystem ist eine Umsetzung des ‚observer patterns‘ aus der Informatik. Ereignisse, die innerhalb des Systems stattfinden, senden Signale aus, die von interessierten Beobachtern abonniert werden. Beobachter reagieren dann auf die Ereignisse, indem sie ‚Handler-Code‘ ausführen. Zwischen dem Sender und dem Beobachter besteht eine lose Kopplung, so dass sie unabhängig voneinander funktionieren können.

**Wie setzt sich eine Signalfunktion zusammen?**Beispiel: \_on\_area\_2d\_body\_entered(body)

🡺 Präfix, emittierender Node-Name und Signal-Name

* Das Präfix \_on\_ signalisiert, dass diese Funktion ein **Signal-Callback** ist, d. h., sie wird **automatisch aufgerufen**, wenn ein spezifisches Signal von einem Node ausgelöst wird
* Der Name area\_2d bezieht sich auf Signalsender -Node (könnte auch flame\_area\_2d heißen)
* body\_entered ist der Name des Signals, das vom Area2D-Node ausgesendet wird
* Das Argument body ist eine Referenz auf den Körper (Node), der das Signal ausgelöst hat (typischerweise ein CharacterBody2D, RigidBody2D, oder ein anderer PhysicsBody2D)

Welcher Knoten soll das Signal abonnieren?

* Originäres Problem: es wird nicht empfohlen direkt auf die Methoden des ‚parent nodes‘ oder nodes in anderen Szenen zuzugreifen.
* Warum? Ein eng gekoppelte Pfad zur einer Methode kann leicht unterbrochen werden, wenn eine Unterknoten im Spiel solo getestet wird. Daher wird die Verbindung von Signalen mit Knoten ‚lose coupling‘ genannt.

🡺 Signal wird also vom Parent ‚energie\_flame‘ abonniert, um upstream Verlinkung zu vermeiden.

Was macht der Handler Code?

extends Node2D

@export var value: int = 1 # Wert einer Energieflame, später anpassbar

func \_on\_area\_2d\_body\_entered(body):

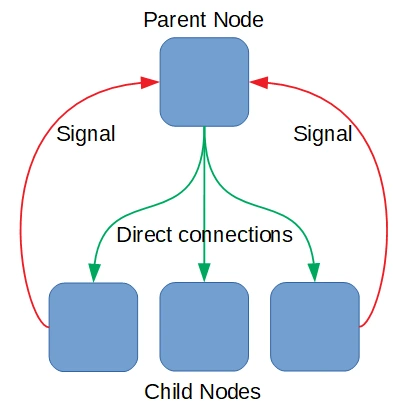
print('body: ', body)

if body is Player:

GameController.flame\_collected(value) #siehe nächster Abschnitt

self.queue\_free()

... siehe auch <https://gdscript.com/solutions/signals-godot/>:



## Zentrales Punktekonto und zentraler Eventmanager (Autoload Singletons)

**Ziel**: einen ‚Energiepunkt‘ je eingesammelte Flamme in die zentrale Punkteübersicht eintragen

**Was sind Autoload Singletons?**

* globale Instanz eines Skripts oder einer Szene
* **Autoload**: wird zum Spielstart geladen und ist somit für alle anderen Skripte und Szenen im Projekt verfügbar
* **Singleton:** es wird immer dieselbe Instanz zur Laufzeit verwendet, d.h. Daten oder Zustände im Singleton bleiben über verschiedene Szenen hinweg persistent

**Wie werden Autoload-Singletons eingerichtet?**

1. Scriptewerden keiner Szene zugeordnet, sondern per Autoload geladen
2. Gehe zu ***Projekt > Projekt-Einstellungen > Autoload***
3. Füge ein Skript oder eine Szene hinzu und gib ihr einen globalen Namen   
   (z.B. GameController)
4. Danach können die Inhalte an jeder Stelle im Projekt über diesen globalen Namen aufgerufen werden

**Typische Singletons:**

* *Game Controller* (z.B. Spielerstatus wie Gesundheit, Inventar abspeichern
* *Audio Controller* (z.B. Wechsel der Hintergrundmusik, Lautstärkeregelung, Soundeffekte wie Schritte, Einstellungen im Audio Controller bleiben über Szenen hinweg erhalten)
* *Input Controller* (z.B. Pause, Mini-Map, Erklärungen, Inventarübersicht anzeigen)

Beispiele: ‚event\_controller‘ und ‚game\_controller‘ Scripte

1. event\_controller definiert ein Signal

extends Node

signal flame\_collected (value:int)

1. game\_controller speichert Punkteveränderungen UND sendet ein Signal mit dem neuen Punktestand (unter Nutzung des event\_controller)

extends Node

var total\_flames: int = 0

func flame\_collected (value:int):

total\_flames += value

EventController.emit\_signal("flame\_collected", total\_flames)

## Mehrfache Instanzen des Energy Flame Nodes in der World Scene aufrufen

* Beliebig viele Instanzen sind möglich (STRG + D)

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

Automatisch generierte Beschreibung

* um Spielinformation immer im Sichtfeld zu haben, wird die Game\_UI scene
* alternativ kann ein Extra ‚CanvasLayer‘ Node verwendet werden (enable die Einstellung ‚Follow Viewport‘)

## Weitere Nutzung von Signale

Beispiel

* Spieldauer wird angezeigt: Timer Node sendet ein Signal im Sekundentakt um ein Update der Zeitanzeige zu triggern
* Per Escape Taste erscheint dann ‚Pause‘ und die Zeitanzeige bleibt stehen:  
  InputController (ein Autoload Singleton) beobachtet Tastatureingaben und triggert ein Signal sobald die Escape Taste betätigt wird

Oder auch ...

* Mit jeder Flame wird der Sprite größer oder schneller

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

Automatisch generierte Beschreibung