Opgave/tutorial – skift mellem scener i Godot med lyd og dataoverførsel

For at kunne sikre en god dynamisk måde at skifte scener på i Godot, er man nødt til at benytte en parent node/parent scene til ens scener som står for selve sceneskiftet. Ved at gøre det på den måde vil det også fremadrettet være muligt at overføre data fra den ene scene til den næste uden at skulle lave en masse global-data som alle scener har adgang til (på store projekter vil programmøren let drukne i datamuligheder og performance af spillet vil falde grundet ekstra brug af hukommelse).

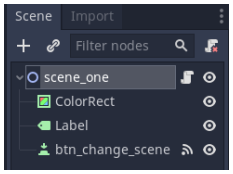
Koden til denne opgave er stærkt inspireret fra følgende tutorial (nr. 1 ud af 3) fra nettet:
https://www.youtube.com/watch?v=XHbrKdsZrxY&t=179s

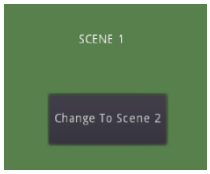
Opgave – skift scene:

1. Opret et nyt projekt, navnet er frivilligt

2. Opret en 2D scene med et passende navn f.eks. ”scene_one”.

3. Tilføj tre ”Child notes” (ved at højreklikke på scenen) af typerne: ColorRect, Label og Button (en af gangen)

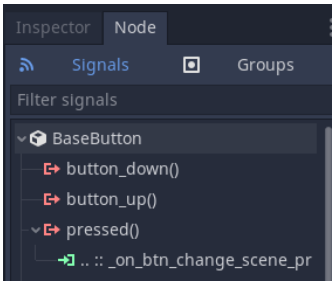
   

   Stræk ColorRect elementet ud så det danner en farvet baggrund.
   Sæt Label elementet midt i vinduet og skriv en kort tekst.
   Placer Button elementet midt i vinduet et sted og giv den ligeledes en sigende tekst f.eks. ”Change scene” og et navn.

4. Lev en nye scene der er magen til den første, kald den evt. ”scene_two”. Vælg en anden farve til denne scenes ColorRect.

5. De to scener skal nu tilknyttes til samme script, så højreklik på en af scenerne og tilføj et nyt script med et fornuftigt navn f.eks. ”level.gd”. Højreklik dernæst på den anden scene og tilføj det samme script.

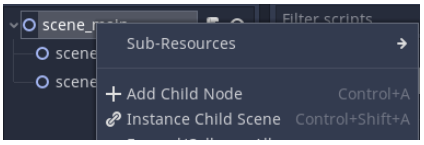
6. I scriptet skal tilføjes den funktion som kaldes når der klikkes på en af de to knapper. Den dannes lettest ved at vælge en af knapperne, og dernæst ”Signal” under fanebladet ”Node”.
   Dobbeltklik på ”pressed()” og Godot forslår et navn til funktionen hvilket er en sammensætning af knappens navn og det signal den reagerer på f.eks. _on_btn_change_scene_pressed
   Accepter navnet ved at klikke ”connect” og funktionen skrives nu ind i scriptet.
   (Der står pass som udgangspunkt i stedet for print(…) hvilket er tilføjet efterfølgende og ”level_name” beskrives senere)

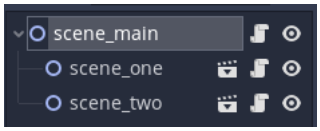
   


```
func _on_btn_change_scene_pressed():
    print("change from scene " + level_name)
```

7. Kopier navnet på funktionen og gør nu det samme for den anden knap. Hvis Godot forslår et andet funktionsnavn (hvilket den vil medmindre navnet på de to knapper er ens), paste navnet ind på den eksisterende funktion således at når der trykkes ”connect”, så vil begge knapper kalde den samme funktion. Hvis der dannes en ny funktion i scriptet, er der gået noget galt og man skal disconnect den nye funktion (under ”signals”) og prøve igen.

8. Der laves en tredje scene der gives et navn og tilføjes et nyt script f.eks. ”scene_main” og scene_main.gd. De to andre scener tilføjes nu som ”Child Scene” til denne scene ved at højreklikke på scenen og vælge ”Instance Child Scene”. I listen der vises, vælges de to scener således at de nu står som underscener.

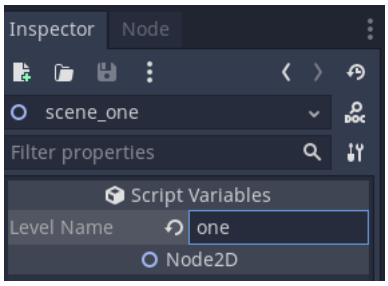




9. Det resterende der skal laves, er kode. I top scenens script (scene_main.gd) skal skrives en funktion der håndterer sceneskiftet og i underscene scriptet (level.gd) skal knapfunktionen vi tidligere lavede, kalde denne funktion.
   Disse to funktioner ligger ikke i samme script og derfor skal der benyttes nogle nye teknikker: signal og export (de er kun beskrevet kort her i opgaven).
   Signal bruges som når man klikker på en af knapperne ved at kalde en bestemt funktion. Ved at lave vores eget signal kan vi selv bestemme i vores script hvornår signalet skal sendes.
   Export benyttes til at gøre en variabel tilgængelig for flere noder eller scripts. Funktionen der laver sceneskiftet skal modtage en værdi (den har en input parameter) så derfor skal vi bruge en export variabel.
   Indsæt i toppen af level.gd scriptet (under extends…), følgende linjer kode →
   Default værdien af level_name kan sættes til hvad som helst, her er den sat til ”level”.

```
signal level_changed(level_name)

export (String) var level_name = "level"
```

10. Klik nu på den første underscene og tilføj et navn (f.eks. one) til Level Name der står til højre i ”inspector” under ”Script Variables”. Gør det samme for den anden underscene f.eks. med navnet two.
    Disse navne skal afgøre hvilken scene det er som kalder sceneskfitfunktionen.



11. Der tilføjes nu den kode til knapfunktionen der sender det signal der kalder sceneskiftfunktionen. Den kan sættes ind efter print(…) linjen således:

```
func _on_btn_change_scene_pressed():
    print("change from scene " + level_name)
    emit_signal("level_changed", level_name)
```

12. Sceneskriftfunktionen og en variabel skrives dernæst i scene_main.gd. Koden til variablen er:

```
onready var current_level = $scene_one
```

Den benyttes til at huske hvilken scene der er den aktive scene.
Sceneskiftfunktionen ser således ud:

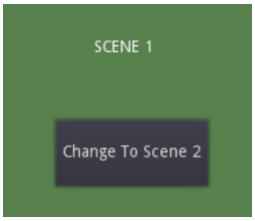
```
func handle_level_changed(current_level_name: String):
    var next_level
    print(current_level_name + " handle")
    match current_level_name:
        "one":
            next_level = load("res://scene_two.tscn").instance()
        "two","first":
            next_level = load("res://scene_one.tscn").instance()
        _:
            return

    add_child(next_level)
    next_level.connect("level_changed", self, "handle_level_changed")
    current_level.queue_free()
    current_level = next_level
```

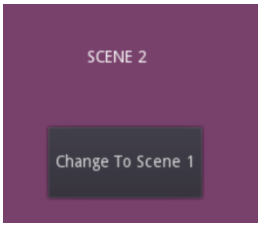
Funktionen modtager et input som er enten ”one” eller ”two” alt efter hvilken scene der har kaldt den. I kodelinjen match… der svarer til switch i mange andre sprog, bestemmes hvilken scene der nu skal skiftes til.
Scenen loades in i variablen next_level og tilføjes til scene_main som underscene med kommandoen add_child(…). (Den er allerede en underscene fordi alle scener der skal vises er gjort til underscene fra start af hvilket afviger fra den tutorial der blev benyttet til denne opgave).
”next_level.connect(…)” sørger for at funktionen ”handle_level_changed” kaldes når der sendes signalet ”level_changed” fra knapfunktionen, altså den sikre at vores knapper virker.
Dernæst frigives den nuværende scene så den ikke længere er aktiv og skiftet sker.
”current:level = next_level” skrifter variablens data omkring hvilken scene der er den aktive.
(Print(…) kan undlades (det kan den også i den anden funktion))

13. Som det sidste skal vi kalde funktionen _ready(…) i scene_main.gd for at sætte hvilken scene der skal startes med. Koden til det ser sådan ud:

```
func _ready():
    print("main_ready")
    handle_level_changed("first")
```

Det skal nu være muligt at starte vores projekt og scene_one vises. Når der trykkes på knappen skiftes til scene_two og tilbage igen når knappen på scene_two aktiveres. Det ser ca. sådan ud:





opgave – skift scene er færdig…

LYD: Næste opgave går ud på at tilføje en lydeffekt til sceneskiftet hvilket er noget der ofte bruges i spil. Opgaven tager udgangspunkt i følgende tutorial: https://www.youtube.com/watch?v=VcI22IKoT_E&t=42s

Det er nr. 2 tutorial ud af de tre og foruden introduktion til lyd omhandler den også hvordan man kan lave en grafisk fade in og out effekt. Dette er ikke med i denne opgave men kan være et interessant selvstudie dog med den lille advarsel af der har vist sig at være afvigelser fra hvordan koden i disse tutorials virker og hvad der faktisk fungerer med ovenstående kode. Hvad årsagen til disse forskelle er, har desværre ikke været muligt at finde, så blot vær opmærksom på at selve kodningen kan drille undervejs og koden skal tilpasses.

Inden vi starter på selve opgaven, skal I finde en lydfil med en passende lyd. Den lydfil som er brugt i kodeeksemplerne hedder ”mixkit-arcade-game-opener-222.wav” og blev fundet på følgende website:

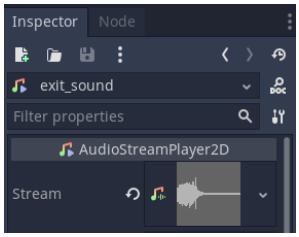
https://mixkit.co/free-sound-effects/game/

Opgave – sæt lyd på sceneskiftet

1. Træk den valgte lydfil ind i Godot under FileSystem

2. Under din første scene (scene_one), tilføj en ny ”Child Node” af typen AudioStreamPlayer eller AudioStreamPlayer2D. Giv den et passende navn f.eks. exit_sound.

3. I ”Inspector” vinduet findes ruden ”Stream”, træk din lydfil fra FIleSystem og over i denne rude.
   For at høre lyden fungerer klikkes ON i ruden ”Playing” lige neden for.



4. Nu skal lyden kopieres til vores anden scene (scene_two). Højreklik på denne scene og vælg ”Merge from scene”. I det viste vindue, vælg scene_one og ”open” og dernæst dernæst exit_sound og ”OK”. Nu skal der ligge en kopi af exit_sound under scene_two.

5. Nu skal lyden kaldes hver gang vi klikker på ”skift scene” knappen. Fra forrige opgaver havde vi allerede funktionen ”_on_btn_change_scene_pressed()” og her tilføjes nu det kald er starter afspilningen af lyden: $exit_sound.play()
   Udkommenter følgende linje koden i samme funktion:# emit_signal("level_changed", level_name)
   Prøv at kør programmet nu…

6. Hvis alt fungerer korrekt, bliver lyden afspillet men vi skifer ikke scene. Fjern udkommenteringen af # emit_signal("level_changed", level_name) og prøv igen. Nu skifter scenen igen men der er ingen lyd. Problemet er at vi skal vente med at kalde scene skiftet til efter lyden er blevet færdig, ellers vil vi nedlægge scene og dermed lyden før den kan nå at blive afspillet.

7. For at vente med sceneskiftet tilføjes der derfor to linjer kode før emit_signal. ”Yield” kaldet gør at programmet venter til vores lyd når til status ”finished” før den fortsætter og kalder dernæst emit_signal. Tilføj koden og se at det virker.

```
if $exit_sound.playing:
    yield($exit_sound, "finished")
emit_signal("level_changed", level_name)
```

I nr. 2 tutorial laves en funktion der hedder cleanup som står for afspilningen af lyd men den fremgangsmåde virkede ikke da metoden quede_free() ikke kunne kaldes fra level.gd modsat hvad der er vist i vidioen??? Årsagen til dette blev aldrig fundet men har nok noget at gøre med en forskel i den måde scenerne sidder sammen.

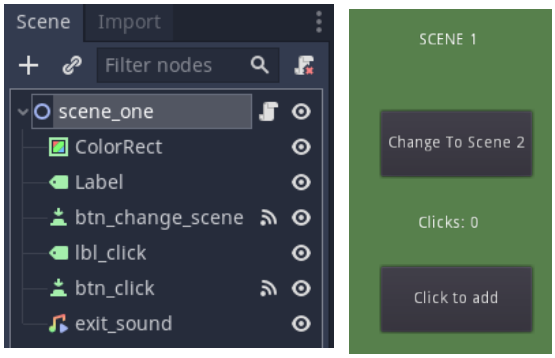
Opgave – sæt lyd på sceneskiftet er færdig…

DATA FRA SCENE TIL SCENE: Den sidste opgave omhandler overførsel af data fra en scene til en anden. Den tager udgangspunkt i tutorial nr. 3: https://www.youtube.com/watch?v=N4iV1L6xb04

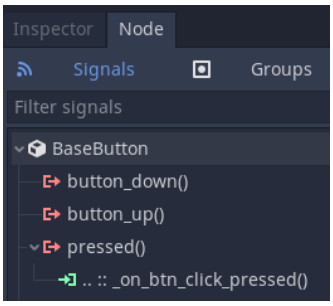
Koden i denne tutorial virker helt som forventet og der er derfor mindre grund til at se videoen.

Der er kodeeksempler på sidste side til opgave 4 til 6.

Opgave – overfør data fra scene til scene



1. På første scene (scene_one) start med at duplikere den label og knap som allerede findes (Label og btn_change_scene).
   Marker dem begge, højreklik og vælg ”Dublicate”.
   Giv dem dernæst et nyt navn f.eks. lbl_click og btn_click og en fornuftig tekst (se billede).
   Inde på scene_one → 2D, flyt dem ned på scenen så de ligger under den gamle knap og label.



2. Da den nye knap stadigvæk kalder den samme ”pressed” funktion (_on_btn_change_scene_pressed()), klik på knappen og under ”Node” ude til højre, disconnect den gamle funktion og connect til den funktion som Godot foreslår ”_on_btn_click_pressed()”.

   Kopier knappen og label til scene_two v.h.a. ”Merge from scene” (se evt. pkt. 4 fra forrige opgave) således at begge scener nu har en click knap og label.

3. Næste skridt er at gemme hvor mange gange der er trykket på click knappen (det er blot for at have noget data vi kan sende fra scene til scene).
   I stedet for at oprette en enkel variabel til dette formål, er det overordnet set mere smart at samle alle vigtige værdier som parameter i et fælles ”dictionary” hvilket gøres som vist i koden til højre. På den måde kan man tilføje nye parameter under de gamle og samle dem alle under level_parameters.

```
var level_parameters := {
    "clicks_p": 0
}
```

4. I funktionen ”_on_btn_click_pressed”, tilføj to linjer kode, den første der opdaterer vores parameter og den næste der opdaterer teksten på vores label. Kør programmet og se at tælleren fungerer og teksten på vores label tælles op. Hvad sker der hvis vi skifter scene?
   Ja de to tællere er uafhængige hvilket vi vil ændre på ved at overføre data fra en scene til den anden.
   Der er mange programmører (incl. ham fra videoen) der vil flytte de to ovenstående linjer kode ind i sin egen funktion for at holde koden mere overskuelig. Selv har jeg oprettet en funktion kaldet add_click(new_value: int) som håndterer opdateringen og blot kaldt denne funktion fra ”_on_btn_click_pressed”.

5. For at overføre data fra den gamle scene til den nye skal der laves en ny funktion i scene_main.gd der tager den gamle og nye scene som inputparameter. Giv funktionen et sigende navn f.eks. ”transfer_data_between_scenes” og inputparametrene ”new_scene” og ”old_scene”. Der behøves nu kun en linje kode i funktionen for at overføre data da man kan tilgå level_parameter i begge scener således: <navn på scene>.level_parameters (f.eks. old_scene.level_parameters).
   I ”handle_level_changed” (funktionen) tilføjes nu et kald til ovenstående funktion lige efter current_level.connect(…) koden således:
   ```
   transfer_data_between_scenes(current_level, next_level)
   ```
   Kør programmet og se om der overføres data?
   Det skulle fungere nu men der er en lille fejl, teksten på vores label siger først ”Clicks: 0” for derefter at vise det korrekte antal kliks når man tilføjer et nyt klik.

6. Vi behøver derfor også at opdatere teksten på vores label og for at gøre koden overskuelig, samler vi både dataoverførslen og label opdateringen i en funktion i level.gd.
   Skriv en ny funktion i level.gd der tager et dictionary som input f.eks. kunne den hedde ”load_new_parameters(new_level_parameters: Dictionary)” og skriv to linjer kode der først overføre de nye level_parameters og dernæst opdaterer vores label.
   Tilbage i funktionen ” transfer_data_between_scenes” fra tidligere, skift nu koden ud så den kalder denne funktion på den nye scene (new_scene. load_new_parameters(….)).
   Kør programmet og hvis alt er gået godt, vil data blive overført og teksten på vores label opdateret korrekt.

Hvis der er tid til det kan I arbejde med at tilføje flere parameter og se at det virker eller tilføje en lydfil til scene opstart hvilket er forklaret i nr. 2 video. Det er også muligt at lave fade in og out effekter til jeres scener hvilket også er forklaret i denne video. Vær dog hele tiden opmærksom på at der har været en række kode problemer jeg har stødt på med denne tutorial så alternativ kode kan vise sig at være nødvendigt.

En lille sidste ting, så har jeg opdaget at der dannes muligvis kopier af scene_two hver gang man skifter til denne scene fordi den oprindeligt ikke skulle ligge under scene_mail. Et problem som jeg tidligere har omtalt og da jeg ikke har fået scene fjernet fra scene_main grundet det er lidt bøvlet (det er min umiddelbare oplevelse), så har jeg lavet et lille hack ved at sætte en ekstra linje kode ind i scene_main.gd der fjerne scene_two med quede-free(). Koden ser således ud:
(Det er ikke kønt men det virker 😊)

```
func _ready():
    print("main_ready")
    $scene_two.queue_free()
    handle_level_changed("first")
```

Kodeeksempler fra mit projekt:

Opgave 4:

Fra level.gd

```
func add_click(new_value: int):
    level_parameters.clicks_p = new_value
    $lbl_click.text = "clicks: " + str(level_parameters.clicks_p)


func _on_btn_click_pressed():
    add_click(level_parameters.clicks_p + 1)
```

Opgave 5:

Fra scene_main.gd

```
func transfer_data_between_scenes(old_scene, new_scene):
    new_scene.level_parameters = old_scene.level_parameters
```

```
add_child(next_level)
next_level.connect("level_changed", self, "handle_level_changed")
transfer_data_between_scenes(current_level, next_level)
current_level.queue_free()
current_level = next_level
```

Opgave 6:

Fra level.gd

```
func load_new_parameters(new_level_parameters: Dictionary):
    level_parameters = new_level_parameters
    $lbl_click.text = "clicks: " + str(level_parameters.clicks_p)
```

Fra scene_main.gd

```
func transfer_data_between_scenes(old_scene, new_scene):
    new_scene.load_new_parameters(old_scene.level_parameters)
```