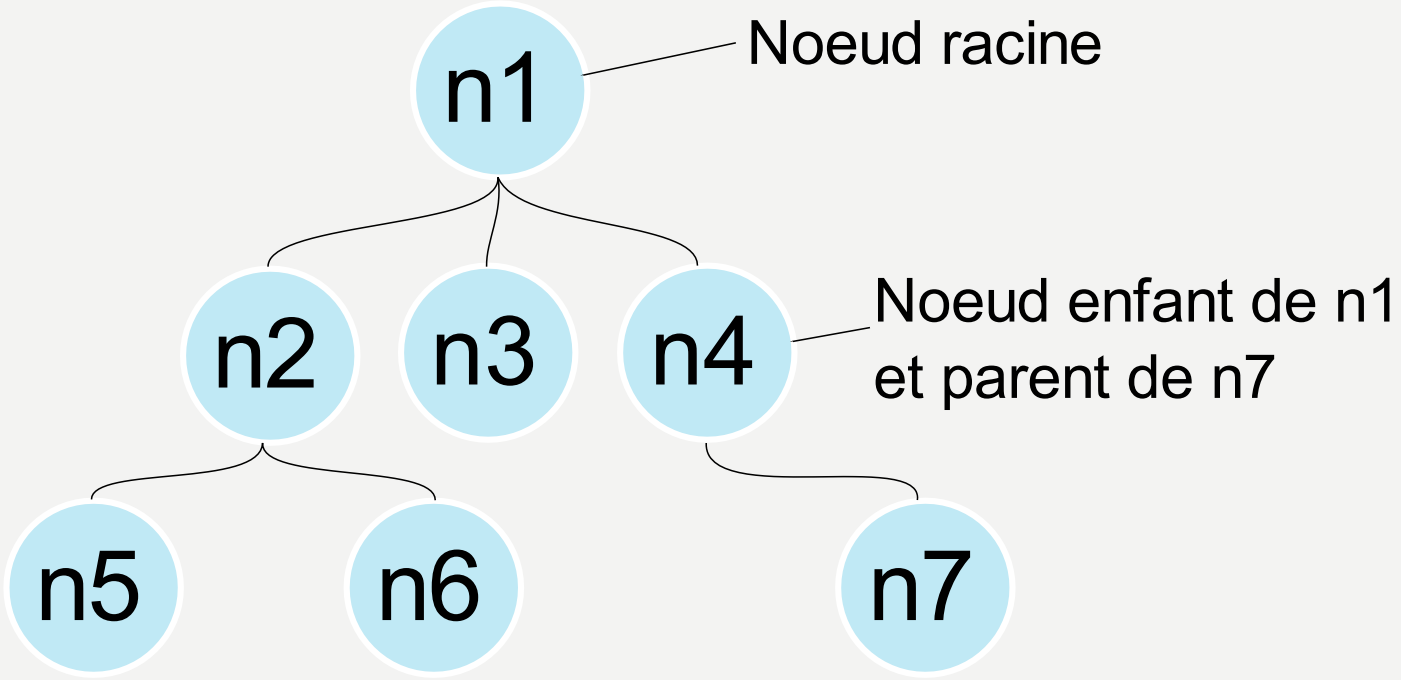
GODOT SCÈNE

SCÈNE

Noeuds

Godot regroupe de nombreux objets tels que des Sprite, des Line2D, des CollisionShape, … et chacun de ces objets peuvent être vus comme des nœuds. En effet Godot organise ses objets sous la forme d'un arbre comportant un nœud racine. Dans l'exemple ci-dessous, on peut voir le nœud racine n1 et ses nœuds enfants n2,n3 et n4. Ces même nœuds enfants peuvent également avoir des nœuds enfants comme n2 avec ses nœuds enfants n5 et n6



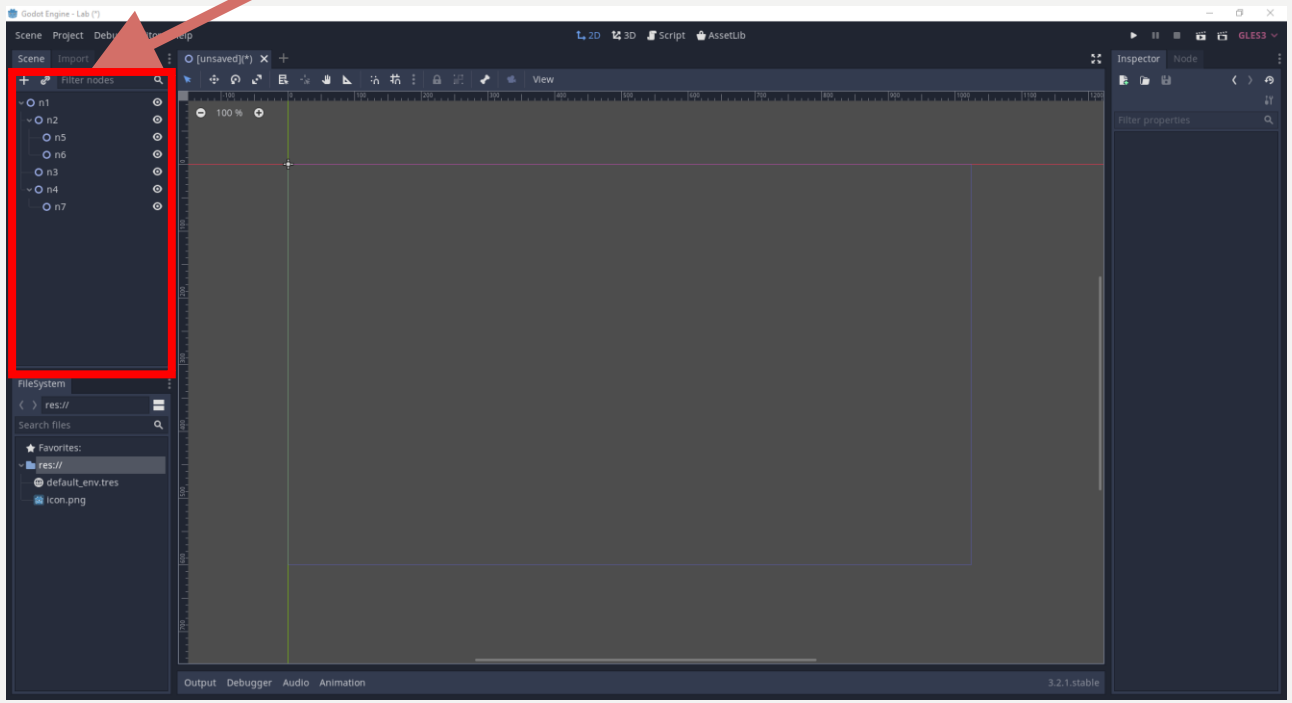
Organisation de la scène en arbre de Godot.

SCÈNE

Quezaco?

Dans Godot une scène correspond donc à un arbre de nœuds où chaque nœud peut être n'importe quoi.

Représentation de la scène sous
la forme d'un arbre de noeuds

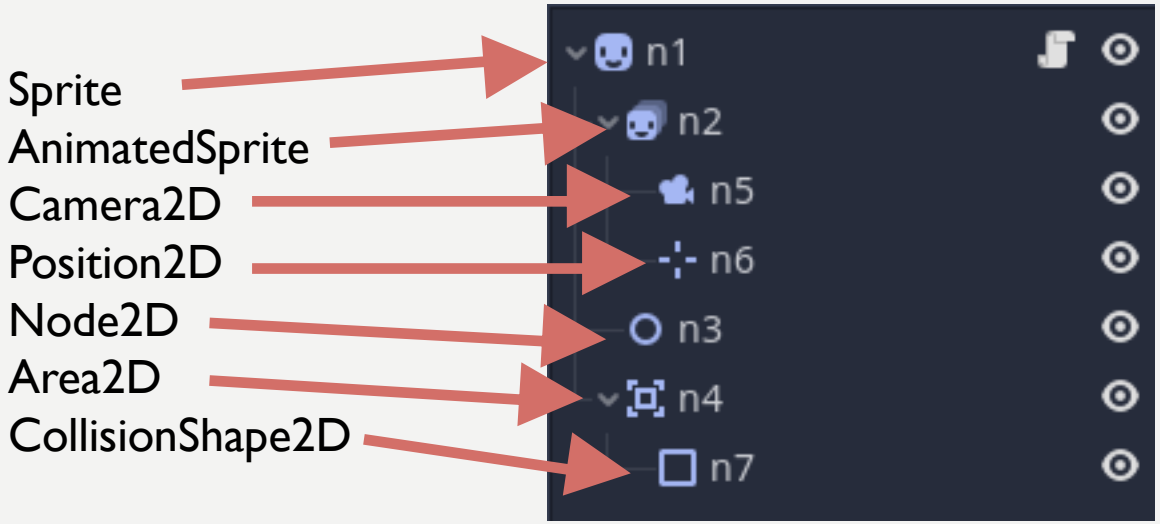


Organisation de la scène en arbre de Godot.

SCÈNE

Quezaco?

Evidemment ici la scène n'est composé que de nœuds de type Node2D. Il est tout à fait possible d'avoir d'autres types de nœuds à la place.



NOEUD

GDScript

Lorsqu'un script est attaché à un nœud, celui-ci se comporte comme une extension du nœud, où l'on peut écrire du code à la main pour manipuler les méthodes (ses prorpres fonctions) et les attributs (ses propres variables) du nœud en question. Dans notre exemple, on attache un script au nœud n1 pour accéder via le code à tout ce qu'il peut nous offrir en tant que Node2D. On s'intéresse ici à comment accéder aux nœuds enfants depuis le nœud n1. Il existe différentes manières d'accéder à un nœud enfant depuis le code GDScript.

On attache un script au nœud racine qui est n1

- Utiliser la méthode self.get_children() qui va renvoyer un tableau contenant tout les enfants du nœud sur lequel est notre script. Il suffira ensuite d'accéder au nœud enfant que l'on veut avec le bon indice. L'enfant le plus haut dans la liste commence par l'indice 0 soit n2.

```
10  func _ready():
11      var children = self.get_children()
12      #on veut avoir le noeud n3 depuis le noeud n1
13      var n3 = children[1] # le second noeud enfant de n1
```

NOEUD

GDScript

Lorsqu'un script est attaché à un nœud, celui-ci se comporte comme une extension du nœud, où l'on peut écrire du code à la main pour manipuler les méthodes et les attributs du nœud en question. Dans notre exemple, on attache un script au nœud n1 pour accéder via le code à tout ce qu'il peut nous offrir en tant que Node2D. On s'intéresse ici à comment accéder aux nœuds enfants depuis le nœud n1. Il existe différentes manières d'accéder à un nœud enfant depuis le code GDScript.

On attache un script au nœud racine qui est n1

- Utiliser la méthode du chemin relatif depuis le nœud auquel est attaché le script grâce à la méthode get_node(chemin_vers_un_noeud_enfant)

```
10 func _ready():
11     # accès au noeud n3 depuis n1
12     var n3 = get_node("n3")
13     # accès au noeud n7 depuis n1
14     var n7 = get_node("n4/n7")
```

NOEUD

GDScript

Lorsqu'un script est attaché à un nœud, celui-ci se comporte comme une extension du nœud, où l'on peut écrire du code à la main pour manipuler les méthodes et les attributs du nœud en question. Dans notre exemple, on attache un script au nœud n1 pour accéder via le code à tout ce qu'il peut nous offrir en tant que Node2D. On s'intéresse ici à comment accéder aux nœuds enfants depuis le nœud n1. Il existe différentes manières d'accéder à un nœud enfant depuis le code GDScript.

On attache un script au nœud racine qui est n1

- Utiliser une variante de la méthode précédente avec le caractère $.

```
10 func _ready():
11     # accès au noeud n3 depuis n1
12     var n3 = $n3
13     # accès au noeud n7 depuis n1
14     var n7 = $n4/n7
```

NOEUD

GDScript

En ce qui concerne le nœuds parent, le processus reste relativement similaire. On va juste utiliser d'autres méthodes pour y arriver.

On attache un script au nœud racine qui est n5

- Utiliser la méthode get_parent() pour récupérer le parent direct du noeud

```
5  func _ready():
6      var n2 = get_parent()
```

NOEUD

GDScript

En ce qui concerne le nœuds parent, le processus reste relativement similaire. On va juste utiliser d'autres méthodes pour y arriver.

On attache un script au nœud racine qui est n5

- Utiliser la méthode get_node() pour récupérer le parent direct ou un parent plus haut dans l'arbre

```
5  func _ready():
6      # .. est le chemin relatif accédant au parent du noeud
7      var n2 = get_node("..")
8      var n1 = get_node("../..")
```

GODOT COLLISIONS

COLLISIONS

Intro

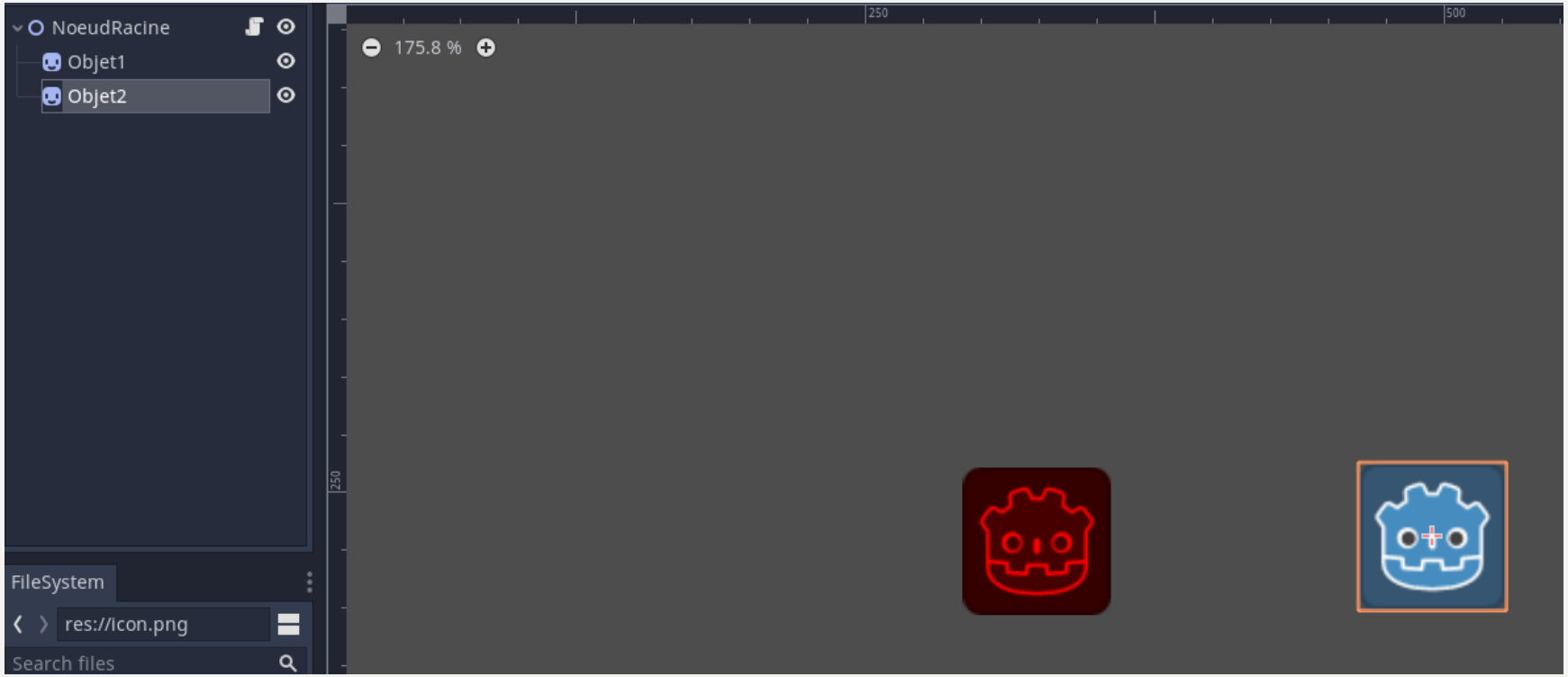
Afin que les objets puissent détecter s'ils se superposent, nous avons besoin d'au moins de type de nœuds spécifiques : Area2D et CollisionShape2D. Pour bien comprendre la différence entre ces deux zones, on peut comparer ce système à celui d'un agent de sécurité qui doit surveille plusieurs zones d'une entreprise depuis ses poste de télé. Si un intrus rentre dans une des zones alors l'agent de sécurité le voit et alerte immédiatement les autorités afin de traiter le problème. Ici l'agent de sécurité correspondrait au nœud Area2D et les zones à surveiller les CollisionShape2D.

COLLISIONS

Setup

Pour mettre en place un système de collision, considérons deux objets dans la scène qui sont représentés grâce à un nœud de type Sprite :

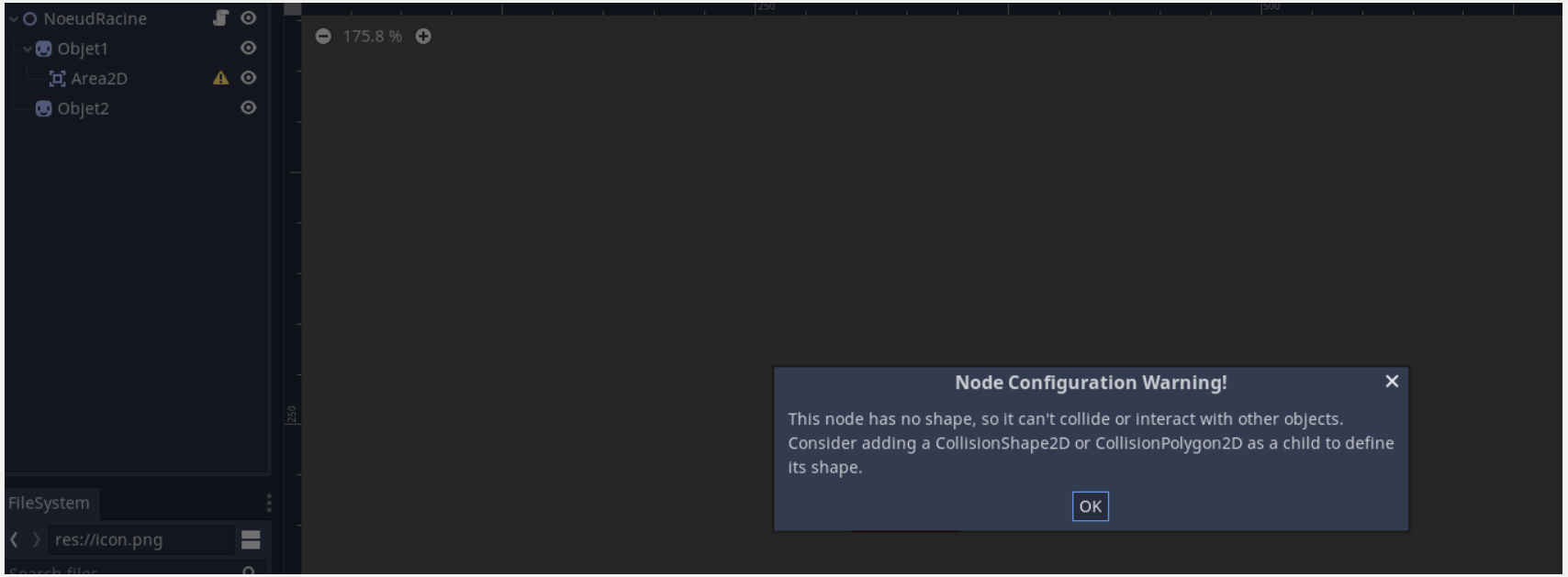
- Objet1
- Objet2



COLLISIONS

Setup

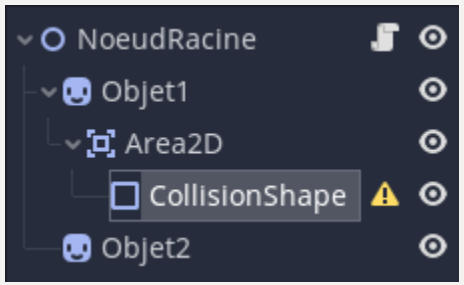
Le but va être de placer une zone de collision pour chacun de ces deux objets afin qu'ils puissent chacun détecter si l'autre lui rentre dedans. Commençons pas le notre ami Godot en rouge qui correspond à Objet1.
La première chose à faire est donc de lui ajouter un agent de sécurité pour détecter les possibles collisions, c'est-à-dire un Area2D.
Ce qui est intéressant c'est qu'un petit icone warning s'affiche à côté du nœud Area2D en alertant qu'il n'a rien à surveiller, c'est-à-dire qu'il n'a aucune zone de collision attachée.



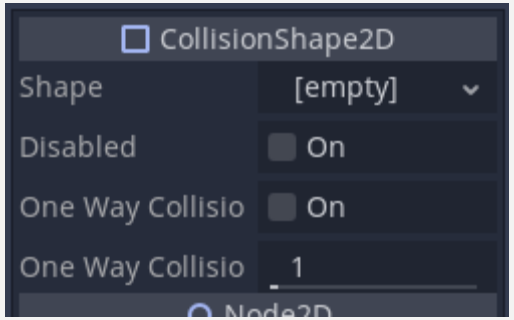
COLLISIONS

Setup

On va donc lui ajouter une zone de collision (CollisionShape2D) afin qu'il ait de quoi surveiller. Ici, nous n'en mettons qu'une, mais il faut garder à l'esprit qu'il est tout à fait possible d'en ajouter plus d'un.
Après avoir ajouter la zone de collision, il faut lui donner une forme dans la scène et ce que l'on veut c'est avoir la même forme que l'image du godot et çà semble bien correspondre à un carré.
Si on regarde dans les propriétés CollisionShape2D on peut voir que la propriété « Shape » est vide pour le moment. Si on clique sur empty, une liste de différentes formes sont proposées comme un rectangle, un cercle, … Créons un rectangle.



Nouveau CollisionShape

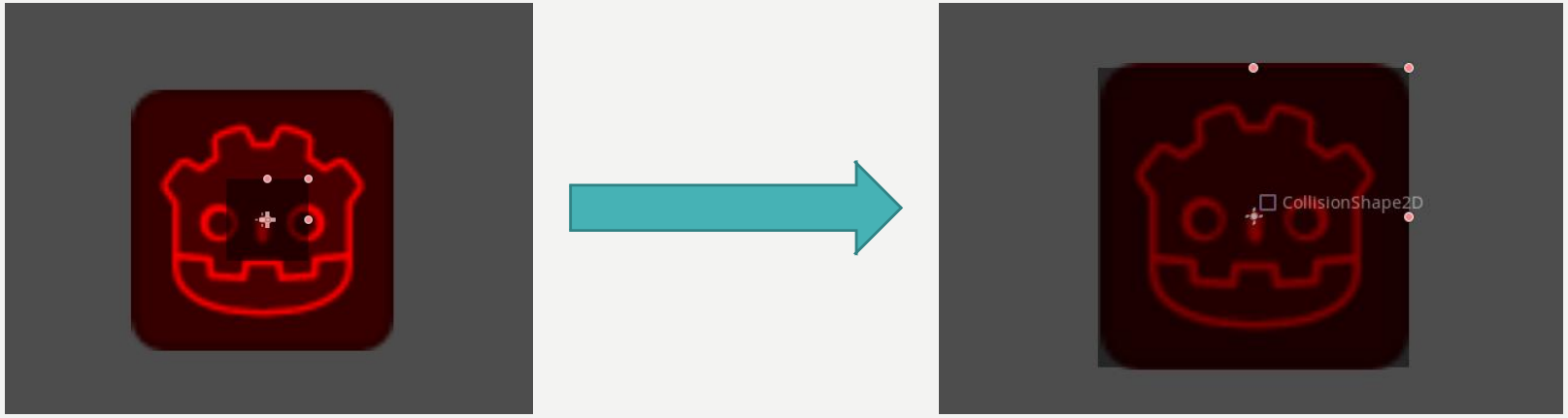


Propriétés de CollisionShape2D

COLLISIONS

Setup

Comme on peut le voir un petit rectangle vient d'apparaître sur notre Godot rouge, mais la taille du rectangle ne semble pas correspondre à la taille de l'image. On va donc tout simplement agrandir la taille du rectangle pour l'adapter.
Maintenant nous avons bien un Area2D et un CollisionShape2D associés à l'Objet1.
Le même processus est fait pour Objet2.



COLLISIONS

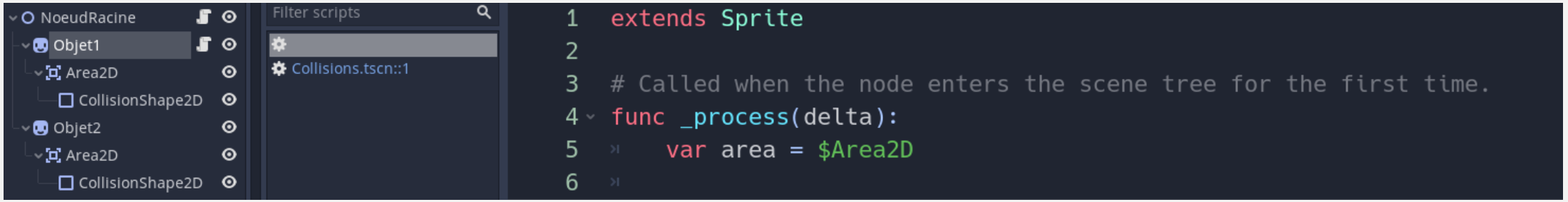
Et dans le code?

Maintenant, intéressons-nous aux méthodes qui vont nous permettre de détecter justement ces autres zones de collisions.
Il faut rappeler que le script qui est attaché sur un nœud ne peut accéder qu'aux attributs et méthodes de ce nœud en question. Pour accéder aux attributs et méthodes d'autres nœuds, deux solutions:

- Attacher un script au nœud sur lequel on veut accéder aux informations
- Récupérer le nœud en question dans une variable (expliqué dans la partie précédente).

On va utiliser la seconde option ici, parce que l'on veut à la fois utiliser les propriétés de Objet1 et de ses enfants dans le même script.

Voilà donc un exemple où on attache un script à Objet1 et on récupère son nœud enfant Area2D dans le script :



COLLISIONS

Et dans le code?

Une fois le nœud stocké dans une variable, on peut donc utiliser les méthodes de ce nœuds et une nous intéresse particulièrement (ne pas hésiter à aller voir la doc!) qui est get_overlapping_areas() et qui nous donne un tableau de toutes les Area2D qui intersecte notre Area2D.
A partir de cela on peut déjà faire des actions en conséquences, comme cacher l'objet intersecté, appliquer des règle physiques pour éviter qu'ils ne se rentrent dedans, informer le jeu qu'il y a eu une intersection, et pleins d'autres actions. C'est à toi de voir ce que tu veux en faire.

```
4  func _process(delta):
5      var area = $Area2D
6      var intersected_areas = area.get_overlapping_areas()
7      # Parcours de chaque area intersecté
8      for intersected_area in intersected_areas:
9          print(intersected_area)
```