1 Définitions et fonctions

1.1 Introduction

Le *Visual Scripting* est présent dans les moteurs de jeu tels que *Unity* et *Unreal Engine*. L'objectif est de permettre la définition de comportement logique uniquement avec des graphes et des représentations visuelles ; classiquement, cela permet plus de collaborations entre programmeurs, artistes et *designer* pour réaliser rapidement des prototypes de jeu.

Le *Visual Scripting* dans un moteur de jeu suppose :

- La définition d'une interface composée de composants graphiques ; dans les sections de cette première partie, ces composants graphiques sont des boutons (appelés *button*) et des libellés (appelés *label*) ; dans les sections des parties suivantes, ces composants graphiques seront composés d'images et de formes géométriques
- La définition d'un graphe définissant le comportement de l'interface
- La définition d'une arborescence de noeuds correspondant aux composants graphiques

D'un point de vue fonctionnel :

- L'interface place visuellement les éléments graphiques des noeuds
- Le graphe définit le corps des fonctions des noeuds associés à un *visual script*
- L'arborescence définit la hiérarchie entre les noeuds

Dans le graphe d'un *visual script* comme présenté en Fig. 1, on a :

- Des liens blancs pour l'exécution du programme
  [Manipuler les liens blancs en les prenant par leur début]
- Des liens bleus clairs pour le passage de paramètres
  [Manipuler les liens bleus clairs en les prenant par leur fin]
- Des rectangles blancs pour les définitions des fonctions
- Des rectangles verts pour les accesseurs *Get-Set* de variables
- Des rectangles violets pour les opérations logiques
- Des rectangles rouges pour les appels de fonction
  [En incluant les accesseurs *Get-Set* associés aux propriétés des composants]
- Des rectangles roses pour les constantes (enfin il n'y a pas de constante ici)
  [Quand un rectangle est sélectionné, il apparaît en bleu foncé]
- Des types utilisés en bleu clair (ici on a des *int* et des *string* abrégés *Str*)

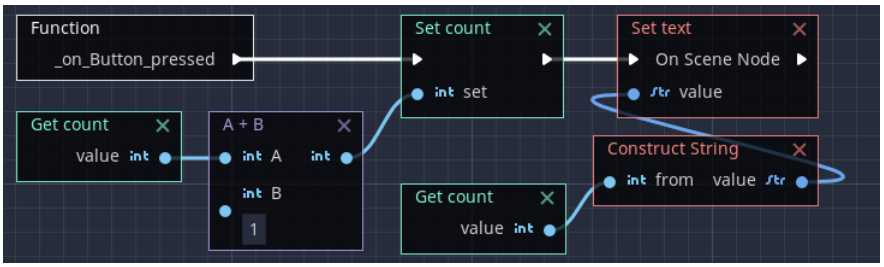

Fig. 1 – Exemple de *visual script*.

1.2 Incrémenter un *label* avec un opérateur *Add*

Quand on clique sur un *button*, la valeur d'un *label* est incrémentée par l'intermédiaire d'une variable `count`.

Fig. 2 et 3 présentent l'arborescence, les fonctions et les variables de la scène.

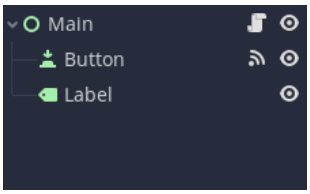

Fig. 2 – Arborescence.

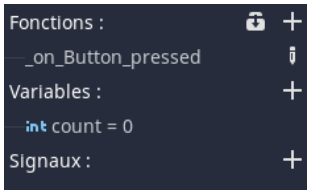

Fig. 3 – Fonctions et variables.

Pour définir l'interaction entre les noeuds `Button` et `Label` de Fig. 2, suivre la procédure ci-dessous pour obtenir le graphe présenté en Fig. 4 :

- Sélectionner l'interface utilisateur (representée par un rond vert), renommer le noeud `Control` par `Main` et l'enregistrer dans `Main.tscn`
- Ajouter un *button* et un *label* (en cliquant sur +)
  [Placer et dimensionner le *button* et le *label* dans l'interface]
- Ecrire *count click* sur le *button* et `0` dans le champ *Text* de *label*
- Attacher un *visual script* `Main.vs` (par un click droit sur `main`)
  [La suite se déroule dans le *visual script*]
- Définir la fonction *_on_Button_pressed* en ajoutant un signal au `Button`
  [Sélectionner `Button`, double-cliquer sur *pressed* dans l'inspecteur]
  [La fonction *_on_Button_pressed* est créée]
  [Cette fonction apparaît dans le *visual script* sous forme d'un rectangle blanc et est ajoutée dans la liste des fonctions comme présenté en Fig. 3]
- Définir une variable `count` de type *int* et *exportable*
  [Glisser-déplacer permet d'obtenir un noeud *Get*]
  [Glisser-déplacer avec Ctrl permet d'obtenir un noeud *Set*]
- Ajouter un opérateur *Add* de Math (par un click droit dans le *visual script*)
- Relier les noeuds *Get*, *Set* et *Add*
- Sélectionner `Label` et utiliser l'inspecteur pour ajouter un noeud *Set text*
- Ajouter un noeud *Set* pour redéfinir le texte de `Label`
- Un noeud de conversion d'*int* en *string* est automatiquement créé
- Exécuter (appuyer sur *play* et définir `Main.tscn` comme scène par défaut)

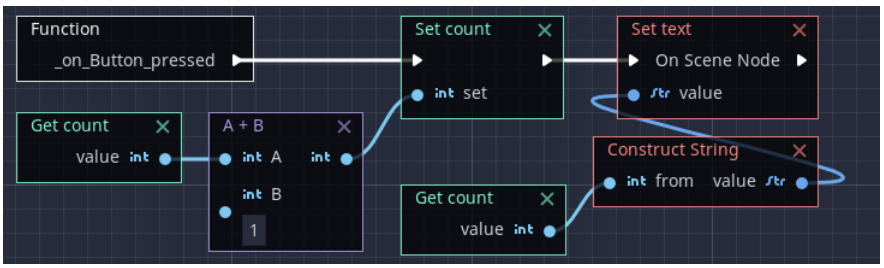

Fig. 4 – Incrémenter un *label* avec un *button* et une fonction *Add*.

1.3 Incrémenter un *label* avec une fonction prédéfinie

Quand on clique sur un *button*, la valeur d'un *label* est incrémentée avec une fonction prédéfinie *Add count*.

L'arborescence, les fonctions et les variables sont identiques à la section 1.2.

Pour définir l'interaction entre le *button* et le *label*, suivre la procédure ci-dessous pour obtenir le graphe présenté en Fig. 5 :

- Ajouter un *button* et un *label* (en cliquant sur +)
- Ecrire *count click* sur `Button` et `0` dans le champ *Text* de `Label`
- Attacher un *visual script* `Main.vs`
- Définir la fonction *_on_Button_pressed*
- Définir une variable `count` de type *int* et *exportable*
  [Etre *exportable* permet de modifier sa valeur via l'inspecteur]
  [Modifier le nom d'une variable implique de cocher-décocher le champ *exportable* pour mettre à jour le nom de la variable dans l'inspecteur]
  [Sélectionner `Main` et `count`]
  [Glisser-déplacer `count` de l'inspecteur dans le *visual script* permet d'obtenir la fonction *Set count* qu'il est possible de réassigner à la fonction *Add* en modifiant le champ *Assign Op*]
- Ajouter un noeud *Get* de `count`
- Ajouter un noeud *Set text* pour *label*
- Relier les noeuds et exécuter

Fig. 5 – Incrémenter un *label* avec un *button* et une fonction prédéfinie.

Il est important de noter que l'inspecteur permet d'obtenir des fonctions plus paramétrables (telles que la fonction *Add count* dans la procédure ci-dessus).

1.4 Afficher dans un terminal les secondes qui passent

Au fil des secondes, une valeur *float* correspondant aux secondes écoulées est affichée dans un terminal.

Fig. 6 et 7 présentent l'arborescence, les fonctions et les variables de la scène.

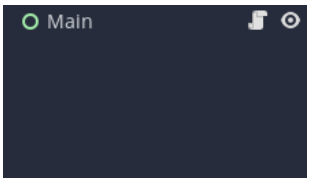

Fig. 6 – Arborescence.

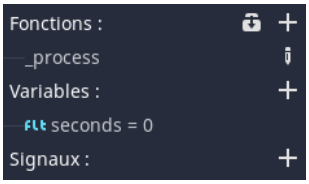

Fig. 7 – Fonctions et variables.

Pour définir un tel processus, suivre la procédure ci-dessous pour obtenir le graphe présenté en Fig. 8 :

- Définir `Main` et attacher un *visual script* `Main.vs`
- Ajouter une fonction prédéfinie de type *Process(float)*
- Définir une variable `seconds` de type *float* et *exportable*
- Ajouter la valeur de *process* à `seconds` avec *Add*
- Ajouter une fonction *print* utilisant un *Get* de `seconds`
- Exécuter (en constatant l'affichage des dixièmes et des centièmes)

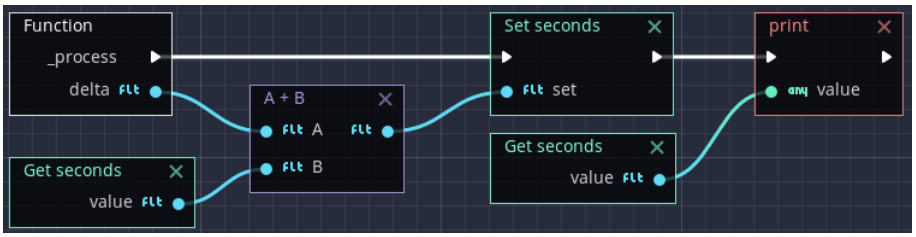

Fig. 8 – Utilisation d'un noeud *Process(float)*.

Pour afficher les secondes sans les dixièmes et les centièmes, on peut utiliser un *Timer* se déclenchant toutes les secondes pour incrémenter une variable `seconds` de type *int* comme présenté en Fig. 9.

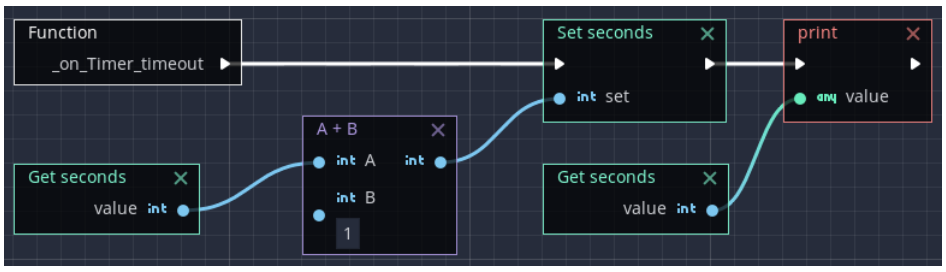

Fig. 9 – Utilisation d'un noeud *Timer*.

Fig. 10 et 11 présentent la nouvelle arborescence, les nouvelles fonctions et les nouvelles variables de la scène.

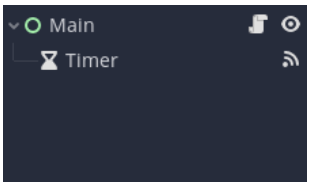

Fig. 10 – Arborescence.

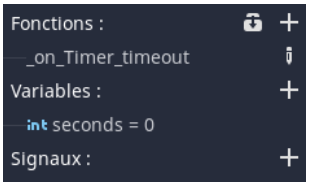

Fig. 11 – Fonctions et variables.

Pour définir ce processus affichant les secondes sans les dixièmes et les centièmes, suivre la procédure ci-dessous correspondant au graphe présenté en Fig. 9 :

- Définir `Main` et attacher un *visual script* `Main.vs`
- Ajouter un noeud de type *Timer* dans `Main`
  [Cocher *Autostart* dans l'inspecteur]
  [Il est possible de modifier sa fréquence avec le champ *Wait Time*]
- Attacher un signal *timeout* de `Timer` vers le script `Main.vs` créant une fonction *_on_Timer_timeout*
- Définir une variable `seconds` de type *int* et *exportable*
- Ajouter les fonctions *print*, *Add*, *Get* et *Set* de `seconds`
- Exécuter (en constatant l'affichage des secondes)

1.5 Cumuler les clicks et les secondes qui passent

La valeur de *label* est incrémentée :
- Au fil des secondes par les signaux d'un *timer*
- A chaque click sur un *button*

Fig. 12 et 13 présentent l'arborescence, les fonctions et les variables de la scène.

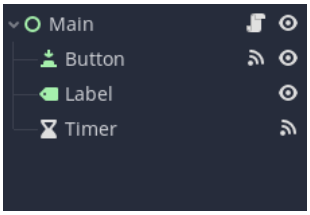

Fig. 12 – Arborescence.

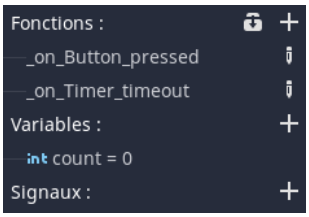

Fig. 13 – Fonctions et variables.

Pour définir un tel processus, réaliser le graphe présenté en Fig. 14 en s'inspirant des procédures des sections 1.3 et 1.4 ; on utilise une variable `count` de type *int* et des fonctions *Add* sur `count`.

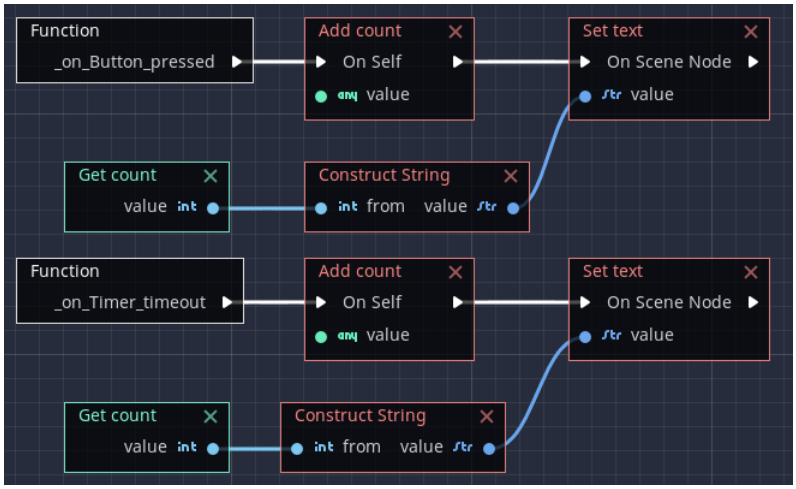

Fig. 14 – Combinaison d'un *timer* et de click d'un *button*.

1.6 Conditionner un click par une valeur minimale

L'interface contient deux *label* et deux *button*; les *label* s'appellent `Label` et `Label2`; les *button* s'appellent `Button` et `Button2`; à chaque click sur `Button`, on incrémente `Label`; cliquer sur `Button2` incrémente la valeur de `Label2` et soustrait 10 à la valeur de `Label`; si la valeur de `Label` est inférieure à 10, cliquer sur `Button2` ne fait rien et les valeurs de `Label` et `Label2` restent inchangées.

Fig. 15 et 16 présentent l'arborescence, les fonctions et les variables de la scène.

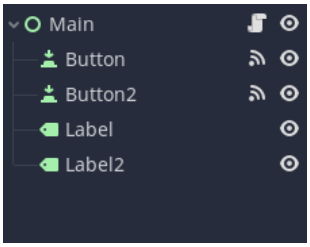

Fig. 15 – Arborescence.

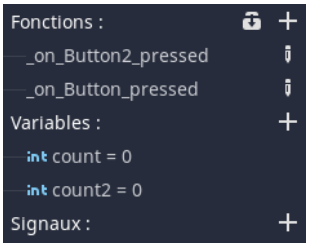

Fig. 16 – Fonctions et variables.

Pour définir un tel processus, suivre la procédure ci-dessous pour obtenir le graphe présenté en Fig. 17 :

- Ajouter deux *button* et deux *label* (en cliquant sur +)
- Les *button* se nomment `Button` et `Button2`
- Les *label* se nomment `Label` et `Label2`
- Ecrire *count click* sur `Button` et `0` dans le champ *Text* de `Label`
- Ecrire *count 10 click* sur `Button2` et `0` dans le champ *Text* de `Label2`
- Attacher un *visual script* `Main.vs`
- Définir les fonctions *_on_Button_pressed* et *_on_Button2_pressed* en utilisant l'inspecteur
- Définir les variables `count` et `count2` de type *int* et *exportable*
- Ajouter un opérateur *Add* et un opérateur *Compare Greater*
- Ajouter des noeuds *Get* et *Set* à `count` et `count2`
- Conditionner la mise à jour de `count2` par un minimum de 10 sur `count`
- Clicker sur `Button2` décrémente `count` de 10 et incrémente `count2` de 1 [Décrémenter est réalisé avec la fonction *Subtract*]
- Ajouter les mise à jour des *label* aux mises à jour des valeurs `count` et `count2`
- Relier les noeuds et exécuter

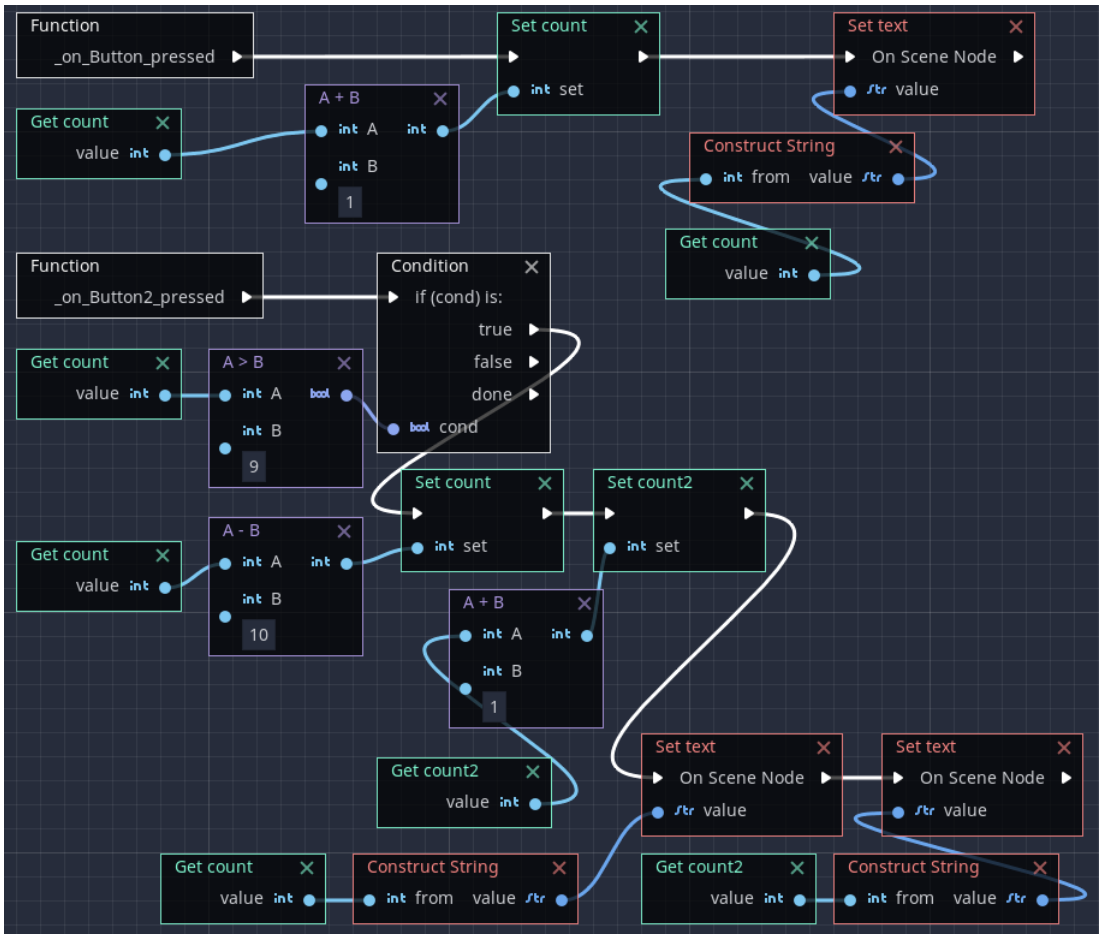

Fig. 17 – Click conditionné par une valeur.

1.7 Activer un *button* selon une valeur minimale

L'interface contient deux *label* et deux *button* avec des conditions identiques à la section précedente; les *button* s'appellent `Button` et `Button2`; quand la variable `count` est inférieure à 10, `Button2` n'est pas cliquable.

Fig. 18 et 19 présentent l'arborescence, les fonctions et les variables de la scène.

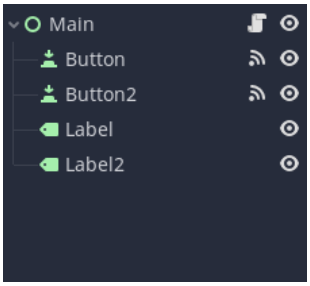

Fig. 18 – Arborescence.

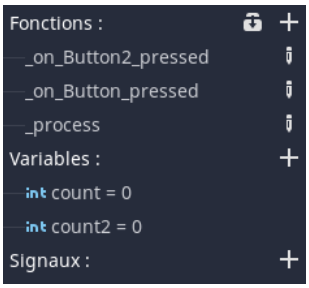

Fig. 19 – Fonctions et variables.

Pour définir un tel processus, suivre la procédure ci-dessous pour obtenir les graphes présentés en Fig. 20, 21 et 22 :

- Ajouter deux *button* et deux *label* (en cliquant sur +)
- Renommer les noeuds `Button`, `Button2`, `Label` et `Label2`
- Ecrire *count click* sur `Button` et 0 dans le champ *Text* de `Label`
- Ecrire *count 10 click* sur `Button2` et 0 dans le champ *Text* de `Label2`
- Attacher un *visual script* `Main.vs`
- Définir les fonctions *_on_Button_pressed* et *_on_Button2_pressed*
- Définir les variables `count` et `count2` de type *int* et *exportable*
- Ajouter des opérateurs *Add* pour ajouter 1 et mettre à jour les *label*
  [On a deux sous-procédures indépendantes pour `Button` et `Button2`]
  [On ajoute une troisième sous-procédure pour activer/désactiver `Button2`]
- Définir `Button2` comme initialement désactivé
- Utiliser une fonction *_process* qui vérifie la condition sur `count` pour activer/désactiver `Button2`]
- Ajouter un opérateur *Compare Greater* et une condition
- Ajouter les fonctions *Set Disable* à *True* et *False* de `Button2`
- Relier les noeuds et exécuter

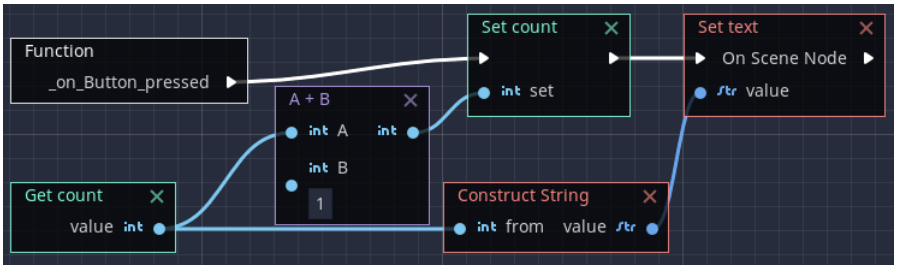

Fig. 20 – Click sur le premier *button*.

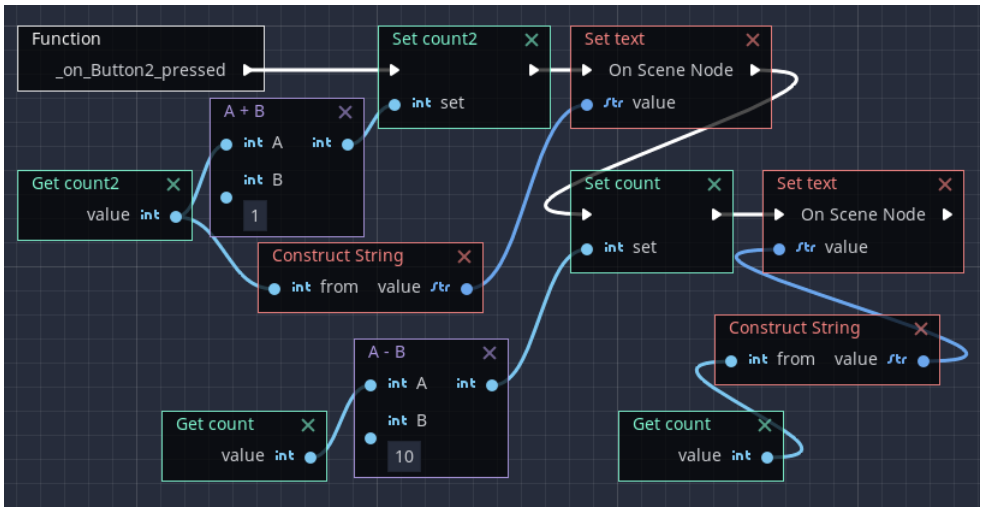

Fig. 21 – Click sur le deuxième *button*.

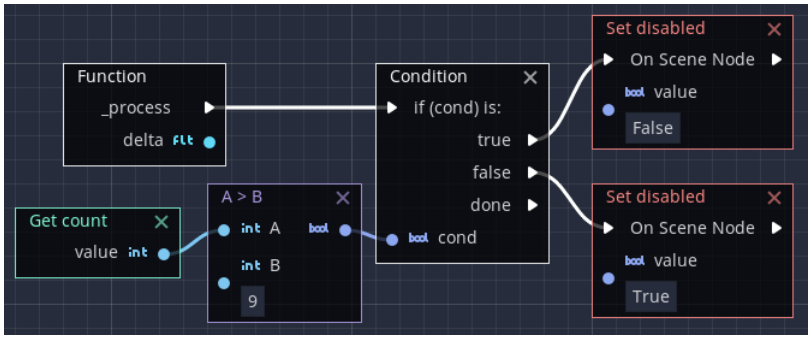

Fig. 22 – Fonction *Process* définissant la relation entre les deux *button*.

Dans certains cas (comme dans Fig. 20 et 21), on peut utiliser plusieurs fois une fonction (respectivement la fonction *Get* de `count` et de `count2`).