Introduction

(NB : J'espère que ce tuto vous sera utile et qu'il sera compréhensible par tous. Si vous avez des questions n'hésitez pas à m'envoyer un message sur Facebook « Antoine Gargasson » ou par mail antoine.gargasson@gmail.com)

(NB2 : Ce tuto est réalisé sur la version 4.8.3 de UE4)

Dans le premier tuto nous avons mis en place un système d'inventaire et une manière de le visualiser.
Dans le second nous avons pu nous déplacer et bouger la caméra. Vous avez également créé des objets que vous pouvez looter.
Dans le tuto d'aujourd'hui nous allons continuer sur ce projet et rajouter le craft d'objet.

Je rappelle l'objectif final du tuto : un système inventaire/craft à la The last of us.

Dans le premier tuto nous avions mis en place le carré de droite avec les items et leur nombre.
Aujourd'hui nous allons rajouter la barre de gauche avec le craft et la sélection de l'objet à crafter.

Avant toute chose il va falloir **réfléchir** à comment nous allons mettre en place le système de craft. Je vous encourage à toujours vous poser **5 minutes** et à réfléchir un minimum à ce que vous allez faire avant de vous lancer à corps perdu dans la bataille. Je ne dis pas que ma manière de faire est la seule et l'unique, elle à l'avantage de marcher, d'être **flexible** et pas trop compliquer à réaliser.

Pour ma part, j'ai d'abord **dessiné et écrit** comment j'allais procéder avant de me lancer dans la réalisation de ce tuto.
Commençons par le **fonctionnel** pour finir par du **visuel** !

Craft

Un craft va être constitué de plusieurs choses. Comme l'Item, le craft va avoir un **ID** pour les différencier.

Si vous souhaitez que votre objet crafté puisse être jeté dans le monde il faudra procéder comme pour l'Item et faire une classe mère et de l'héritage. Ce n'est hélas pas prévu dans ce tuto (peut-être plus tard :P).

On va également lui mettre un tableau d'objets requis. C'est une partie **importante** qui sera à ne pas négliger car c'est le pivot de votre **craft**. Ce tableau sera le tableau de **Requirement** (requis en anglais).
Ce **Requirement** sera une structure (et oui encore!) qui possédera un **ID** qui fera référence aux objets de l'inventaire et une **quantité** dont le joueur aura besoin pour crafter.
Ensuite on laissera au joueur la liberté de se **balader** dans les craft et de pouvoir **effectuer** le craft d'un objet.

C'est d'ailleurs par les **Requirement** que l'on va commencer !

Ouvrez Visual Studio 13 et créez un nouveau fichier.h dans le dossier de votre projet, nommez le "**Data.h**". Pour cela, clique droit sur votre dossier de projet dans les sources et faites **Ajouter->Nouvel élément**.

Dans ce fichier nous allons pouvoir ajouter une structure. Dans ce fichier vous pourrez également rajouter des structures ou autre dont vous aurez besoin. Rajoutez les éléments comme suit :

Data.h

```
#pragma once

#include "TutoInventaireC.h"

USTRUCT()
struct FRequirement
{
	GENERATED_USTRUCT_BODY()

	//mettre vos variables et fonctions ici
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =
"Requirement")
	int32 ID;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =
"Requirement")
	int32 Quantite;

	FRequirement()
	{
		//mettre l'initialisation des variables ici
		ID = -1;
		Quantite = 1;
	}
};
```

Nous pouvons à présent faire le **lien** entre le craft et les Item.

Attention, votre fichier.h ne sera pas créé au bon endroit ! Vous le trouverez dans Intermediate->ProjectFiles. Il faut le déplacer dans Source->NomDeVotreProjet.

Toujours dans le fichier **Data.h**, **créez** une nouvelle **structure** qui servira aux **craft**. Nommez la **Fcraft**. Dans ce craft, rajoutez une ID afin de les différencier, un tableau de Requirement si vous souhaitez avoir plusieurs Item nécessaires aux craft, une quantité initialisée à 0 et un Nom.

Ce qui ressemble à ceci :

Data.h

```
USTRUCT()
struct FCraft
{
	GENERATED_USTRUCT_BODY()

	//mettre vos variables et fonctions ici

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Craft")
	int32 ID;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Craft")
	int32 Quantite;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Craft")
	TArray<FRequirement> Requirement;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Craft")
	FString Nom;

	FCraft()
	{
		//mettre l'initialisation des variables ici
		ID = -1;
		Quantite = 0;
		Nom = "UNKNOWN";
	}
};
```

Une fois cette préparation effectuée, nous pouvons **retourner** sur la classe du joueur **TutoJoueur.**

Pour commencer nous allons ajouter deux variables qui seront très utiles pour le craft :
- Un **tableau de craft** que vous pré-remplirez (je vous donnerai les infos à mettre dedans ne vous inquiétez pas ;)).
- Un **index** qui va servir à la navigation dans l'inventaire de craft.

Commençons par le plus rapide : l'index. Ajoutez une variable **Integer** et nommez là **IndexCraft**.

Ensuite créez une deuxième variable de type **tableau de FCraft** et nommez là **Craft**.
Dans le TutoJoueur.cpp, vous allez ajouter **5 éléments**.

Voici comment va marcher le tableau à **remplir** :

ID Nom Quantité Requirement : ID, quantite

ID : 0 Nom : Kit soin Quantité : 0 Requirement : 0,1 2,1
ID : 1 Nom : Couteau Quantité : 0 Requirement : 4,1 5,1 1,1
ID : 2 Nom : Crochetage Quantité : 0 Requirement : 1,2
ID : 3 Nom : Molotov Quantité : 0 Requirement : 2,1 3,1
ID : 4 Nom : Bandage Quantité : 0 Requirement : 3,2

A partir ce tableau, vous pouvez remplir le tableau craft dans le constructeur de votre classe :)

Tout d'abord je **déclare** un **FCraft** qui va me servir **d'objet** à **remplir** et que je vais **ajouter** ensuite au **tableau**. J'utilise également un **FRequirement** qui va me servir à **remplir** le tableau de **requirement**. Je **vide** le tableau de **requirement** entre chaque car sinon j'aurais à la fin une liste énorme puisque le tout se **cumulerait**.

Il est **important** ici d'utiliser les .**Add**(). J'ai eu le malheur une fois d'utiliser directement les cases du tableau.. J'ai du aller voir dans les logs car UE4 ne voulait même plus s'ouvrir, il crashait à chaque fois.

Une fois vidé, vous pouvez passer au craft suivant et ainsi de suite en vidant le tableau des FRequirement (require) à chaque fois.

TutoJoueur.cpp

```
// Sets default values
ATutoJoueur::ATutoJoueur()
{
	// Set this character to call Tick() every frame.
	PrimaryActorTick.bCanEverTick = true;

	NombreMaxItem = 5;

	FCraft ACraft;
	FRequirement require;

	ACraft.ID = 0;
	ACraft.Quantite = 0;
	ACraft.Nom = "Kit Soin";
	require.ID = 0;
	require.Quantite = 1;
	ACraft.Requirement.Add(require);
	require.ID = 2;
	require.Quantite = 1;
	ACraft.Requirement.Add(require);

	Craft.Add(ACraft);

	ACraft.Requirement.Empty();

...
```

Une fois le tableau rempli, nous allons nous attaquer à la mécanique de craft. Nous allons utiliser les **fonctions** déjà faites et en rajouter d'autres.

Pour commencer, il faut **bloquer** le joueur quand il ouvre son inventaire. En effet la navigation va s'effectuer avec les touches directionnelles haut et bas et le craft va être effectué avec la touche action.

Rien de plus simple quand on a une variable **InventaireVisuel** qui nous indique si l'inventaire est ouvert ou non. **On vérifie** juste avant le déplacement et on vérifie **l'état** d'ouverture de l'inventaire.

Rajoutez simplement la condition **avant** le déplacement (en rouge) :

TutoJoueur.cpp

```
//Move the player left and right
void ATutoJoueur::Droite(float value)
{
	if ((Controller != NULL) && (value != 0.0f))
	{
		if(!InventaireVisuel)
		{
			//Find out which way is forward
			FRotator Rotation = Controller->GetControlRotation();
			//add movement in that direction
			const FVector Direction =
FRotationMatrix(Rotation).GetScaledAxis(EAxis::Y);
			AddMovementInput(Direction, value);
		}
	}
}
```

Faites de même pour votre déplacement en avant.

Ici on test juste si l'inventaire est ouvert ou fermé pour décider de l'action à effectuer.

On compile ! *(J'ai eu un crash mais pas de soucis ! C'est l'avantage de travailler en C++, votre VS2013 ne se ferme pas lui!)*

Si vous testez vous allez remarquer que vous ne bouger plus quand votre inventaire est ouvert mais que la sourie peut encore tourner.

Il est temps de rajouter deux Mapping d'action pour changer l'index du craft. On ne vas pas utiliser les axes car ils sont testés toutes les frames ! :)

Action (MonterCraft, Z)
Action (BaisserCraft, S)

De retour dans **TutoJoueur**, voici comment nous allons procéder pour **changer d'index** de craft :

Commencez par **ajouter** une **fonction** nommée **MonteCraft** de type **void** et une autre **BasseCraft**.

TutoJoueur.h

```
// Go UP in the crafting list
UFUNCTION()
void MonteCraft();

// Go DOWN in the crafting list
UFUNCTION()
void BaisseCraft();
```

N'oubliez pas de **binder** les **Inputs** aux nouvelles **fonctions** !

TutoJoueur.cpp

```
InputComponent->BindAction("MonterCraft", IE_Pressed, this,
&ATutoJoueur::MonteCraft);
InputComponent->BindAction("BaisserCraft", IE_Pressed, this,
&ATutoJoueur::BaisseCraft);
```

Ensuite il nous reste à **écrire** les **fonctions** qui **changeront** simplement **l'index** :

```
// Go UP in the crafting list
void ATutoJoueur::MonteCraft()
{
	if (InventaireVisuel && IndexCraft - 1 >= 0)
	{
		IndexCraft--;
	}
}

// Go DOWN in the crafting list
void ATutoJoueur::BaisseCraft()
{
	if (InventaireVisuel && IndexCraft + 1 < Craft.Num())
	{
		IndexCraft++;
	}
}
```

Quand nous voulons **monter visuellement** il faut **baisser l'index**, on vérifie qu'elle ne sera pas inférieur à 0 et on la baisse.
Quand nous voulons **baisser visuellement** il faut **monter l'index**, on vérifie qu'elle ne dépassera pas le tableau et on la monte.

J'ai oublié de vous parler des **commentaires** !
C'est une base à avoir pour ne pas vous perdre dans votre code et **retrouver** rapidement du code que vous cherchez.

Pour ajouter un **commentaire**, il suffit simplement de mettre

//écrivez ici.

Pour faire un pavé mettez :

/*
écrivez ici
*/

Maintenant que notre code est commenté, nous allons pouvoir ajouter une **fonction** qui va nous renvoyer **vrai ou faux** après avoir testé si nous avons tous les objets **requis** pour lancer le craft.

Dans le TutoJoueur.h, créez une nouvelle fonction et nommez là **GetEnoughItem**. Elle prendra en entrée un **Index** qui sera celui du craft à effectuer et en sortie un **booléen** qui sera le résultat de la requête.

TutoJoueur.h

```
// Check if we have all the Item to craft the selected Item
UFUNCTION()
bool GetEnoughItem(int32 TheIndex) ;
```

Créez une **variable locale** booléenne initialisez-la à **true** et nommez là **RetourRequete**.

Voici comment je l'ai créé :

TutoJoueur.cpp

```
// Check if we have all the Item to craft the selected Item
bool ATutoJoueur::GetEnoughItem(int32 TheIndex)
{
	bool RetourRequete = true;
	FCraft TheCraft = Craft[TheIndex];

	for (int32 i = 0; i < TheCraft.Requirement.Num(); i++)
	{
		int32 ReturnNumber =
GetNumberFromID(TheCraft.Requirement[i].ID);
		if (ReturnNumber < TheCraft.Requirement[i].Quantite)
		{
			RetourRequete = false;
			return RetourRequete;
		}
	}
	return RetourRequete;
}
```

On récupère le craft demandé en paramètre. Pour chacun de ses requirement on vérifie si on a assez de cet Item en utilisant la fonction GetNumberFromID. Si on en a pas assez, on passe à false et on renvoi directement false car on a pas besoins de vérifier les autres puisque c'est déjà false. A la fin on renvoi le résultat qui doit être true.

Retrouvez la fonction **OnUse** qui est effectué quand vous appuyez sur E. Si vous avez bien commenté, ça devrait être **rapide** à retrouver ;)

Rajoutez un test au début pour voir si on est dans l'inventaire. S'il n'est pas ouvert mettez ce que vous aviez déjà et s'il est ouvert, Utilisez la fonction GetEnoughItem que vous venez de créer. Affichez un message à l'écran pour vérifier qu'elle fonctionne.

```
// handle use
void ATutoJoueur::OnUse()
{
	if (InventaireVisuel)
	{
		bool RetourFonction = GetEnoughItem(IndexCraft);
		if (GEngine)
		{
			GEngine->AddOnScreenDebugMessage(-1, 5.f,
FColor::Yellow, RetourFonction ? TEXT("true") : TEXT("false"));
		}

	}
	else
	{
			…
	 }
}
```

Ici on test si on a pas ouvert l'inventaire, alors on check pour le ramassage d'un item, sinon on check pour un craft et on imprime à l'écran le résultat. Normalement ça devrait être false puisque on a aucun item à la base.

Lorsque nous avons créé l'action qui rajoute un **item** quand on appuie sur une certaine touche, c'était **P** et ça s'appelle **AjoutObjet**. Nous avions utilisé **AddItemWithId**. Retrouvez cette fonction et changez l'ID pour le AddItemWithID par l'**ID 3** et dupliquez la. Si vous appuyez sur P vous aurez alors 2 tissus.

```
// TEST : Test the adding of an object
void ATutoJoueur::AjoutObjet()
{
	AddItemWithID(3);
	AddItemWithID(3);
}
```

Maintenant **lancez le jeu**, faites **P, ouvrez** l'inventaire, appuyez 3 ou 4 fois sur **S** pour arriver en bas de la liste de craft et appuyez sur **E**. Vous voyez **True** à l'écran ? C'est que votre craft **marche** :D

Il reste à augmenter de 1 la quantité du craft et à réduire d'autant que nécessaire les Items pour le craft. Pour cela devinez quoi ? On va faire une **fonction** ;)

Elle va s'appeler **CraftItem**, va prendre un **Index** en entrée et n'aura **pas** de sortie.
Nous allons dans un premier temps **augmenter** la **quantité** du craft en question puis dans un second temps nous allons, et de manière automatique, **réduire** tous les **Items** concernés.

J'ai dupliqué ma fonction **RemoveItemWithID** que j'ai appellé **RemoveItemWithIDAndNumber** et qui sera plus pratique pour cette partie.

```
// Remove an Item in the inventory with the specific ID and Number
UFUNCTION(BlueprintCallable, Category = Gameplay)
void RemoveItemWithIDAndNumber(int32 TheID, int32 TheNumber);
```

Voici ma fonction :

TutoJoueur.cpp

```
// Remove an Item in the inventory with the specific ID and Number
void ATutoJoueur::RemoveItemWithIDAndNumber(int32 TheID, int32
TheNumber)
{
	int32 Num = GetNumberFromID(TheID);
	if (Num - TheNumber > 0)
	{
		int32 Index = GetItemIndexWithID(TheID);
		Inventaire[Index].Quantite-=TheNumber;
	}
	else
	{
		int32 Index = GetItemIndexWithID(TheID);
		Inventaire[Index].Quantite = 0;
	}
}
```

C'est exactement la même fonction, sauf que je **décrémente** de la valeur choisie et si je suis en dessous de 0, je met la quantité à 0.

Et enfin voici ma fonction **CraftItem** finie !

TutoJoueur.cpp

```
void ATutoJoueur::CraftItem(int32 TheIndex)
{
	Craft[TheIndex].Quantite++;
	for (int32 i = 0; i < Craft[TheIndex].Requirement.Num(); i++)
	{
		RemoveItemWithIDAndNumber(Craft[TheIndex].Requirement[i].ID,
Craft[TheIndex].Requirement[i].Quantite);
	}
}
```

Il ne reste plus qu'à mettre le **craftItem** après avoir tester si on a assez d'Item pour le craft et à l'exécuter sur L'IndexCraft.

TutoJoueur.cpp

```
// handle use
void ATutoJoueur::OnUse()
{
	if (InventaireVisuel)
	{
		bool RetourFonction = GetEnoughItem(IndexCraft);
		if (GEngine)
		{
			GEngine->AddOnScreenDebugMessage(-1, 5.f,
FColor::Yellow, RetourFonction ? TEXT("true") : TEXT("false"));
		}
		if(RetourFonction)
		{
			CraftItem(IndexCraft);
		}
	}
	else
	{
...
```

Si comme moi la compilation se passe bien mais que au moment de lancer vous avez un message qui s'affiche vous disant qu'il y a des erreurs dans les blueprints c'est que le HotReload à foiré. Pas d'inquiétude ! Fermer votre UE4 et relancez le ! Le soucis sera réglé ;)

Lancer le jeu et comme tout à l'heure, faites **P** et descendez avec **S** une fois l'inventaire **ouvert**. Vous devriez voir un **true** et si vous appuyez une **deuxième** fois un **false**. Si c'est le cas votre craft marche et tout se passe pour le mieux.

Nous allons maintenant partir dans la partie bonus avec que du visuel. Ça va être un peu plus compliqué mais ça permettra de voir sur quel craft nous sommes et de voir quels Items sont nécessaires à la fabrication du craft.

Visuel

Abordons maintenant la partie visuel !

Pour commencer, récupérez des **images** en **64x64** sur **google**. Il vous faut **5 images**, une image de bandage, une de couteau, une de crochet, une de molotov et une de tissu.

Il se peut que votre dossier est été supprimé si vous avez fermé votre projet sans rien mettre dans le dossier Images. Ce n'est pas grave, recréé le .

Allez dans le dossier **Images** et faites **Import**. Choisissez vos images et validez.

Vous allez pouvoir créer à présent un nouveau **Widget** dans le dossier **UMG** et vous le nommerez **UMG_CraftingList**.

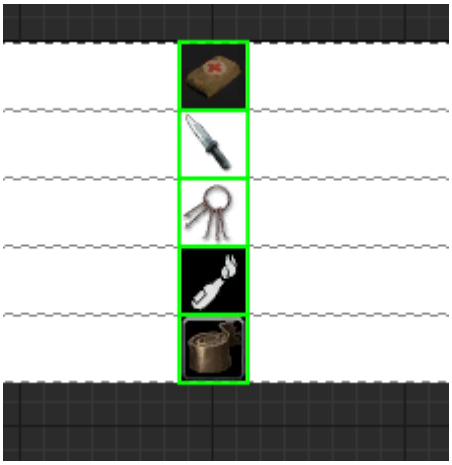
Supprimez le **Canvas Panel**. **Ajoutez** un **vertical Box**. **Dedans** mettez-y **5 Bordures** qui correspondront aux 5 craft et nommez les **Selection1**, **Selection2**, etc. Dans **chaque** bordure mettez une **image**. Et cliquez sur **Is variable** pour chaque bordure.

Une fois tous ces éléments ajoutés, mettez vos **images** que vous avez précédemment récupérées. Dans l'ordre : Kit soin, Couteau, Crochetage, Molotov, Bandage

Pour attribuer une image, **sélectionnez** l'image dans la **hierarchy** et trouvez **Brush**. Déroulez l'onglet vous pourrez attribuer l'image.

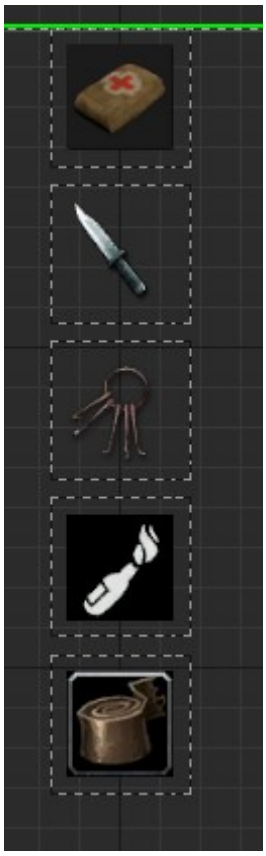
Et là vous regardez le résultat et… y a u problème de taille.. on va résoudre ça !

Sélectionnez toutes vos **images** et cliquez sur **Horizontal Alignment** au **centre**.



Ok vos images sont centrées mais y a des grandes bandes blanche.. Ce sont les bordures.. on va ajouter un peu de **padding** pour espacer les craft et ajouter une bordure tout **autour**.

Sélectionnez toutes les **bordures**. Mettez le **Padding Content** à **10** et changez le **Brush Color** par un blanc avec un **alpha** à **0**. Rajoutez un **Padding Layout** à **10** sur le **bottom uniquement** en déroulant le Padding.



Ça ressemble déjà plus à ce que l'on souhaite !

Sélectionnez toutes vos **bordures** et cliquez sur **Horizontal Alignment** au **centre** sur le **layout**.

On a donc mis en transparent les bordures pour pas gêné la vue dans l'éditeur. Le graph se chargera de gérer la couleur. On a mis une bordure à 10 pour avoir la couleur du craft. Blanc (Ok), Rouge (pas bon).

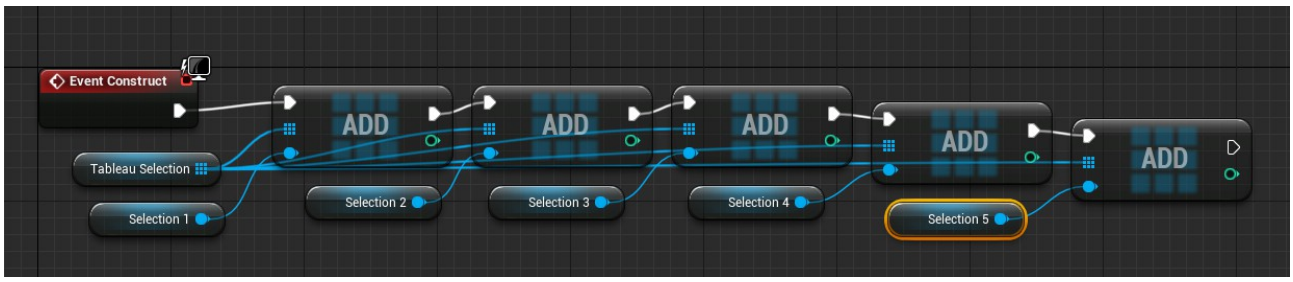
Passons au Graph !

On va commencer par ajouter une variable **Integer** nommée **Selected** et on l'initialise à **0**.
Cette variable va servir à la navigation dans la ligne de craft.

On créé un tableau de **Border** qu'on va appeler **TableauSelection**. Ce tableau, on va le **manipuler** pour effectuer des **traitements** sur la **couleur** des **bordures**.

On va également créer une variable **Joueur** de **type BP_TutoJoueur** qui va servir à avoir une **référence** sur le joueur.

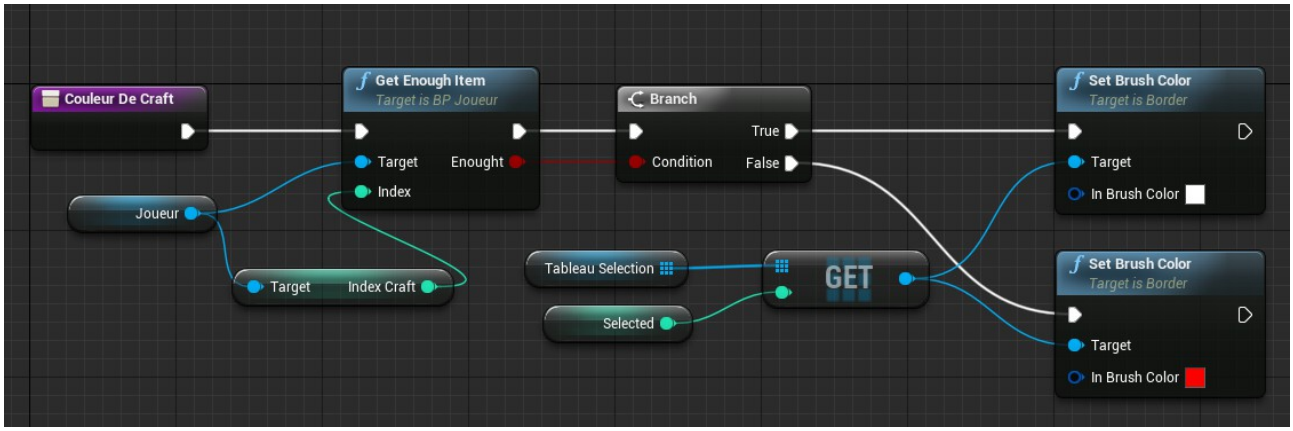
Dans le graph, On part de l'event construct et on rempli le tableau :



On va créer maintenant 3 **fonctions**. La première va servir à déterminer si l'objet est **craftable** ou non. La seconde va servir à passer au craft **suivant**. La dernière va servir à revenir au craft **précédent**.

Commençons par la vérification de la craftabilité de l'objet (j'aime inventer des mots).

Créez une fonction **CouleurDeCraft** avec **aucune** entrée et **aucune** sortie.



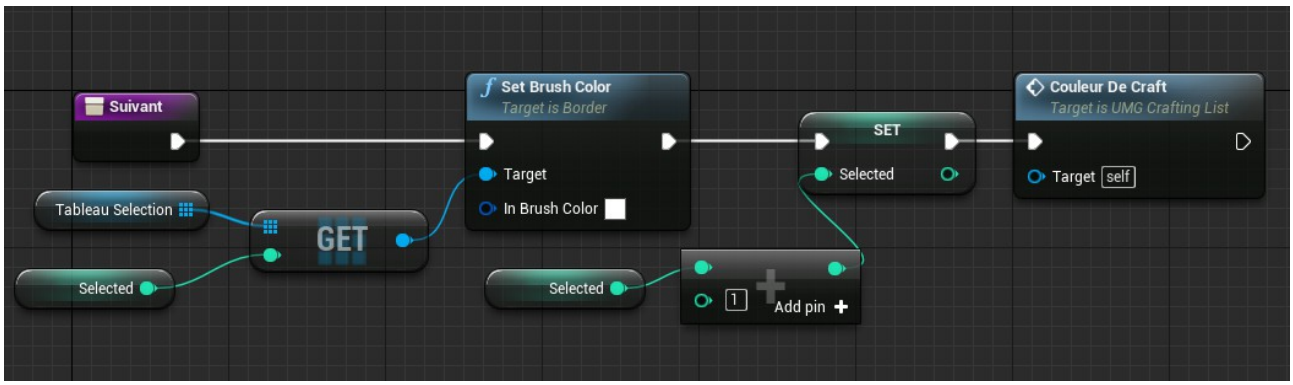
Voici ma fonction **CouleurDeCraft**. On voit si il y a assez d'items pour **crafter** et on **ajuste** la couleur suivant le résultat : Blanc si c'est **ok** et rouge si c'est **pas bon**.

Si vous n'arrivez **pas** à avoir la fonction **GetEnoughItem** c'est qu'elle n'est pas appelable depuis blueprint. Il faut y remédier en la modifiant dans **TutoJoueur.h** .

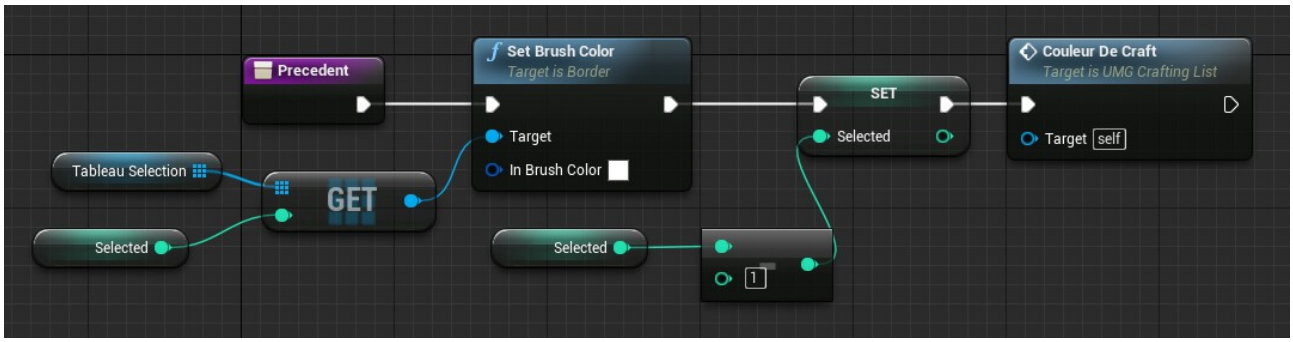
TutoJoueur.h

```
// Check if we have all the Item to craft the selected Item
UFUNCTION(BlueprintCallable, Category = Gameplay)
bool GetEnoughItem(int32 TheIndex);
```

Ensuite on va créer la fonction **Suivant** et **Precedent**. Elles seront très **similaires**.

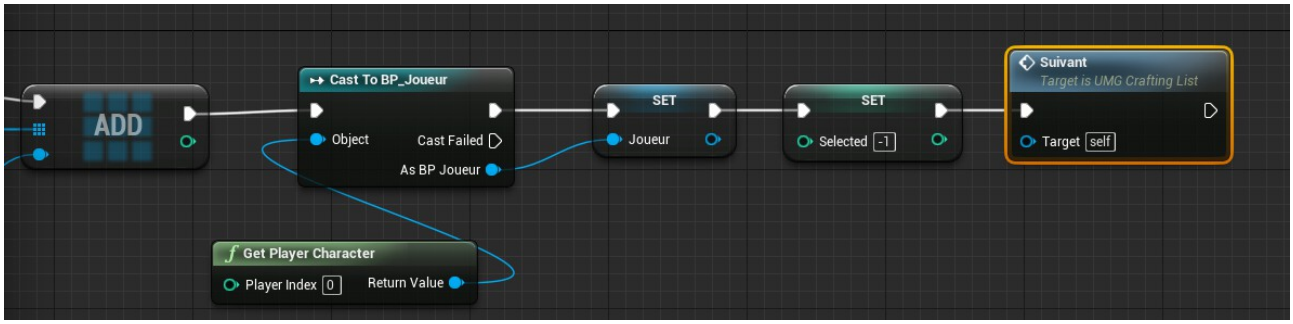


On passe en **transparent** la couleur du brush (Blanc avec **alpha** à **0**), on **incrémente** et on regarde la **couleur** du nouveau craft.



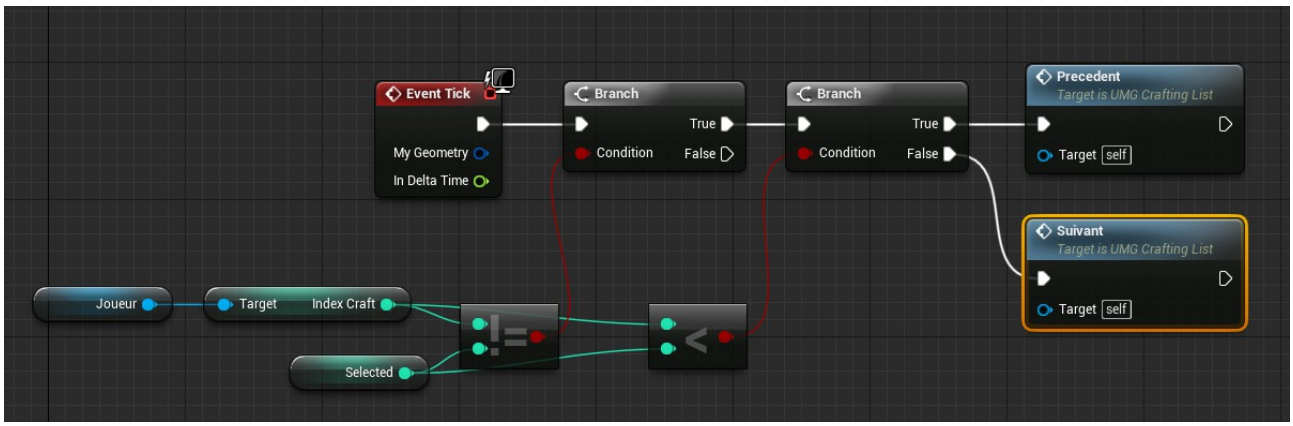
Un simple **copié collé** et j'ai remplacé le + par un **-** .

De retour sur l'**event graph**, on fini le **construction event** en ajoutant les fonctions que l'on vient de créer à la suite des **Add**.



Il faut maintenant savoir **quand** on doit changer de craft et **actualiser** le tout.

On va utiliser l'**event tick** qui est exécuté toutes les **frames**, soit **60 fois par secondes** sur une machine normale, attention à son utilisation donc.



On vérifie si l'index a changé et on vérifie si on incrémente ou décrémente. Ensuite on exécute la fonction adaptée.

On compile et on sauvegarde tout si ce n'est pas déjà fait !

Pour finir on se rend dans **UMG_HUD**. Dans la **palette**, naviguer **tout en bas** et **déroulez User Created**. Vous devriez voir **UMG Crafting List**. Faites un **glissé déposé** à coté de l'inventaire.

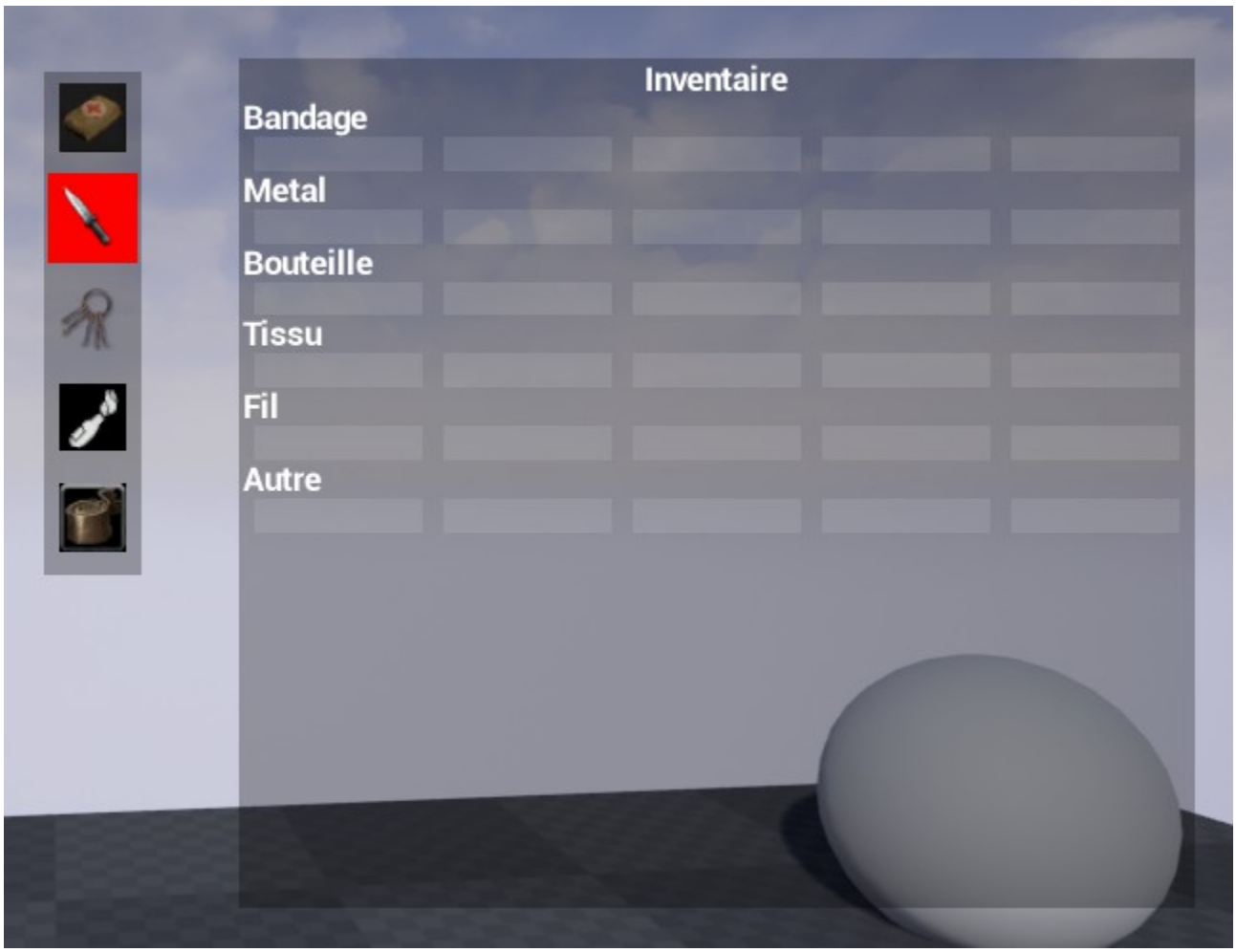
Compilez et sauvegardez !

Il ne reste plus qu'à le cacher quand il n'est pas voulu.

Quand votre **CraftingList** est sélectionnée dans **UMH_HUD**, déroulez le **Is Enabled** et vous devriez **retrouvez** le **bind** que vous avez fait pour l'inventaire. Faites de même avec **Visibility**.

Sauvegardez et testez.

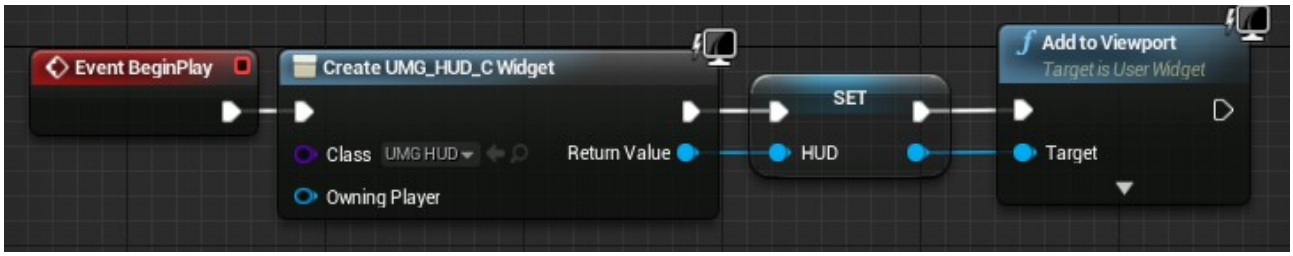
Quand vous ouvrez votre inventaire, vous devriez voir le tout et vous pouvez naviguer entre les craft. Ajoutez avec P du tissu, naviguez jusqu'au bandage, c'est le dernier, il devrait être blanc.
Si vous étiez déjà dessus ça n'a pas du changer de couleur, c'est normal, c'est un test et il n'actualise pas la couleur en direct, monté et redescendez et ça ira.



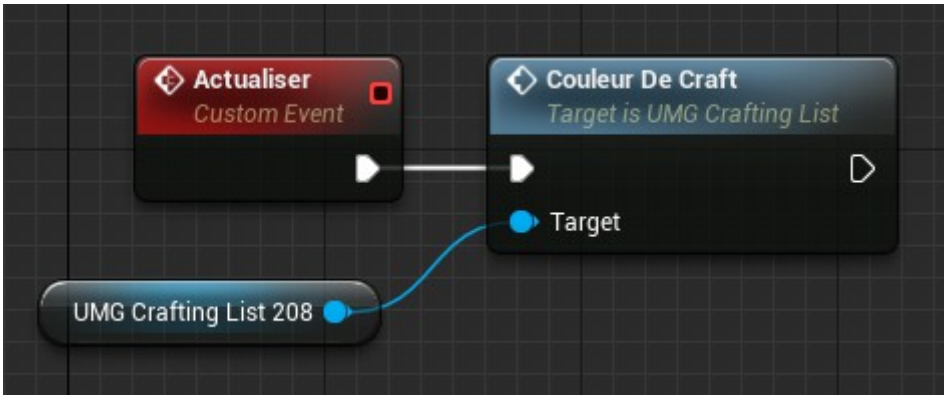
Je vous laisse le soin de **rajouter** une **bordure** dans **UMG_HUD** comme pour l'inventaire. **Cochez Size To Content** et n'oubliez pas de mettre les **bind** sur la **bordure** et non sur la liste de **craft**.

En revanche, quand on effectue le craft, la couleur ne change pas et là c'est un peu plus embêtant…

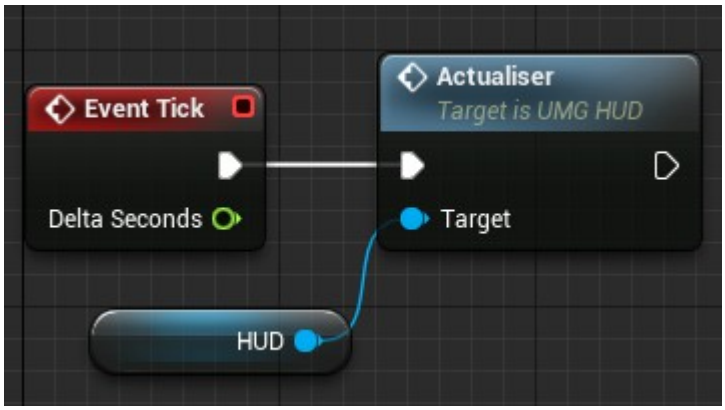
Je vous propose tout simplement pour régler ce petit soucis **d'exécuter** la fonction **CouleurDeCraft** toutes les frames. Pour cela rendez vous dans votre **blueprint** de joueur. **Créez** une variable **HUD** de type **UMG_HUD** et affectez là quand vous créez votre HUD.



Sur votre **UMG_HUD**, créez un **custom event**, nommez le **Actualiser**. Cet event **appellera** la fonction **CouleurDeCraft** sur la **crafting list** que vous avez ajoutez précédemment.



De retour sur le BP_TutoJoueur, localisez l'event tick. Il ne vous reste plus qu'à appeler l'événement Actualiser sur votre HUD.



Cette dernière partie était pas facile du fait qu'il faille jongler avec le HUD et le joueur. C'était du bonus et félicitation si vous êtes arrivez jusqu'au bout !

Vous voilà désormais avec un inventaire, un système de craft, vous pouvez ramasser des objets, les crafter, voir les objets en inventaire, etc.

Vous pouvez être fier de vous, vous avez été de bout en bout de ce tuto pas simple à tout les endroits. Vous **connaissez** maintenant quelques ficelles qui ne demandent qu'à être **utilisées** et **manipulées** au cours de **projets** et **d'expérimentations**.

Vous avez pu voir comment faire de l'UMG, comment faire le lien UMG/blueprint, comment faire un blueprint, comment faire des fonctions, des commentaires, des variables, comment commenter votre code, le rendre maintenable et lisible et surtout vous avez fournis un code flexible et adaptable à toutes circonstances !

Félicitation à vous !

Vous pouvez commenter, aimer, partager ce contenu. Tester des choses et vous amuser avec !

Antoine Gargasson Septembre 2015

Pour toute éventuelles questions, remarques, détails ou demandes, n'hésitez pas à prendre contact avec moi ! (Cf : début du tuto)