Module Programmation: base algorithmique avec Unreal Engine
Travaux Pratiques 2

Premiers algorithmes en *Blueprint*

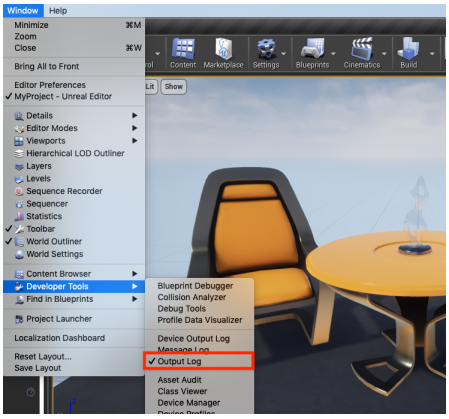
Objectif: L'objectif de ce TP est de se familiariser avec la programmation en *Blueprint*. Pour cela nous travaillerons dans un premier temps en mode console afin de réaliser le jeu du nombre mystère.

Partie I: Premiers tests en mode console

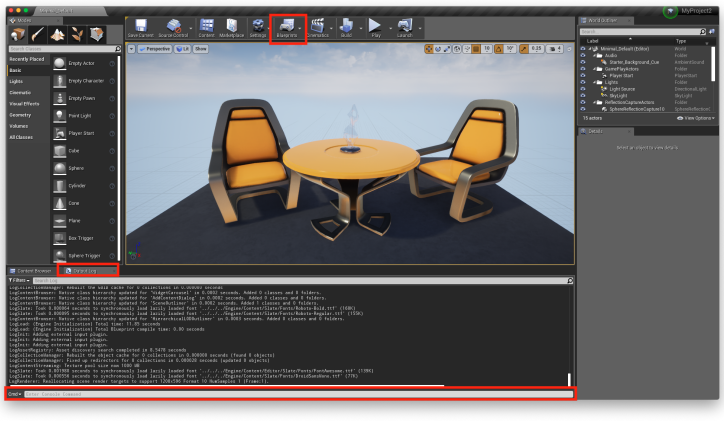
Dans cette partie, même si ce n'est pas forcément le moyen d'interaction le plus habituel, nous allons utiliser le mode console qui permet d'interagir avec la programmation *Blueprint*. L'intérêt sera principalement d'être capable de comprendre l'utilisation de variables, boucles, tests conditionnels qu'offrent la programmation *Blueprint*. A l'issue de cette partie vous serez capable de :

- Savoir rajouter et chercher une instruction *Blueprint*.
- Comprendre l'enchainement des instructions *Blueprint*.
- Rajoutez un événement personnalisé dans le *Level Blueprint*.
- Déclencher un événement depuis la console.
- Savoir utiliser un paramètre et une fonction.

1.1 Lancez un nouveau projet de type `Blank` et faites apparaître la fenêtre des logs de sortie (voir images (a) et (b) de la figure suivante).

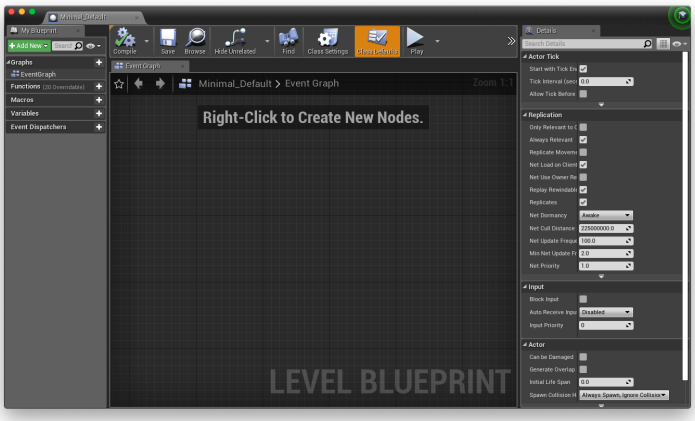


(a)

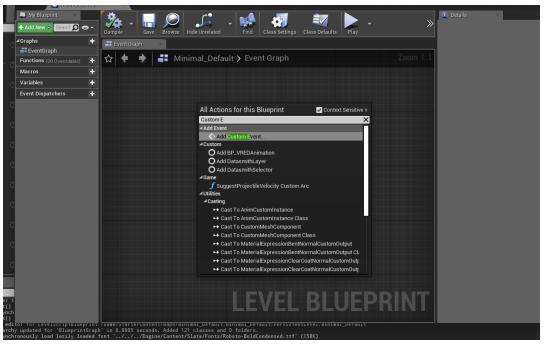


(b)

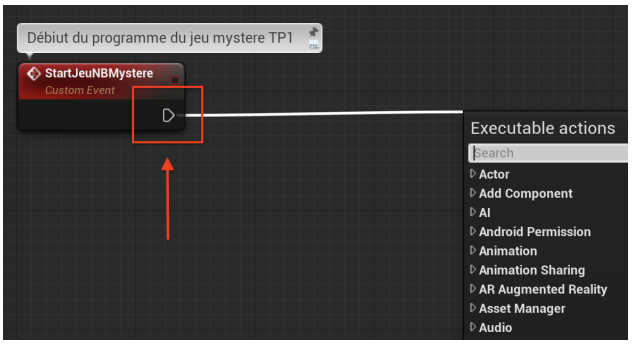
1.2 A partir du bouton *Blueprint* de l'image (b), ouvrez un *Level Blueprint* qui sera l'endroit où nous allons coder notre premier algorithme défini en *Blueprint*. Il code sera rajouté dans cette nouvelle fenêtre, comme sur l'image suivante.



1.3 Pour tester un premier programme, il s'agit de rajouter un nouvel élément associé à un événement que nous pourrons **ensuite déclencher depuis la console**. Il constituera le point de départ du programme. Pour ce faire, rajoutez un nouvel élément `Custom Event` que vous nommerez `StartJeuNbMystere`. L'ajout d'un élément *Blueprint* s'effectue à partir d'un clic droit dans la zone du *Level Blueprint* (voir image (a) ci dessous).



(a)



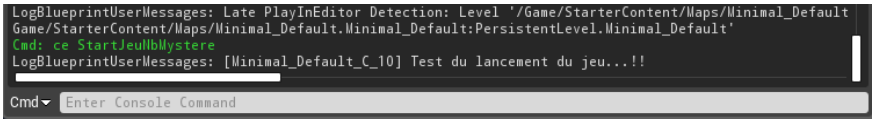
(b)

1.4 Sélectionnez l'extrémité du fil d'exécution du le bloc rouge précédant (voir image (b)), et ajoutez une instruction permettant l'affichage du texte : `Lancement du jeu du nombre mystère...`. Vous pourrez utiliser la fonction `print`. Cette dernière permet à la fois un affichage dans la console et dans l'écran de l'application du jeu. Pour l'instant, rien de doit s'afficher.

1.5 Le bloc rouge que vous avez rajoutez dans la question 1.3 peut être appelé directement depuis la console en utilisant la syntaxe suivante depuis la ligne `cmd` sous la fenêtre de la console (voir image (c) ci-dessous).

```
ce <nomDeEvenement> <paramètre>
```

Testez d'appeler l'événement créé précédemment (en remplaçant le champ `<nomDeEvenement>` par le nom du `Custom Event` de la question 1.3 et sans mettre de paramètres) et vérifiez que le message apparaisse bien dans votre console et sur l'écran de votre jeu. **Attention :** pour que cela fonctionne, il faut que l'application du jeu soit lancée (avec le bouton *play*). Vous devriez voir apparaitre un message comme celui de l'image (d) ci-dessous.

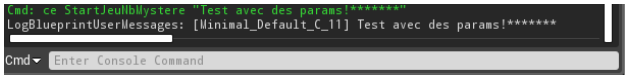


(c)

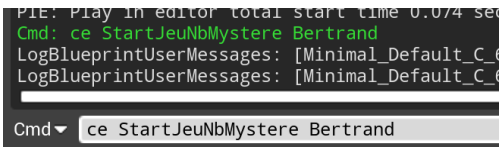
(d)

1.6 Rajoutez à l'événement précédant un *input* de type string. Cela peut être rajouté facilement en utilisant l'onglet *Détails* après avoir sélectionné l'objet associé à l'événement. Le bloc aura ainsi changé comme sur l'image ci-dessous. Raccordez le ensuite à la fonction `print` de façon à afficher le message donné par l'utilisateur.

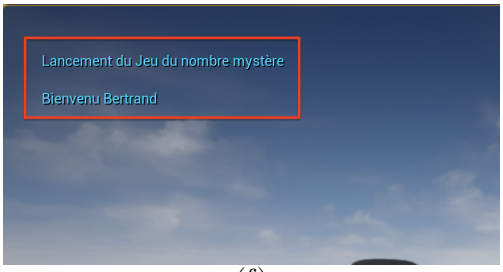
1.7 Testez le bon fonctionnement du passage des paramètres depuis la console. Si tout fonctionne correctement, vous devriez avoir un exemple comparable au suivant :



1.8 En utilisant la fonction `Append` et plusieurs appels à la fonction affichage, faire en sorte que l'utilisateur puisse passer son prénom en paramètre (image (e)) et que l'affichage l'exploite en affichant un message personnalisé comme sur l'image (f) ci dessous.



(e)



(f)

Après avoir utilisé vos premières instructions *Blueprint*, et exploité des paramètres, nous allons mettre en place la base de la structure d'un jeu.

Partie II: Base du jeu du nombre mystère

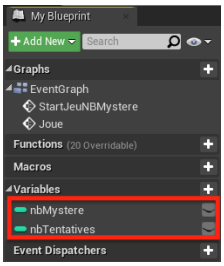
Le jeu du *nombre mystère* consiste à faire deviner un nombre aléatoire au joueur en lui donnant uniquement des indications d'infériorité ou de supériorité. Dans la suite, nous allons définir la base du jeu et à l'issue de cette partie vous serez capable de :

- Définir des variables dans *Blueprint* et leur donner des valeurs par défaut.
- Exploiter des structures de contrôle comme les `Branch`.
- Utiliser des tests booléen, incrémenter des variables.

2.1 Dans la fenêtre *Blueprint*, rajoutez deux variables qui vous servirons à définir les paramètres du jeu :

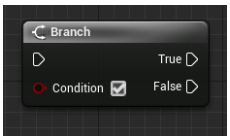
- `nbMystere` : représente la variable qui sera à deviner par le joueur. Pour l'instant vous pouvez choisir arbitraitrement sa valeur par défaut.
- `nbTentatives` : afin d'enregistrer le nombre de tentatives utilisées par le joueur.

Les variables peuvent être rajoutées dans l'onglet `My Blueprint` (voir image suivante). Il est demandé de donner des valeurs par défaut à toutes les variables.

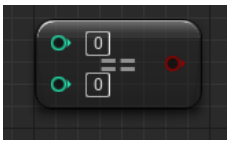


2.2 Rajoutez un événement personnalisé (de type `Custom Event`, voir question 1.3) qui correspondra à l'action de jouer et qui contiendra un champ *input* correspondant au nombre joué par l'utilisateur.

2.3 Rajoutez un test de façon à détecter si l'utilisateur a gagné ou non. Vous pourrez utiliser les blocs suivants :

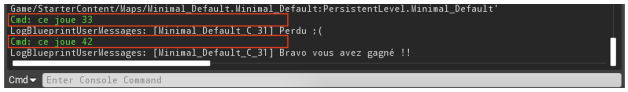


bloc de test conditionnel



opérateur ==

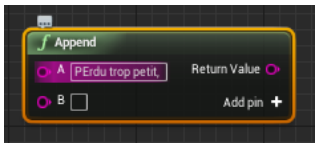
2.4 Testez en mode console que le programme détecte bien quand le joueur a gagné. Par exemple, si vous avez choisie le nombre mystère égale à 42 vous obtiendrez une exécution ressemblant à :



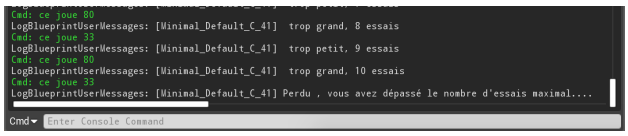
2.5 En vous inspirant du code précédant, complétez votre programme de façon à indiquer si le nombre joué est trop grand ou trop petit.

2.6 En utilisant la fonction `increment`, modifier le code de façon à le programme mette à jour le score du joueur (voir image ci-dessous).

2.7 Modifiez votre code de façon à afficher à chaque essai le nombre d'essai utilisé par le joueur. **Indication :** il existe une fonction permettant de fusionner deux ou plusieurs chaînes de caractères :



2.8 Rajoutez un nombre d'essais maximal en utilisant une nouvelle variable et détecter la fin du jeu. Vous pourrez arriver sur l'exemple suivant :



2.9 Modifiez votre code de façon à ce que le joueur puisse relancer une partie à partir de la console, sans relancer tout le jeu. Il s'agira de réinitialiser les variables `nbTentatives` et `nbMystere`. Pour l'instant la valeur de `nbMystere` sera réinitialisée par une constante différente de la valeur par défaut donnée à la variable.

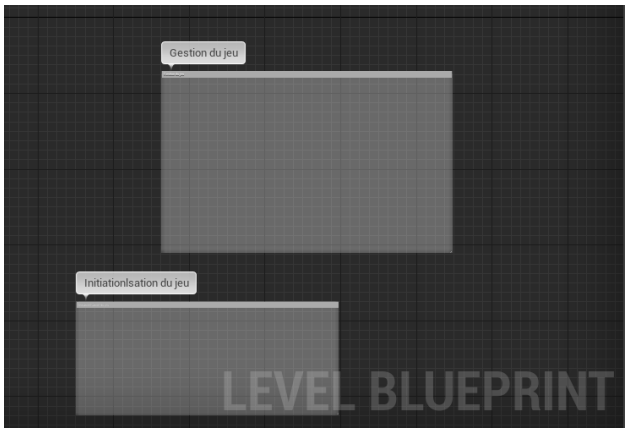
Partie III: Finalisation du jeu

Après avoir produit les bases du jeu, il s'agit de le finaliser en ajoutant la gestion d'un nombre mystère choisi aléatoirement et en améliorant l'affichage en fonction de la performance du joueur. Nous améliorerons aussi l'organisation du code à travers l'utilisation des **blocs de commentaires** et de `fonctions`.

Après avoir fait cette partie, vous serez capable de :

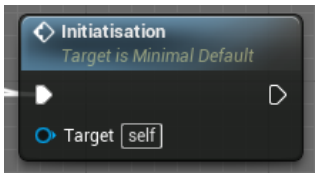
- Organiser un bloc de code commenté.
- Créer rapidement une `fonction` en *Blueprint* et l'utiliser.
- Générer et utiliser un nombre aléatoire en *Blueprint*.
- Avoir compris les liaisons des variables dans *Blueprint*.
- Appliquer des structures de contrôles itératives.

3.1 En sélectionnant à la souris un ensemble d'instructions, rajoutez des blocs de commentaires comme illustré sur l'image ci-dessous. Ces blocs seront associés aux fonctionnalités de `Gestion du jeu` et `Initialisation du jeu`.



On souhaite aussi que le jeu puisse se réinitialiser automatiquement lorsque le joueur a soit gagné, soit perdu en utilisant tous les essais autorisés. Afin de ne pas dupliquer le code de la question 2.9, nous allons voir dans la suite comment créer et utiliser une fonction.

3.2 Il est possible de créer rapidement une `fonction` avec la souris en sélectionnant les instructions que l'on souhaite y mettre. Une fois sélectionnées, à partir d'un clic droit vous utiliserez *collapse to function*. Une nouvelle `fonction` sera alors automatiquement crée. Vérifiez que cette dernière n'a pas altéré le bon fonctionnement de votre programme. Le code initial doit être remplacé par l'appel de fonction comme illustré ci-dessous.



3.3 Utilisez la fonction afin de réinitialiser le jeu à la fois quand le joueur à perdu au bout des dix tentatives autorisés et aussi lorsqu'il a gagné.

3.4 Pour que le jeu devienne réellement intéressant, changer l'initialisation du jeu en utilisant la fonction `RandomInteger` et en réglant à 100 le paramètre de la valeur maximale. La fonction devra désormais initialiser le jeu avec une variable aléatoire comprise entre 0 et 100.

3.5 Améliorez votre programme de façon à ce qu'il commente votre performance. En particulier, il pourra commenter en fonction de votre score :

— Score entre 1 et 3 : " Bravo, excellente stratégie"
— Score entre 4 et 6 : " Pas mal, bien joué"
— Score entre 7 et 10 : " Bien, mais vous pouvez vous améliorer"

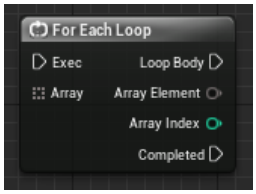
Indications : vous pouvez utiliser un `Branch` ou un `SwitchOnInt`. Afin d'avoir du code plus lisible, rajoutez une fonction qui prendra en paramètre un entier et retournera une chaîne de caractères associé au commentaire.

3.6 Améliorez votre programme pour détecter quand un joueur utilise bêtement deux fois le même nombre de manière consécutive. Dans ce cas il affichera un message d'avertissement et ne prendra pas en compte sa tentative.

3.7 On souhaite désormais stocker tous les nombres joués par l'utilisateur. Pour cela, rajoutez une nouvelle variable `listeJouee` de type liste d'entiers. Comme illustré sur l'image suivante :

3.8 Modifiez votre code de façon à ce que chaque nombre joué par l'utilisateur soit enregistré dans la liste précédente. L'enregistrement sera effectué même si l'utilisateur met deux fois le même nombre.

3.9 On souhaite à la fin du jeu pouvoir afficher toutes les tentatives de l'utilisateur. Rajouter une nouvelle fonction `AfficheTentative` qui affichera tous les nombres rentrés par l'utilisateur. On utilisera une structure itérative `For Each Loop` (voir image ci-dessous).



3.10 Reprendre la question 3.7 mais en faisant en sorte de détecter si un nombre a déjà été joué sans être forcément le dernier.

3.11 Améliorer l'affichage de votre jeu.

3.12 Rajoutez un mode difficile qui sera proposé à l'utilisateur avant le lancement du jeu. Le choix de l'utilisateur sera stocké dans une variable booléene.

3.13 Implémentez le mode difficile en répondant une fois sur trois par une fausse réponse.