Travail pratique #3

Objectif : Pratiquer les techniques de Réplication

Consignes :

- Votre programme doit être développé en C++ 17
- Votre programme doit utiliser le « build-system » CMake.
    - Si possible utilisez la version de visual studio 2019 qui ajoute les fonctionnalités Cmake nativement
    - Utilisez l'exemple git vu en cours pour le Cmake
- Le travail se fait en équipe de **trois** personnes
    - Pas deux, ni une
    - On est 33 donc théoriquement, dans une algèbre standard, c'est un multiple de 3
- Si le TP comporte des questions à développement, y répondre dans un README au format Markdown à la racine de votre Git
- Le rendu du TP se fera au moyen du gestionnaire de version Git
    - Un rappel de l'outil Git sera fait en classe
    - **Attention** : le graphe des contributeurs au projet peut être un excellent indicateur de qui à travailler et avec quelle proportion sur le projet
        - **Non un TP n'a pas besoin d'un chef de projet qui fait de la gestion**
- La date de rendu du TP est le **7 novembre 2021 à 23H59**
- Il vous est **INTERDIT** d'utiliser une librairie faisant la réplication
- **Rappels du plan de cours** :
    - Tout travail remis en retard, et ce, sans motif valable, sera évalué sur 50%. Les retards pour les travaux sont notés à raison de -10% par jour (p. ex. pour 2 jours de retard donnera -20% de la note sur 50% de l'évaluation).
    - Tout travail ne compilant pas sera noté sur 20% seulement
    - Un malus allant jusque -20% sanctionneront tous les travaux dont le français sera jugé incorrect.
- **Zone risquée :**
    - Si vous avez des étapes avant la compilation de votre projet veuillez m'indiquer les démarches à suivre dans un README au format Markdown à la racine de votre Git.

Énoncé:

L'équipe gameplay avance sur son jeu et va bientôt vouloir le tester en réseau. Vous allez devoir faire une première itération d'un système de réplication.

1er partie

Conseils :

La liste n'est pas exhaustive, il faudra rajouter différents éléments qui pourraient manquer ou non pour réaliser cette librairie.

Création des objets

1) Créez une classe **Player** et une classe **Enemy**
2) **Player** et **Enemy** héritent de **NetworkObject**
    a) **Player** et **Enemy** devront avoir une enum anonyme du nom du ClassID
    b) Methode virtuel de **NetworkObject**
        i) **Write**
        ii) **Read**
        iii) **Destroy**
    c) **Player** reprenez celui du TP2
    d) **Enemy**
        i) Ajoutez une position (3 nombres flottants)
        ii) Ajoutez un type (Enum)
            (1) Boss
            (2) Sbire
        iii) Ajoutez une rotation (4 nombres flottants)
        iv) Ajoutez une vie (0-1000)

LinkingContext

1) Ajoutez une classe **LinkingContext** qui permettra de lier un identifiant d'objet avec une instance de **GameObject**.
2) Ajoutez deux **std::map**
    a) Une va lier un identifiant d'objet avec un pointeur sur cet objet
    b) Une va lier un pointeur d'objet vers un identifiant d'objet
3) Ajoutez une méthode permettant d'ajouter un pointeur sur un **GameObject** et un **NetworkID** au contexte
4) Ajoutez une méthode permettant de supprimer un pointeur **GameObject**
5) Ajoutez une méthode permettant d'ajouter un pointeur de **GameObject**
    a) Cette méthode devra par conséquent créer un nouvel identifiant réseau.
6) Ajoutez deux méthodes de récupération :
    a) Une qui vous retourne un **std::optional** d'identifiant réseau à partir d'un pointeur de **GameObject**
    b) Une qui vous retourne un **std::optional** de pointeur de **GameObject** à partir d'un identifiant réseau

ClassRegistry

1) Ajoutez une classe **ClassRegistry.** Cette classe doit être un singleton.
2) Ajoutez une **std::map** qui permet de lier des identifiants de classes avec les pointeurs de méthodes permettant de les créer.
3) Ajoutez une méthode templatée qui permet d'enregistrer une classe dans le registre.

a) Si votre type template se prénomme **T**, alors **T::mClassID**, avec **mClassID** étant le nom de la valeur de l'enum de votre identifiant de classe dans **GameObject**, est la manière d'accéder à votre identifiant de classe
b) N'oubliez pas qu'en C++ vous pouvez représenter un pointeur de fonction sous la forme d'un **std::function<>**

4) Ajoutez une méthode **Create** qui prend en paramètre un identifiant de classe et retourne le **GameObject** créer à partir de cet identifiant de classe.

ReplicationManager

1) Ajoutez une classe **ReplicationManager**
2) Ajoutez lui un **std::unordered_set** de pointeurs de **GameObject** qui contiendra l'ensemble des **GameObjects** déjà répliqués
3) Rajoutez un LinkingContext
4) Ajoutez une méthodes **Update**
   a) Elle va sérialiser tous les **GameObject** dans le **MemoryStream/Serializer** ainsi que l'identifiant de votre protocole et l'identifiant du paquet. N'oubliez pas de sérialiser aussi l'identifiant d'objet et l'identifiant de classe… 😉

2eme partie On ajoute le réseau ?

Serveur

- Le serveur devra créer un **Player** à chaque connexion.
- Lorsque 2 joueurs seront connectés il devra créer un **Enemy**
  - Evidement les objets devront être répliquer
- Updater de manière aléatoire les valeurs des **Player** et de **Enemy**

Client

- Quand le serveur envoie une donnée, vous devez désérialiser cette donnée via le ReplicationManager
- Ajoutez un texte dans la méthode Read de vos objets pour savoir ce qu'il se passe.

Conseil

- C'est via le ReplicationManager que vous allez utiliser votre système réseau.

Bonus 1 C’EST MOI LE PLAYER !

- Client : Maintenant faite que c’est le client qui modifie son player et l’envoie sur le réseau
- Server : Quand il reçoit un message d’un client il doit l’envoyer à l’autre client

Bonus 2 JE VEUX JOUER !

- On va créer un mini jeu tour par tour
- Rajoutez maintenant le système de RPC/Event
    - Contrairement à la réplication qui passe par la méthode Update le message sera envoyé directement
- **Player** peut maintenant attaquer **Enemy** via RPC
    - **Player** peut aussi se prendre des coups de **Enemy**.
- Libre à vous de rajouter autre chose si vous le souhaitez 😉