LOG8715

TP3: Gestion de latence

Version 3.0

Responsable: Olivier Gendreau

Chargé de laboratoire: Keven Chaussé

Auteurs: Samuel Bellomo, Martin Paradis, Louis-Philippe Lafontaine-Bédard et Keven Chaussé

Introduction

L’objectif de ce laboratoire est de comprendre les principes de bases de gestion de la latence. Pour ce faire, vous devrez implémenter une technique vue en cours sur une simulation simple synchronisée entre client-serveur. Vous aurez une implémentation client-serveur déjà fournie et devrez faire en sorte que même avec du délai, les clients et serveurs soient synchronisés.
Lorsqu’un des clients applique un input, les autres clients devront réconcilier leur état avec les données du serveur.
Le travail sera en équipe de trois.
L’implémentation sera faite dans l’engin Unity avec le projet de base fourni sur Moodle.

Travail à accomplir

Implémentation

Installation et ouverture du projet

Si ce n'est pas déjà fait, installez **Unity 2021.3.16f1** et ouvrez le projet pour la plateforme Windows.

Veuillez utiliser la version d'archive disponible au site:
https://unity3d.com/get-unity/download/archive

Assurez-vous d'utiliser la version spécifiée d'Unity, étant donné qu'une version différente demanderait une migration du projet. Des points seront enlevés si la version est différente.

Instructions pour faire fonctionner clients et serveurs

Vous devrez avoir une deuxième instance du jeu qui fonctionne en même temps que l'éditeur afin d'avoir un client et un serveur. Pour ce faire, ce projet utilise l'outil ParrelSync, qui permet de créer et synchroniser des clones de votre projet. Vous pouvez le faire à travers le sous-menu ParrelSync dans la barre de menu de l'éditeur. Ces clones fonctionnent à l'aide de liens symboliques et ne sont disponibles qu'en lecture seule, donc toute modification faite dans le projet de base sera reflétée dans les clones.
Vous pouvez aussi utiliser des builds pour connecter vos clients et serveurs entre eux.

Mode client et serveur

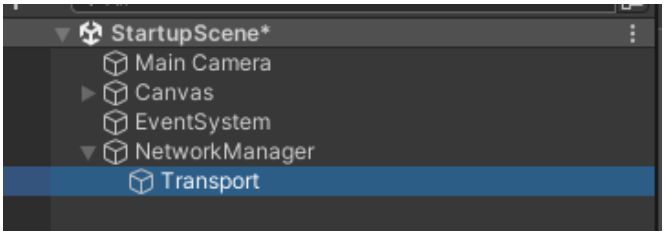
En démarrant la simulation, vous avez deux options. Démarrez la simulation en tant que serveur ou client via l'interface utilisateur.

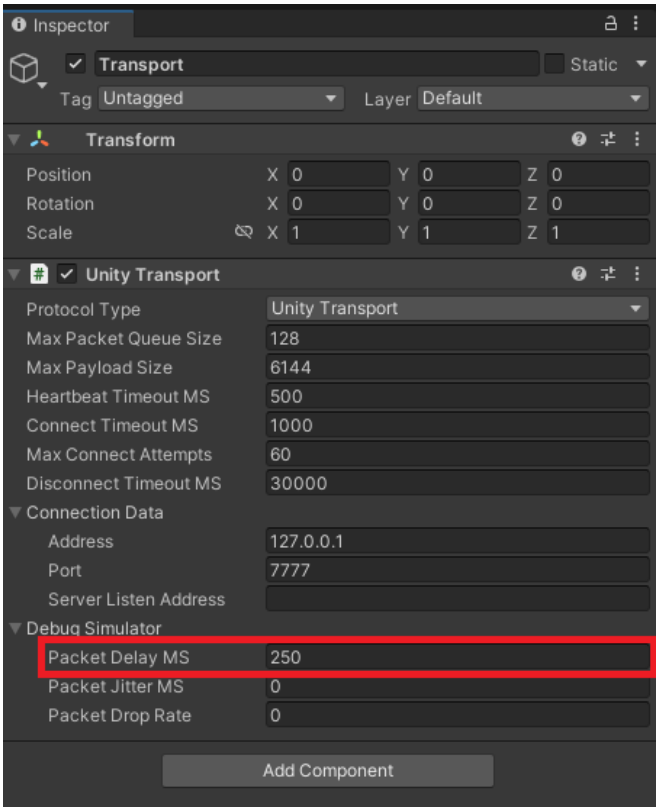
Vous devrez démarrer la simulation dans la scène StartupScene.

Latence simulée

Pour évaluer votre système de prédiction client, une latence sera ajoutée. Lors de la correction, votre solution sera évaluée avec des latences aléatoires entre 0 et 1 seconde.

La latence sera simulée grâce au Debug Simulator de UnityTransport. Vous pouvez modifier la configuration grâce au GameObject Transport se trouvant dans NetworkManager dans la scène StartupScene.





Changer cette valeur seulement sur l'instance qui servira de serveur. Le délai s'applique dans les deux sens donc un délai de 250ms sera effectivement 500ms avant que le client voit l'effet de son action. En réalité, il y aura quelques millisecondes supplémentaires dû au temps d'exécution d'une frame dans les deux sens.

ATTENTION : Vous devez tester votre implémentation avec des latences allant de 0 à 1000ms. Le 250ms dans l'image est seulement un exemple.

Fonctionnalités déjà existantes

La simulation contient déjà une implémentation de base.

Description de la simulation déjà existante:

- La simulation contient des cercles et des qui évoluent à l'écran.
- Les cercles se déplacent à l'écran selon leur vitesse
- Les cercles entrent en collision avec les bords de la simulation, mais pas entre eux.
    - Lors du contact avec un mur, les cercles auront leur vitesse changée par réflexion
- Les entités sont instanciées par le serveur, qui a autorité sur elles
- Les carrés sont des entités contrôlées par les clients
    - Lorsqu'un client se connecte au serveur, celui-ci va instancier un carré pour ce client
    - Un client peut déplacer son carré via les touches WASD
- Les clients peuvent interagir avec la simulation au moyen d'une capacité d' « étourdissement » en appuyant sur la barre d'espacement. Tous les cercles et tous les carrés seront figés pendant un court moment.

Information supplémentaire

- La classe NetworkUtility vous donne accès à différentes méthodes vous donnant accès à des attributs ou méthodes difficiles à trouver dans la hiérarchie du NetworkManager. Vous pouvez y ajouter d'autres méthodes si nécessaire.

Projet de base fourni

Le projet de base est implémenté en suivant le modèle de GameObjects. Il utilise la librairie de haut niveau Netcode For GameObjects pour la communication réseau. Voir https://docs-multiplayer.unity3d.com/netcode/1.1.0/about pour la documentation.

La création d'un client ou d'un serveur et la connexion entre eux sont gérées par le NetworkManager, dans la scène StartupScene. Il est appelé par les boutons de l'interface utilisateur.

Une fois le client ou le serveur démarré, vous serez amenés dans la scène MainScene.

Le script CircleSpawner s'occupe d'instancier le cercle sur le serveur et de les faire « spawner, » ce qui transmet leur information aux clients, qui eux aussi vont les instancier automatiquement.

Le prefab du cercle qui est instancié est composé de deux GameObject, un premier avec le script MovingCircle, et un second avec le script CircleGhost et un sprite. Le script MovingCircle définit la logique du cercle qui se déplace. C'est aussi lui qui s'occupe de la gestion de collisions. MovingCircle possède deux NetworkVariables (voir la documentation), qui sont des variables dans lesquelles seul le serveur peut écrire, mais qui sont automatiquement répliquées par le client, qui peut les lire. CircleGhost s'occupe de l'affichage du côté client. Il prend les données du MovingCircle pour mettre à jour sa position.

Le prefab du carré est similairement composé de deux gameobjects, Player, qui s'occupe de la logique et qui contient les données gérées par le serveur et PlayerGhost, qui s'occupe du rendement du cercle. Player inclut aussi le code de base pour le déplacement du cercle, prenant les inputs du client et les envoyant au serveur à travers une RPC (voir documentation).

Le script StunInputManager s'occupe de la gestion des inputs pour l'étourdissement. Lorsqu'un client appuie sur la barre d'espacement, il envoie l'information au serveur à travers une RPC.

Finalement, l'état général de la simulation est géré par le script GameState. Il contient notamment le champ ServerTime que le serveur met à jour périodiquement et la propriété CurrentRTT qui donne le RTT avec le serveur en secondes.

Fonctionnalités à développer

Gestion de la latence

Votre objectif est d'implémenter un système de gestion de la latence à l'aide de la prédiction de ghosts (cercles et joueurs) et de la prédiction d'inputs.

Un client devrait voir immédiatement les effets de ses inputs. Les entités devraient ensuite être réconciliées au besoin, par exemple lorsqu'un client reçoit la mise à jour du serveur suite aux effets de l'input d'un autre client.

L'objectif est donc de faire en sorte que le client puisse voir immédiatement les effets de ses inputs sur la simulation, tout en restant synchronisé avec le serveur et en minimisant autant que possible les corrections.

Veuillez prendre note qu'il n'est pas nécessaire d'implémenter les autres techniques vues en cours, telles que les techniques de fiabilité et d'interpolation. Seules la prédiction et la réconciliation client seront évaluées. C'est donc normal si une entité se téléporte parce que sa position a été corrigée. Par contre, lorsqu'il n'y a pas de correction, l'expérience devrait être la plus fluide possible. Il ne devrait donc pas y avoir de fausses corrections.

Restrictions

Vous devez implémenter votre prédiction et réconciliation client vous-mêmes. Vous ne pouvez pas utiliser les solutions fournies par Unity ou des librairies tierces.
Vous pouvez modifier les scripts actuels ou en ajouter de nouveau, mais les fonctionnalités existantes doivent rester fonctionnelles.

Rapport

Format PDF, interligne 1.5, 12 pts
Max 300 mots
Vous devrez rendre une courte discussion sur votre implémentation. À noter qu'une discussion implique de justifier vos points. Il vous est donc demandé de décrire votre implémentation, de la justifier et de discuter de la maintenabilité de votre solution.

Évaluation (/10)

Implémentation	
Rapport	3
Qualité générale de la gestion de la latence (expérience fluide pour le client, très peu de réconciliations visibles, sauf dans le cas d'un input d'un autre client)	3
Implémentation de la prédiction d'inputs	2
Implémentation de la prédiction de ghosts	2
Manque de clarté/maintenabilité	-2
Projet ne compile pas	-5
Mauvais format de remise et langue	-2
Retards	Perte de points exponentielle. jour 1: -1 jour 2: -2 jour 3: -4 **0 après 3 jours de retard**

Remise

Votre dossier de remise devrait contenir votre projet Unity. Le nom du fichier zip devrait contenir les matricules des coéquipiers.

LOG8715_TP3_matricule1_matricule2.zip

|___ ProjetUnity

|____ Assets/…

|____ ProjectSettings/…

|____ ….

Ne pas inclure le dossier Library et autres fichiers générés. Pour une liste complète de fichiers à éviter, voir le .gitignore à

https://github.com/github/gitignore/blob/master/Unity.gitignore

Si votre projet utilise git, vous pouvez utiliser la commande git

```
git archive -o TP3.zip HEAD
```

Pour créer votre archive zip.

Assurez-vous de ne pas inclure vos clones dans la remise.

Le projet ne devrait pas être très lourd, veuillez remettre le zip sur Moodle avant la **date indiquée sur Moodle.**