LOG8715

TP2: Optimisation

Version 5.0

Responsable: Olivier Gendreau

Chargé de laboratoire: Keven Chaussé

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

LOG8715 - TP2 - Date de remise : 17 mars 2024

Introduction

L'objectif de ce laboratoire est de mettre en pratique les principes d'optimisation de performance vus en cours pour améliorer les performances d'un projet fourni.

Le travail est composé de 4 exercices et d'un rapport. Chaque exercice correspond à une simulation différente que vous devrez optimiser.

Le travail sera en équipe de trois.

L'implémentation sera faite avec le moteur Unity avec le projet de base fourni sur Moodle.

Préparation

Installation et ouverture du projet

Si ce n'est pas déjà fait, installez **Unity 2022.3.6f1** 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.

Structure du projet

Le dossier Assets du projet est séparé en sous-dossiers selon l'exercice correspondant, et du dossier Main. Le dossier main contient un script simple qui a pour responsabilité de charger les scènes des autres exercices une après l'autre et de noter le taux de rafraîchissement de l'image dans la console.

Vous pouvez trouver la statistique de *FPS* utilisée pour évaluer le TP dans la console d'Unity et dans le fichier de log TP2.log (lorsque vous exécutez le *build*).

Pour chaque exercice, vous trouverez un fichier config.asset, qui sert à modifier certains paramètres. Les valeurs par défaut sont basses pour vous aider à profiler et faire vos tests pour commencer, mais pour évaluer les performances, vous devrez les modifier avec les valeurs qui sont données dans l'énoncé.

Mode *headless*

Votre exécutable doit être compilée en Release. Afin de ne pas être limité par la carte graphique du poste de test, un mode sans rendu graphique est disponible. À ce moment, seule la simulation sera prise en compte et le *bottleneck* devrait être la mémoire et le CPU.

Un *build* du jeu sera utilisé pour l'évaluation des exercices 2 à 4. L'évaluation sera faite en utilisant la ligne de commande:

`TP2.exe -batchmode -nographics -logFile TP2.log`

qui exécutera votre jeu en background.

Pour quitter le jeu, exécuter `taskkill /IM TP2.exe` sur la ligne de commande.

Pour voir vos statistiques, vous pouvez ouvrir le fichier `TP2.log` créé par l'exécution de votre exécutable.

L'évaluation lancera la scène Main, et chargera par la suite les scènes des exercices 2, 3 et 4 une après l'autre. Si vous voulez tester un exercice individuellement, vous pouvez modifier le fichier MainConfig.asset, dans le dossier Main/Config, à partir de l'éditeur. Vous pouvez retirer les scènes que vous ne voulez pas tester de la liste ScenesToLoad.

La performance sera évaluée sur les postes du L4818.

Partie 1 - Implémentation (/7)

Exercice 1 : Gestion de mémoire (/1)

L'objectif de cet exercice est d'optimiser la quantité de mémoire occupée par le programme. Vous devrez modifier les structs UnoptimizedStruct1 et UnoptimizedStruct2 pour minimiser l'espace qu'elle occupe en mémoire tout en conservant tous ses champs.

Seules ces structs peuvent être modifiées. Vous ne pouvez pas utiliser d'attributs comme StructLayoutAttribute pour changer la taille de l'alignement en mémoire.

Pour vous aider à y arriver, vous pourrez utiliser le MemoryProfiler (https://docs.unity3d.com/Packages/com.unity.memoryprofiler@1.0/manual/index.html). C'est un outil qui permet de prendre des captures de la mémoire utilisée par l'application. Vous pourrez donc voir l'espace qu'occupe concrètement le struct et comparer le résultat de vos optimisations. (Voir la documentation ci-dessus pour comment l'utiliser). Vous pouvez y accéder à partir du menu

Window/Analysis/Memory Profiler.

La taille indiquée par le MemoryProfiler pour chaque struct sera celle utilisée pour l'évaluation.

Exercice 2 : Utilisation du profileur CPU (/2)

Fonctionnalités déjà existantes

- La simulation contient des cercles immobiles qui changent de couleur et des carrés qui se déplacent.
- La couleur des cercles est modifiée selon le temps en fonction d'une grille de couleurs définie dans Grid.cs.
- Les cercles sont disposés en carré et sont créés par la grille à l'initialisation.
- Chaque cercle possède un nombre de « points de vie » qui détermine l'intensité de sa couleur
- Les carrés réduisent les points de vie des cercles à proximité, alors que les cercles augmentent les points de vie de leurs voisins.
- Les points de vie d'un cercle ne peuvent pas dépasser leur maximum, ni descendre sous 0.
- Les carrés accélèrent en direction des cercles à proximité ayant le plus de points de vie.
- La taille de la grille et le nombre de cercles sont définies dans le fichier Ex2Config.asset.

Travail à faire

Cette simulation a plusieurs mauvaises odeurs de performance que vous devrez identifier et corriger.

Le profileur d'Unity peut être accédé à partir du menu *Window/Analysis/Profiler*. Le profileur en mode *deep profiling* peut être utile pour trouver les méthodes qui prennent le plus de ressources, mais il ne sera pas assez profond pour vous donner la solution. C'est à vous de comprendre ce qui se passe bas niveau et d'optimiser en conséquence.

Il vous est conseillé de prendre du temps au début de votre implémentation afin de planifier vos optimisations et de ne pas perdre du temps à implémenter une solution qui ne vous rapportera pas beaucoup de gains de performance.

Vous devrez modifier le fichier Ex2Config.asset avec ces paramètres pour l'évaluation :

Nombre de cercles à simuler: 8000

La simulation doit rouler à **60 FPS** avec ces paramètres sur un build en mode headless.

Restrictions:

Vous ne pouvez pas altérer les fonctionnalités de la simulation, seulement modifier leur implémentation. À cette fin, vous pouvez modifier tous les fichiers de cet exercice.

Exercice 3 : Conception d'un allocateur de mémoire personnalisé (/3)

Fonctionnalités déjà existantes

- La simulation contient des cercles qui évoluent à l'écran.
- Il y a deux types de cercles:
    - Des cercles dynamiques qui ont une vitesse et se déplacent à l'écran.
    - Des cercles statiques qui ne bougent pas.
        - Le ⅕ des cercles est statique.
- À chaque détection de collision entre deux cercles, la taille des entités dynamiques est réduite de moitié. Les cercles statiques ne changent pas.
    - Il n'y a pas de restitution au contact d'entités. Lors d'un contact, la vitesse des entités reste inchangée.
    - Une fois en dessous de la taille minimum indiquée dans la config, ces entités n'ont plus de collisions et se passent à travers l'une l'autre.
    - Lorsque l'entité entre en collision avec un bord de l'écran, en plus de rebondir son état se remet à celui d'origine. Ceci inclut sa taille et sa capacité à entrer en collision avec les autres cercles.
- Les entités ont une couleur qui change selon leur type.
    - Les entités statiques sont rouges.
    - Les entités dynamiques avec collisions sont bleues.
    - Les entités dynamiques sans collisions sont vertes.

Quelques options s'ajoutent dans ECSManager pour vous aider à déboguer. "Debug print" fera un log à chaque frame sur le contenu du component manager et "should display entity IDs" affichera à l'écran les IDs des cercles qui évoluent à l'écran.

Travail à faire

Cette simulation implémentant le modèle ECS utilise un dictionnaire de dictionnaires pour contenir ses composants, ce qui n'est pas optimal. Vous devrez donc modifier le ComponentsManager et déterminer et implémenter la ou les structures de données les plus appropriées pour cette simulation.

Les systèmes déjà fournis devraient continuer à exécuter de la même façon.

Vous devrez modifier le fichier Ex3Config.asset avec ces paramètres pour l'évaluation :

Nombre de cercles à simuler: 1000

La simulation doit rouler à **30 FPS** avec ce nombre de cercle sur un build en mode headless.

Restrictions:

Seul le code de ComponentsManager devrait être modifié. Vous êtes libre d'ajouter les fichiers et classes que vous voulez. Tous les autres fichiers seront remplacés à la correction. Vous devrez vous-même faire votre implémentation, vous ne pourrez donc pas utiliser les implémentations déjà existantes comme DOTS.

Le but de l'exercice est de vous faire expérimenter avec les structures vues en cours, votre implémentation devrait donc rester dans cet esprit.

Exercice 4 : Parallélisme (/1)

Fonctionnalités déjà existantes

- Cet exercice est basé sur une simulation de prédateurs et de proies.
- Un certain nombre de plantes, de proies et de prédateurs apparaissent au début de la simulation.
- Commun pour Plantes, Proies et prédateurs
    - Disparaît après un certain temps selon un décompte en secondes.
    - La vitesse du décompte peut changer selon certains facteurs.
- Plantes (Plant)
    - Immobiles
    - Leur taille diminue tout au long du cycle de vie jusqu'à atteindre 0 au moment de disparaître.
    - Elles réapparaissent à un endroit aléatoire à la fin de leur vie.
    - La vitesse de disparition est doublée pour chaque proie qui touche la plante.
- Proies (Prey)
    - Bouge à une vitesse constante en direction de la plante la plus proche.
    - Si une proie touche une autre proie, elles activent un flag de reproduction.
    - Une proie s'étant reproduite réapparaîtra à un endroit au hasard lors de sa mort.
    - La vitesse de disparition est divisée par deux pour chaque plante qui touche la proie.
    - La vitesse de disparition est doublée pour chaque prédateur qui touche la proie.
- Prédateurs (Predator)
    - Bouge à une vitesse constante en direction de la proie la plus proche.
    - Si un prédateur touche un autre prédateur, elles activent un flag de reproduction.
    - Un prédateur s'étant reproduit réapparaîtra à un endroit au hasard lors de sa mort.
    - La vitesse de disparition est divisée par deux pour chaque proie qui touche le prédateur.

- La configuration de la simulation est défini dans Ex4Config.asset

Travail à faire

Vous devez modifier les scripts pour qu'ils utilisent le package DOTS de Unity qui inclut un framework ECS, un Burst Compiler et un Job System.

https://unity.com/dots

Faites les modifications en plusieurs étapes. Dans votre rapport, vous devrez expliquer comment et pourquoi chaque étape améliore les performances.

- Implémentation d'un ou plusieurs IJob pour une exécution single thread utilisant la compilation Burst. (Ne pas oublier d'activer "Enable Compilation" dans le menu Jobs/Burst)
- Parallélisation des jobs
- Amélioration des algorithmes
- Modification du projet pour utiliser ECS for Unity.
- etc.

Restrictions:

Vous avez une grande liberté sur le projet, vous pouvez modifier, supprimer ou ajouter des scripts. Vous pouvez modifier, ajouter ou supprimer des Game Objects. Vous pouvez modifier, ajouter ou supprimer des components sur les Game Objects. Ce qui ne doit pas changer est le comportement de la simulation et l'état initial défini par la configuration.

Évaluation:

Vous devrez modifier le fichier Ex4Config.asset avec ces paramètres pour l'évaluation :

Nombre de plantes: 3000

Nombre de proies: 3000

Nombre de prédateurs: 2500

Grosseur de la grille : 12000

La simulation doit rouler à **30 FPS** avec ces paramètres sur un build en mode headless.

Partie 2 - Rapport (/3)

Format PDF, interligne 1.5, 12 pts

Max 600 mots, utilisation de diagrammes suggérée.

Vous devrez rendre une courte discussion sur les optimisations et la maintenabilité pour chacune de vos solutions pour ce TP. À noter qu'une discussion implique de justifier vos points. Il vous est donc demandé de décrire vos optimisations, de les justifier et de discuter de la maintenabilité de votre solution.

Des captures d'écran de vos résultats de profilage doivent être incluses dans le rapport.

Un diagramme de votre solution à l'exercice 3 doit être inclus dans le rapport.

Évaluation (/10)

Un *build* en mode headless sur la ligne de commande sera utilisé pour l'évaluation pour chaque exercice.

Vos simulations devront respecter les requis et atteindre les critères de performance pour chaque exercice.

L'évaluation de la performance finale de votre solution sera évaluée en l'exécutant sur les postes du laboratoire en mode *headless*, c'est-à-dire sans affichage graphique.

Implémentation	
Exercice 1	/1
Exercice 2	/1
Exercice 3	/2
Exercice 4	/3
Rapport	
Description (contenu, pertinence et clarté)	/3
Général	

Mauvais format / langue (remise et rapport)	-2
Projet ne compile pas	-5
Retards	Perte de points exponentielle. Jour 1: -2 Jour 2: -4 Jour 3: -8 **0 après 3 jours de retard**

Remise

Votre dossier de remise devrait contenir votre projet Unity et votre rapport PDF. Le nom du dossier devrait contenir les matricules des coéquipiers. Vous perdrez des points si vous ne respectez pas cette structure, le nom des fichiers et le nom des dossiers.

LOG8715_TP2_matricule1_matricule2.zip
|___ rapport.pdf
|___ ProjetUnity
|____ Assets/…
|____ ProjectSettings/…
|____ Packages/…

*Pour votre ProjetUnity, incluez seulement les dossiers précisez ci-haut. Les autres dossiers sont des fichiers générés automatiquement par Unity et ne sont donc pas nécessaire pour la remise. Exemple des dossiers et fichiers à ne pas inclure: Library, Logs, obj, Temp, UserSettings, Assembly-CSharp.csproj, Assembly-CSharp-Editor.csproj, *.sln, etc.*
***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 TP2.zip HEAD
```
Pour créer votre archive zip.

Le projet ne devrait pas être très lourd, veuillez remettre le zip sur Moodle avant 23h55 le 17 mars 2024.