Приклади багатопоточності в проекті №1.

Багатопоточність в моєму проекті використовується в деяких випадках. Наприклад, в мене є противник/страшилка (поки ще не визначився), яка переміщає кучу предметів до гравця одночасно **(Follower)**.

Тобто є декілька різних предметів, та коли гравець підходить в заздалегідь визначений радіус роботи цього скрипта, він починає рухати купу предметів до гравці. Цей рух не миттєвий, тобто предмети рухаються часто та помаленьку в сторону гравця. Також кількість предметів, яка рухається, може бути будь-якою.

Будь-яка кількість та те, що вони рухаються декілька разів на секунду (можуть і частіше, система виконана таким чином, що в якомусь заданому інтервалі часу, предмети оновлюють своє місце знаходження поближче до гравця) призводить до падіння продуктивність при великій вибірці об’єктів.

Тому для таких випадків я використовую багатопоточність. Замість того, щоб рухати предмети в одному потоці за чергою, я їх рухаю одночасно в різних потоках. На щастя юніті дозволяє це робити без проблем.

Тут я використовую `IJobParallelForTransform`, вид завдання (Job) в юніті, який дозволяє використовувати багатопоточність для модифікування позиції, повороту та розміру об’єкта в юніті.

Графік показує кількість затрачених ресурсів системи та швидкість. Показує він всі мої скрипти, функції і т.д., а також ті компоненти, які юніті сам використовує і к я не має доступу. Чим вище графік, тим більше ресурсів та часу скрипт, функція, компонент витрачає.

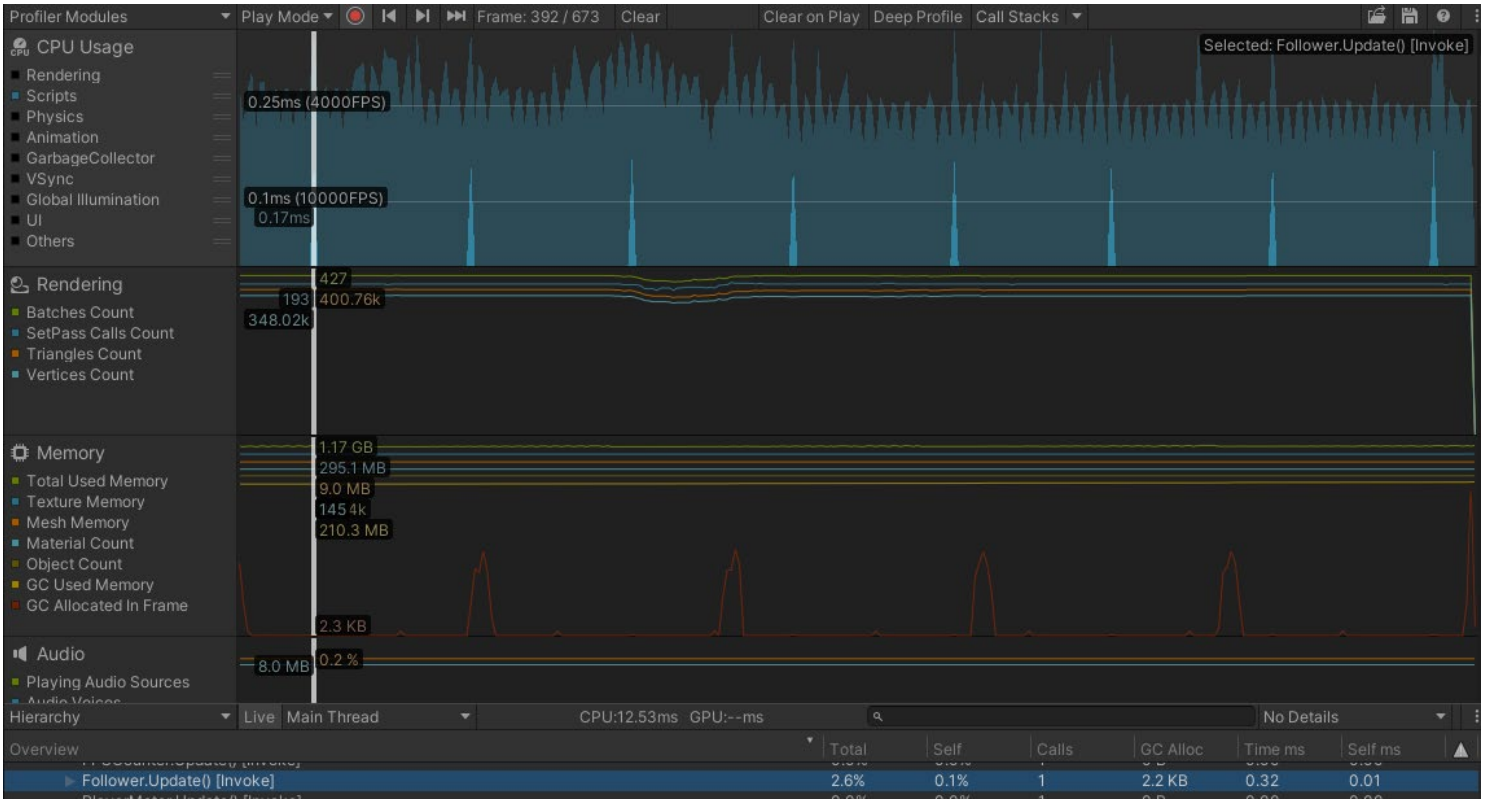
Нам цікавий лише ярко-синій колір, він покує час затрачений на наш скрипт, тобто на скрипт про який йдеться в цьому документі (Follower).

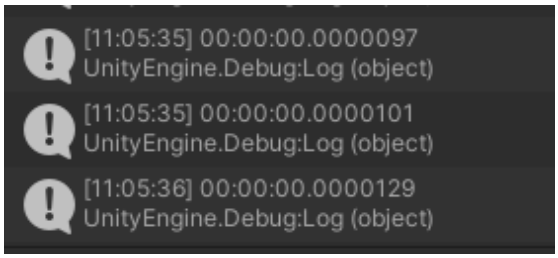
Другий скріншот – це скільки було затрачено часу на операцію, вираховується мною за допомогою стандартних інструментів c#. Цей час не максимально точний, але суть різниці між однопоточністю та багатопоточністю передає.

Деякі тести з різною кількістю об’єктів:

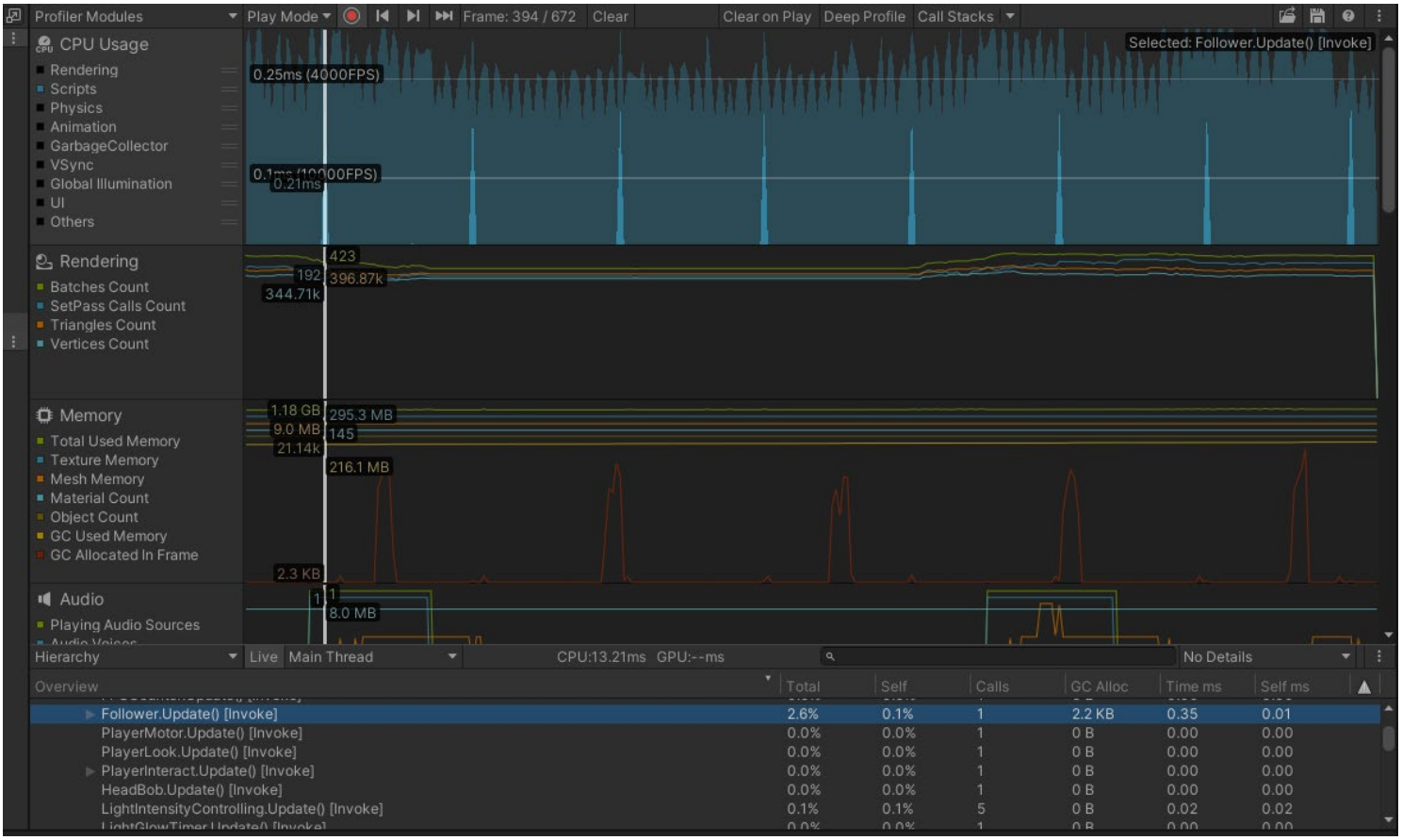
Управління 4 об’єктами.

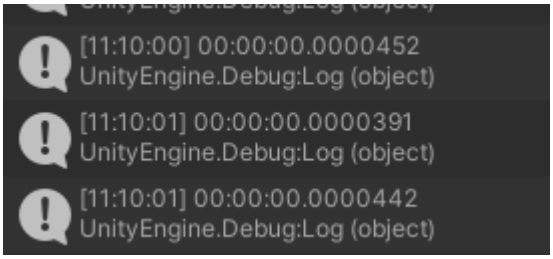
Операція **БЕЗ** потоків витрачає стільки ресурсів:





Операція **3** потоками витрачає стільки ресурсів:

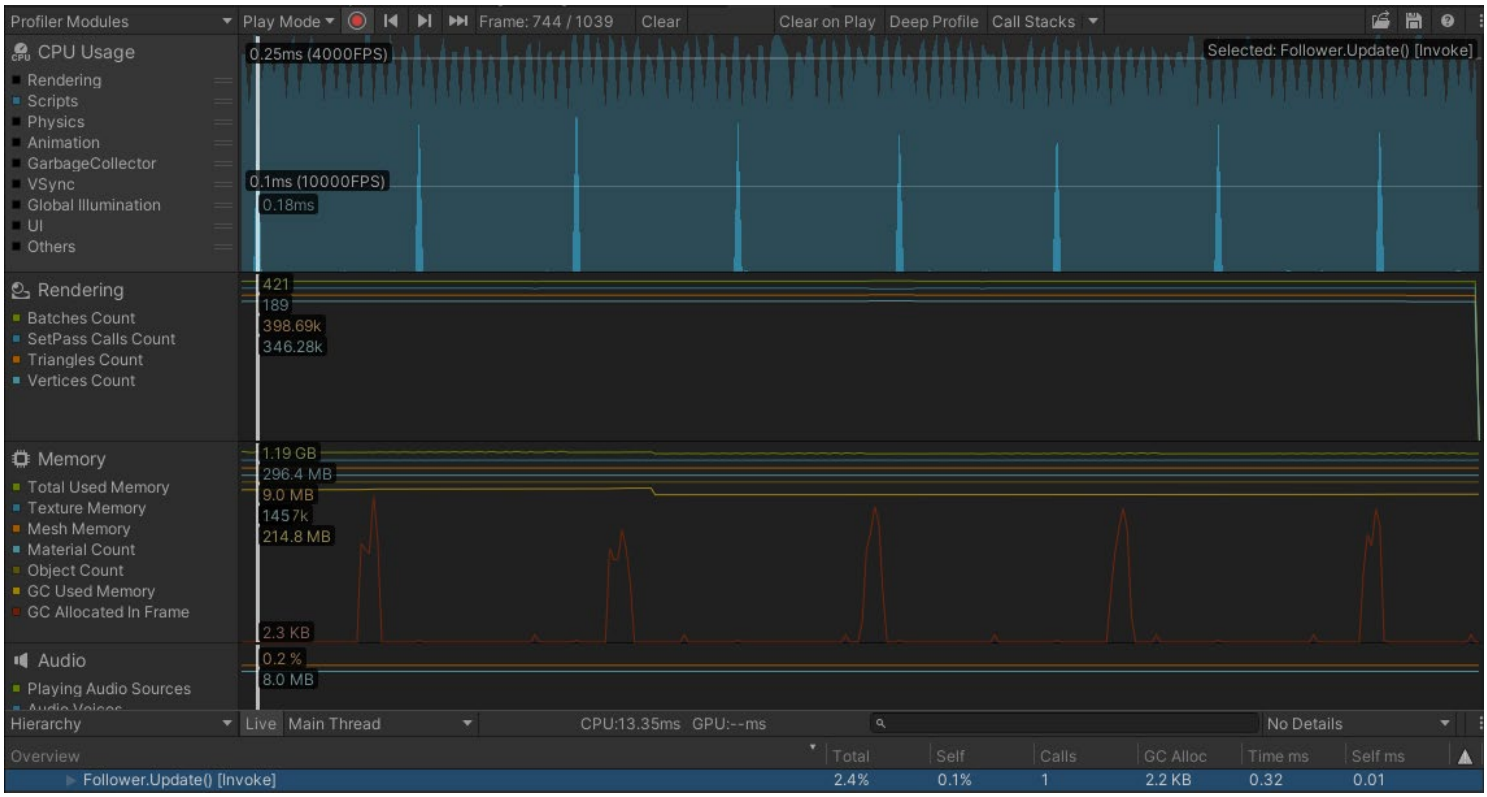


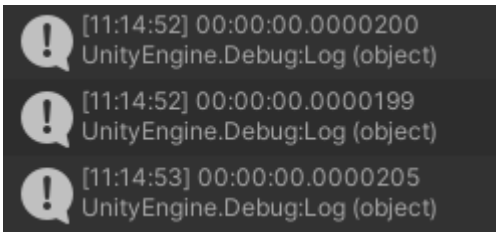


Тобто, як видно з результатів графіку та за часом, багатопоточність витрачає більше часу та ресурсів для управління 4 об’єктами. Перемістити 4 об’єкта в одному потоці буде значно швидше, ніж переводити цю дію на інші потоки, а потім їх синхронізувати.

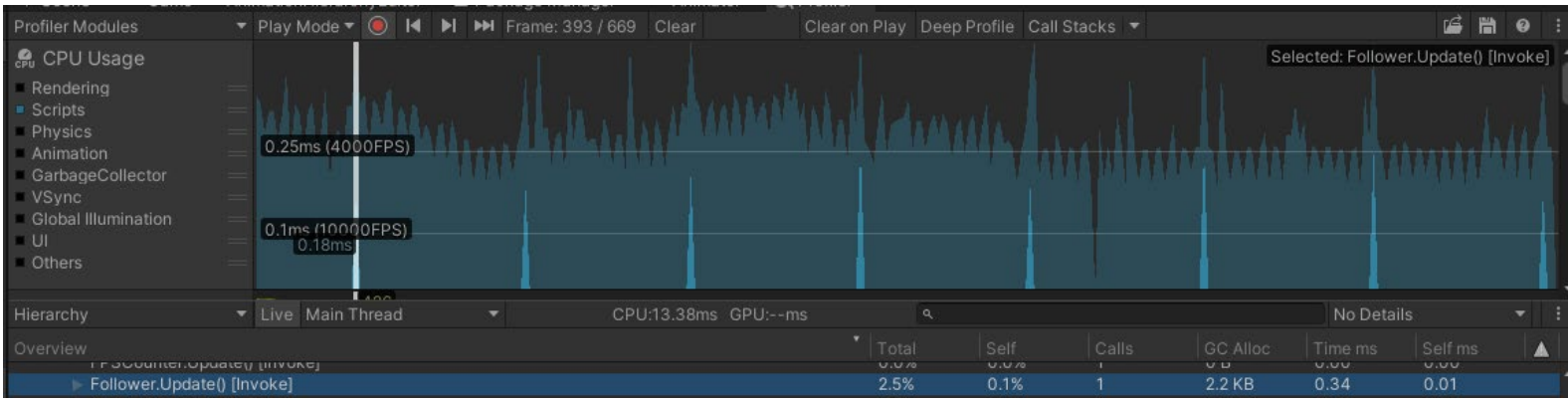
Управління 20 об’єктами.

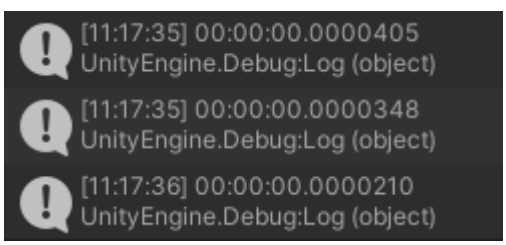
Операція **БЕЗ** потоків витрачає стільки ресурсів:





Операція **З** потоками витрачає стільки ресурсів:



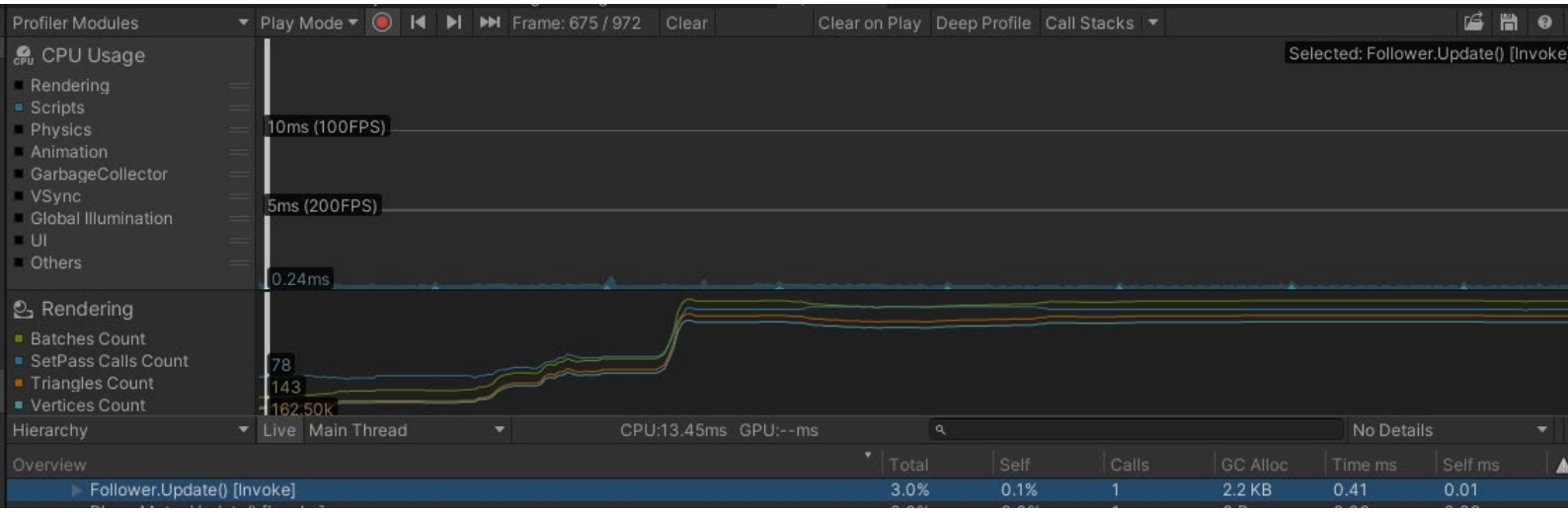


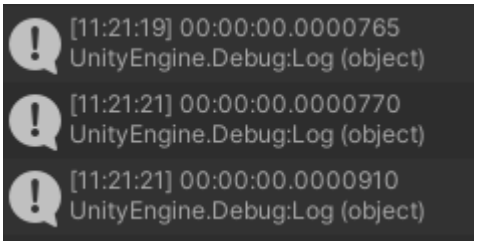
Тобто, як знову видно з результатів графіку та за часом, багатопоточність витрачає більше часу та ресурсів для управління 20 об’єктами. Перемістити 20 об’єктів в одному потоці буде швидше, але уже не значно, в цілому швидкість переміщення 20 об’єктів в багапоточності дорівнює швидкості переміщення 4 об’єктів там же.

Швидкість багапоточності не знизалась, а один поток став в декілька разів повільнішим. Тобто вже переміщення 20 об’єктів визивають у нього деякі проблеми.

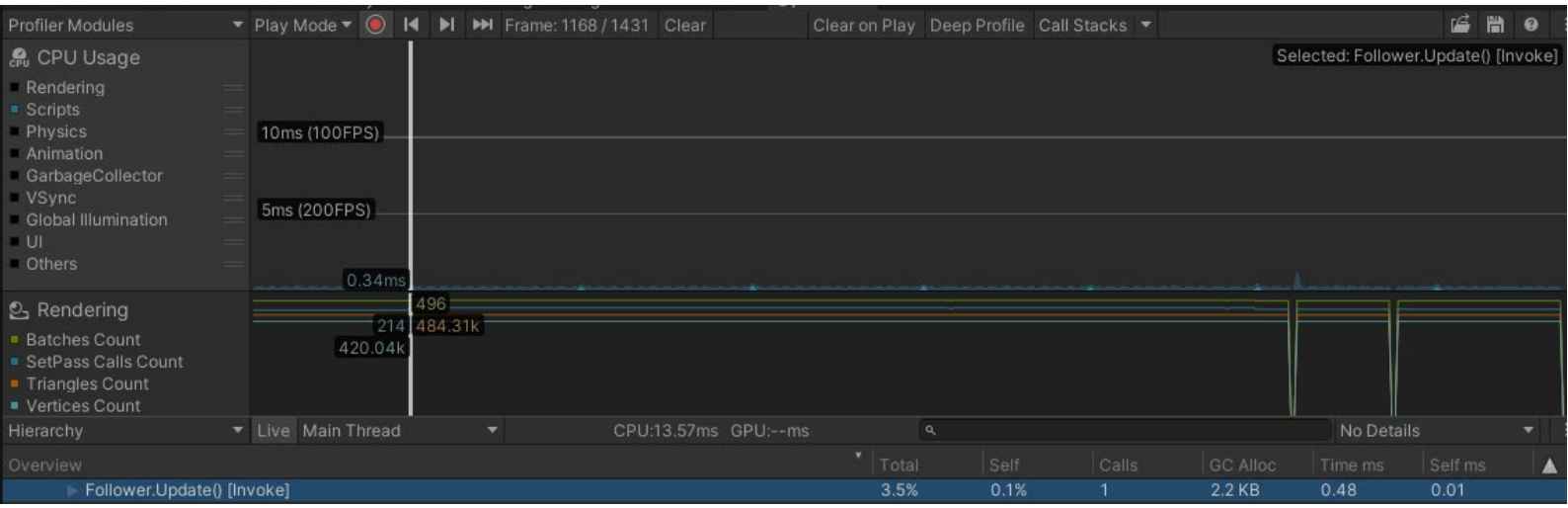
Управління 100 об’єктами.

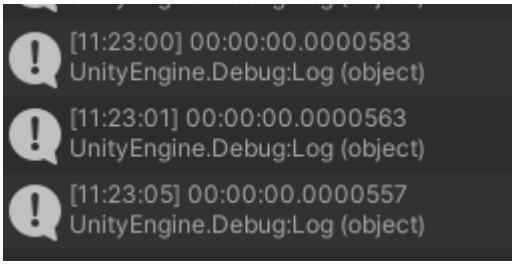
Операція **БЕЗ** потоків витрачає стільки ресурсів:





Операція **3** потоками витрачає стільки ресурсів:





Юніті поміняв трохи масштаб графік, але вони очевидно повільніші, ніж попередні. Як завжди потрібний нам скрипт, позначений яскраво-синім коліром, але тепер графіки менше.

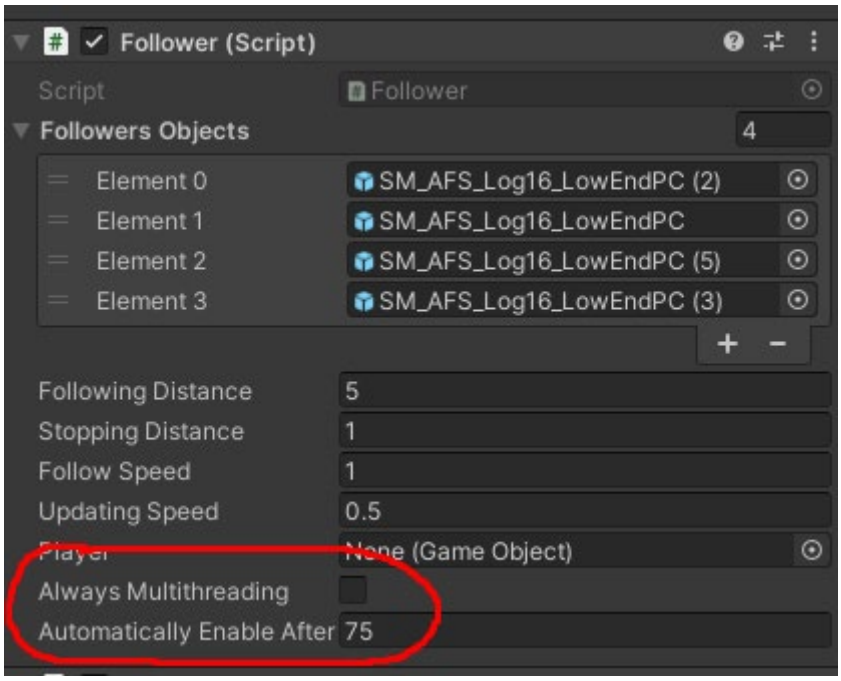
Тобто, як видно з результатів графіку та за часом, багатопоточність витрачає менше часу та ресурсів для управління 100 об’єктами. Перемістити 100 об’єктів в одному потоці буде значно повільніше, ніж в різни потоках.

Як раз для таких ситуацій багатопоточність підходить ідеально, коли потрібно зробити багато +- однакових операцій одночасно для великою кількість об’єктів.

Тому для оптимізації продуктивності в цьому скрипті я використовую багатопотність, коли багато об’єктів та однопоточність, коли власне мало об’єктів.

**Хочу зазначити, що код не був вставлен сюди через його об’єм, він доволі великий, подивитися його можна на гітхабі за шляхом нижче.*

Юзер інтерфейс:



Інтерфейс користувача полягає в двох параметрах.

Always Multithreading – це булевий параметр, як ясно з назви, якщо він увімкнений, то скрипт завжди буде працювати в багатопоточності.

Automatically Enable After – це інтовий параметр, якщо кількість предметів буде більше цього числа, то багатопоточність увімкниться автоматично.

Класи можете знайти за шляхом:

Assets/Scripts/Enemies/EnemiesType/Follower/Follower.cs

А також юніт тести

Assets/Tests/PlayMode/Enemies/EnemiesType/Follower/FollowerTests.cs