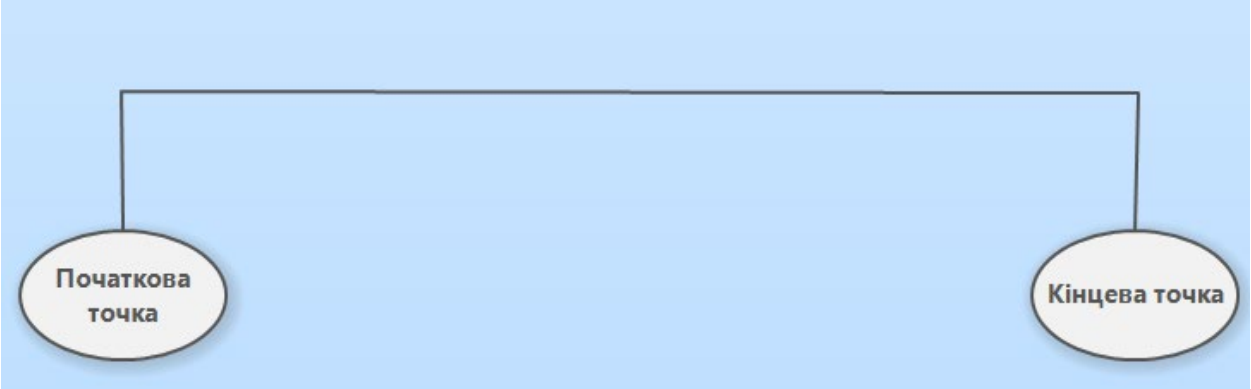
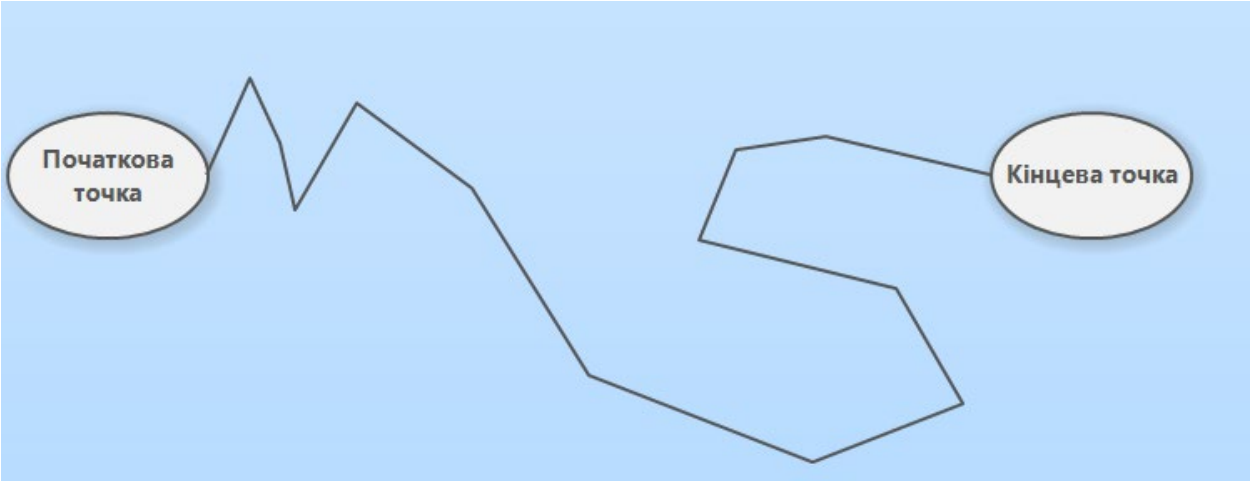
Приклади багатопоточності в проекті №2.

Багатопоточність в моєму проекті використовується в деяких випадках. Наприклад, в мене є генерація рівнів, або якщо точніше “псевдогенерація”. Це коли рівні вже зроблені (зазвичай зроблені в ручну) але йдуть в різному порядку. Тобто при кожному запуску гри в мене шлях відрізняється.

Якщо спробувати це зобразити, то буде виглядати це так:

Є дві точки, перша – це де ми з’являємось, друга – це кінець, остаточна локація. На даний момент вони завжди однакові, тобто всякий раз коли ми запускаємо гру ми будемо починати з одного й того ж місця. Те саме з кінцевою точкою, завжди закінчуємо там. Але локації між ними розташовані у випадковому порядку.





Для генерації я використовую функцію випадковості. Тобто генерую шанси різних локацій на появу.

Це все відбувається в одному потоці (Main Thread), але при великий кількості локацій, я використовую вже багатопоточність.

Тести будуть не максимально точними, тому що клас та його функції викликаються при старті гри, як і багато інших функції, через що визначити точну інформацію доволі проблематично, але результати і не повинні бути точними, тому що вони слугують лише для прикладу.

Тут я використовую `IJobFor`, вид праці (Job) в юніті, який дозволяє використовувати багатопоточність для модифікування різних структур та простих змінних (типу float, int і т.д.).

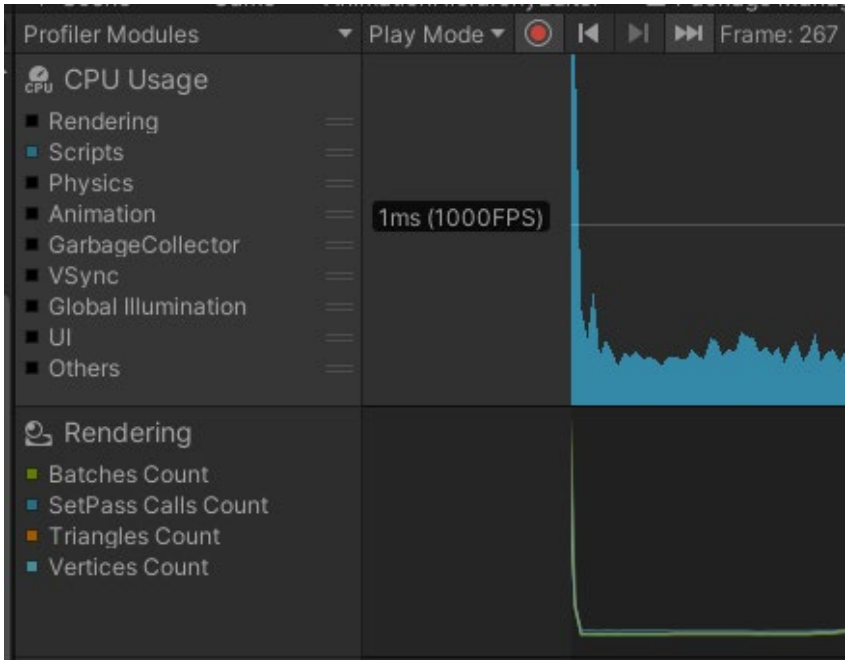
Графік показує кількість затрачених ресурсів системи та швидкість. Показує він всі мої скрипти, функції і т.д., а також ті компоненти, які юніті сам використовує і к я не має доступу. Чим вище графік, тим більше ресурсів та часу скрипт, функція, компонент витрачає.

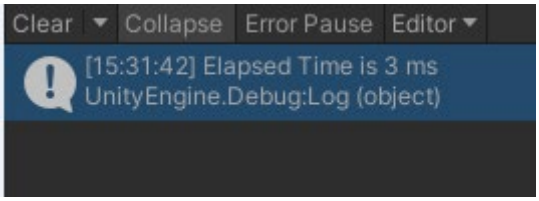
Elapsed Time – це скільки було затрачено часу на операцію, вираховується мною за допомогою стандартних інструментів c#. Цей час не максимально точний, але суть різниці між однопоточністю та багатопоточністю передає.

Деякі тести з різною кількістю локацій:

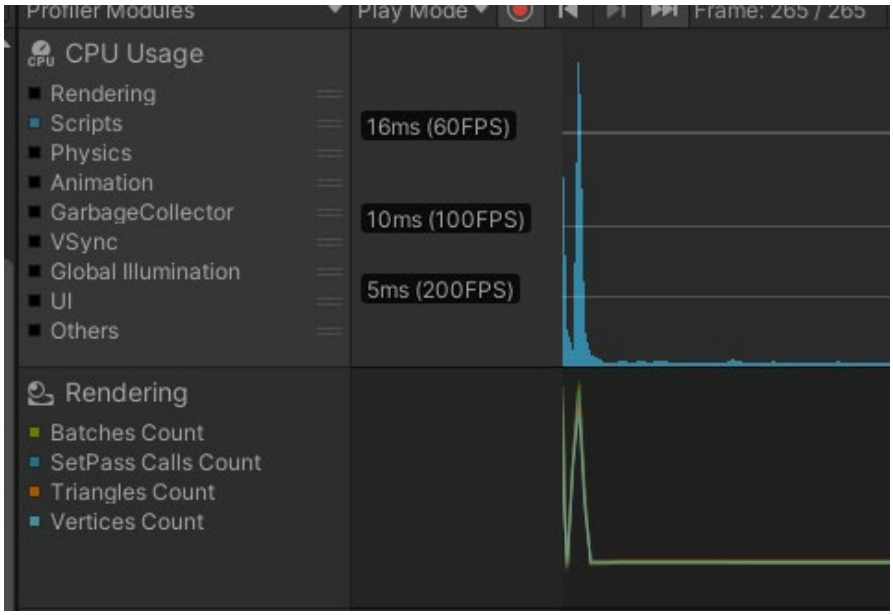
Генерація 20 локацій.

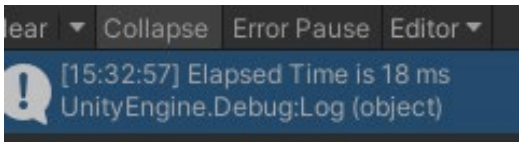
Операція **БЕЗ** потоків витрачає стільки ресурсів:





Операція **3** потоками витрачає стільки ресурсів:





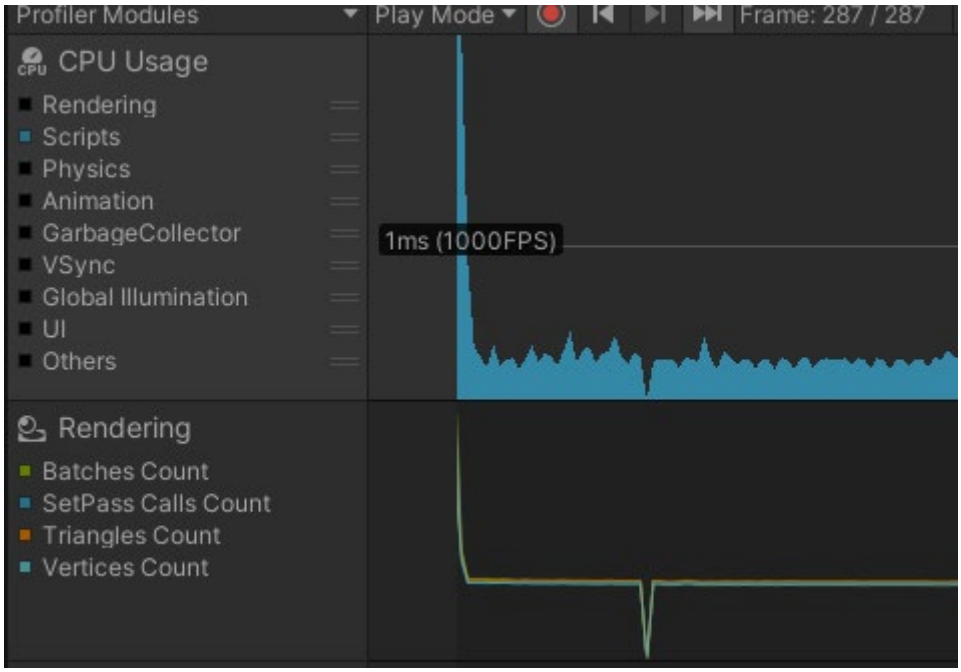
Нам цікавий лише синій графік на початку, оскільки клас визивається при створенні сцени, тобто на даний момент коли гра запущена в перші кадри гри, запускається цей клас. Він показує скільки операція забирає часу та навантаження на процесор. Чим вище, тим гірше.

Тобто як видно з результатів багатопоточність на невеликій вибірці показує себе не дуже. Тому сенсу використання їх для таких об’ємів немає.

Генерація 2100 локацій.

Операція **БЕЗ** потоків витрачає стільки ресурсів:

Операція **З** потоками витрачає стільки ресурсів:

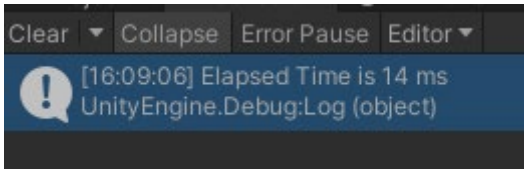


Як видно з потоками така вибірка працює вже значно швидше та займає куди менше ресурсів, ніж однопоточність. Як раз для таких великих завдань/обчислень потрібна багатопочність.

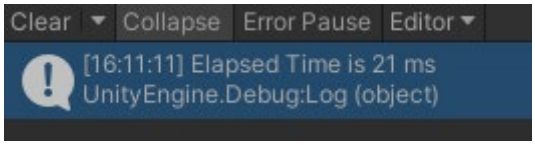
Ще декілька тестів, але без графіку, тільки з часом.

Генерація 500 локацій.

Операція **БЕЗ** потоків витрачає стільки ресурсів:

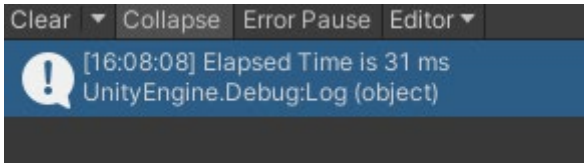


Операція **З** потоками витрачає стільки ресурсів:

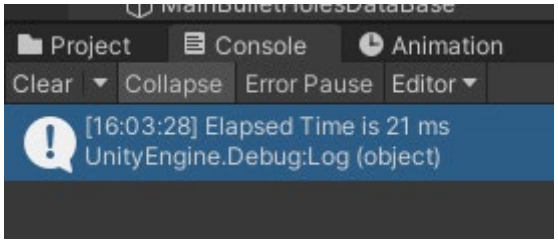


Генерація 1000 локацій.

Операція **БЕЗ** потоків витрачає стільки ресурсів:



Операція **З** потоками витрачає стільки ресурсів:



В цілому можу сказати, що приблизно на вибірці більше 800 локацій багатопоточність починає працювати значно краще, ніж однопоточність.

Така кількість локацій може бути потрібна, наприклад для нескінченного режима. Тобто режим, де рівні з’являються нескінченно доки гравець не програє.

Я використовую та буду використовувати змішаний режим, тобто при маленьких вибірках – однопоточність, при вибірка розміром більше 800 – багатопоточність. Це допоможе оптимізувати мою гру для будь-якої кількості локації.

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

Класи можете знайти за шляхом:

Assets/Scripts/Map/MapData.cs

А також юніт тести

Assets/Tests/PlayMode/Map/MapDataTests.cs