Тести використання потоків за для невеликих обчислень (можна сказати навіть мізерних).

Перший тест

Використання потоку за для обчислення періодичного переміщення камери по одній осі (HeadBob). Тобто коли гравець йде камера не статична, а динамічна, що створює ефект “ трясіння”, як в реальному житті.

Операція **БЕЗ** потоків виглядає так:

```
playerCam.transform.localPosition = new Vector3
    (playerCam.transform.localPosition.x,
    defaultYPos + Mathf.Sin(timer) * walkBobAmount,
    playerCam.transform.localPosition.z);
```

Змінюється тільки вісь y.

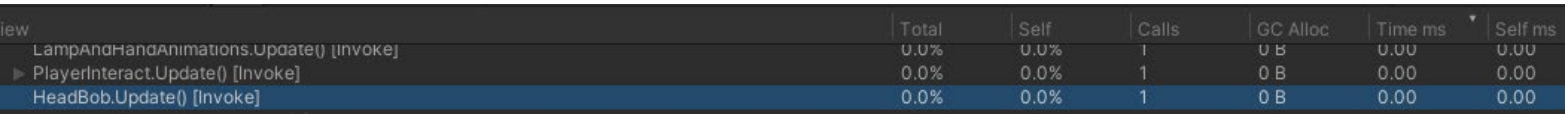
Операція **З** потоками виглядає так:

```
2 references
struct HeadBobJob : IJob
{

    public NativeArray<Vector3> _position;

    public float _deltaTime;
    public float _timer;
    public float _defaultYPos;
    public float _walkBobAmount;


    0 references
    public void Execute()
    {
        _position[0] = new Vector3(_position[0].x, _defaultYPos + Mathf.Sin(_timer) * _walkBobAmount, _position[0].z);
    }
}
```

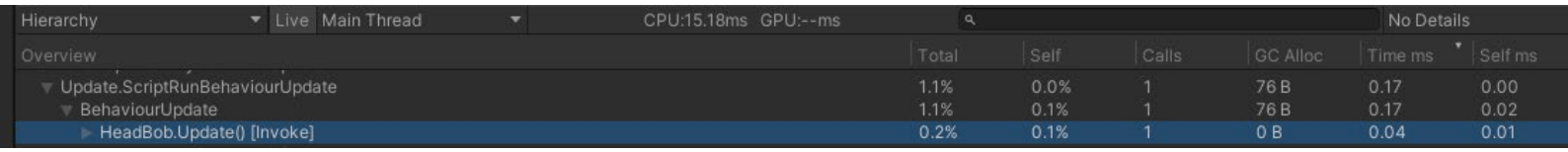
А також оголошення всіх змінних.

```
NativeArray<Vector3> position = new NativeArray<Vector3>(1, Allocator.Persistent);
position[0] = playerCam.transform.localPosition;
HeadBobJob job = new HeadBobJob()
{
    _position = position,
    _deltaTime = Time.deltaTime,
    _timer = timer,
    _defaultYPos = defaultYPos,
    _walkBobAmount = walkBobAmount
};
JobHandle jobHandle = job.Schedule();
jobHandle.Complete();
playerCam.transform.localPosition = job._position[0];
position.Dispose();
```

Операція **БЕЗ** потоків витрачає стільки ресурсів:



Операція **створення** потоків витрачає стільки ресурсів:



Сам потік (тобто операції в ньому) витрачає приблизно стільки ж ресурсів, скільки і операція **БЕЗ** потоку.

Статистика була взяти з основного потоку (Main Thread).

Тест показує, що використання багатопоточності в простих випадках не дає ніякого приросту до продуктивності, а навпаки створює непотрібне навантаження. Тобто сама по особі операція не тратить майже нічого, а якщо за ради неї почати використовувати багатопоточність, то вона починає витрачати ресурси в головному потоці на створення та підтримку нового.

Другий тест

На цей раз була взята функція, яка за допомогою деяких обчислень перевіряє чи може гравець залізти на поверхню. Тобто якщо в нас виставлено, щоб гравець міг забиратись на поверхні під кутом до 60 градусів, то якщо поверхня більше 60, гравець не повинен мати можливість залізти. За перевірку кута нахилу поверхні та блокуванні можливості забратися на неправильні поверхні, відповідає ця функція (функція SlopeCalculation в класі PlayerMotor).

Операція **БЕЗ** потоків виглядає так:

```
private Vector3 SlopeCalculation(Vector3 calculatedMovement)
{
    if (isGrounded)
    {
        float maxDistance = character.height / 2 - character.radius + groundCheckDistance;
        Physics.SphereCast(transform.position, character.radius, Vector3.down,
            out RaycastHit groundCheckHit, maxDistance, slopeLayer);
        Vector3 localGroundCheckHitNormal = transform.InverseTransformDirection(groundCheckHit.normal);
        float groundSlopeAngle = Vector3.Angle(localGroundCheckHitNormal, transform.up);
        if (groundSlopeAngle > character.slopeLimit)
        {

        Quaternion slopeAngleRotation = Quaternion.FromToRotation(transform.up, localGroundCheckHitNormal);
        calculatedMovement = slopeAngleRotation * calculatedMovement;
        float relativeSlopeAngle = Vector3.Angle(calculatedMovement, transform.up) - 90.0f;
        calculatedMovement += calculatedMovement * (relativeSlopeAngle / character.slopeLimit);
        }
    }
    return calculatedMovement;
}
```

Операція **З** потоками виглядає так:

```
struct SlopeCalculationJob : IJob
{

    public NativeArray<Vector3> _position;

    public Vector3 _localGroundCheckHitNormal;
    public Vector3 _calculatedMovement;
    public Vector3 _up;
    public float _slopeLimit;

    0 references
    public void Execute()
    {
        Quaternion slopeAngleRotation = Quaternion.FromToRotation(_up, _localGroundCheckHitNormal);
        _calculatedMovement = slopeAngleRotation * _calculatedMovement;
        float relativeSlopeAngle = Vector3.Angle(_calculatedMovement, _up) - 90.0f;
       _calculatedMovement += _calculatedMovement * (relativeSlopeAngle / _slopeLimit);
        _position[0] = _calculatedMovement;
    }
}
```

А також оголошення всіх змінних.

```
1 reference
private Vector3 SlopeCalculation(Vector3 calculatedMovement)
{
    if (isGrounded)
    {
        float maxDistance = character.height / 2 - character.radius + groundCheckDistance;
        Physics.SphereCast(transform.position, character.radius, Vector3.down,
            out RaycastHit groundCheckHit, maxDistance, slopeLayer);
        Vector3 localGroundCheckHitNormal = transform.InverseTransformDirection(groundCheckHit.normal);
        float groundSlopeAngle = Vector3.Angle(localGroundCheckHitNormal, transform.up);
        if (groundSlopeAngle > character.slopeLimit)
        {

                    NativeArray<Vector3> position = new NativeArray<Vector3>(1, Allocator.Persistent);
                    SlopeCalculationJob job = new SlopeCalculationJob()
                    {
                        _position = position,
                        _localGroundCheckHitNormal = localGroundCheckHitNormal,
                        _calculatedMovement = calculatedMovement,
                        _up = transform.up,
                        _slopeLimit = character.slopeLimit
                    };
                    JobHandle jobHandle = job.Schedule();

                    jobHandle.Complete();
                    calculatedMovement = job._position[0];
                    position.Dispose();



        }
    }
    return calculatedMovement;
}
```

Операція **БЕЗ** потоків витрачає стільки ресурсів:

Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
InputManager.FixedUpdate() [Invoke]	1.2%	1.0%	1	0 B	0.16	0.13
Physics.SphereCast	0.1%	0.1%	1	0 B	0.02	0.02

Операція **З** потоками витрачає стільки ресурсів:

Створення потоку

ew	Total	Self	Calls	GC Alloc	Time ms	Self ms
InputManager.FixedUpdate() [Invoke]	1.4%	1.0%	1	0 B	0.18	0.13
Physics.SphereCast	0.2%	0.2%	1	0 B	0.02	0.02
JobHandle.Complete	0.1%	0.0%	1	0 B	0.01	0.00
EditorOnly [DisposeSentinel.Create]	0.0%	0.0%	1	32 B	0.00	0.00
UnsafeUtility.Malloc	0.0%	0.0%	1	0 B	0.00	0.00
UnsafeUtility.Free	0.0%	0.0%	1	0 B	0.00	0.00

Та сам потік

verview	Total	Self	Calls	GC Alloc	Time ms	Self ms
PlayerMotor:SlopeCalculationJob	0.1%	0.0%	1	0 B	0.01	0.00

Результати доволі очевидні, створення потоку та обчислення однієї функції витрачає більше ресурсів, ніж просто обчислення однієї функції.

Ці тести були проведені для інтересу, подивитися чи буде хоч якись приріст з потоками. Звісно коли потоки використовуються для обчислення однієї функціх, то результату не буде ніякого та користь доволі спірна від такого. Але якщо би таких обчислень було багато і кожне з них було б в своєму потоці, то продуктивність була б значно краще, ніж без потоків взагалі.

Класи можете знайти за шляхом:

Assets/Scripts/CommonCore/PlayerScripts/HeadBob.cs

Assets/Scripts/CommonCore/PlayerScripts/PlayerMotor.cs