Master Scheduler Manual

Master Scheduler is a tool that allows you to execute your CPU heavy jobs in batches.

You use it by creating job queues and adding job to them.

Usage

All queue management is done through QueueHub.

To create a job queue:

```
QueueHub.CreateQueue("QueueName", int jobsPerFrame, bIsLooping);
```

QueueName: is the name of the queue. All queues are identified by name.

jobsPerFrame: determines the desired number of jobs to execute per frame.

bIsLopping: Tells the queue whether it should loop or delete jobs once they are executed. An AI job should loop, for instance, whereas a spawning job probably shouldn't.

This is another way to create a queue:

```
QueueHub.CreateQueue("QueueName", bIsLooping, int maxFrames);
```

With this function we tell the queue to run all jobs within the given maxFrames.

For instance, for a maxFrames = 5, all queued jobs will be executed within 5 frames.

Add a job to the queue:

```
QueueHub.AddJobToQueue("QueueName ", gameObject, Function);
```

Function can be any function in the given gameObject. It returns void and has no parameters.

You can convert a queue from a jobs per frame constraint into a max frames constraints at any moment, by using one of these two functions:

```
QueueHub.SetJobBatchSize("QueueName", int newSize);
```

Or

```
QueueHub.SetMaxFrames("QueueName", int maxFrames);
```

You can also remove specific jobs from the queue:

```
QueueHub.RemoveJobFromQueue("QueueName", GameObject Instigator);
```

Instigator being the game object responsible for the jobs we want to remove from the queue.

Destroy a queue:

```
        QueueHub.DestroyQueue("QueueName", false);
```

This will destroy the given cue. The second parameter determines whether the queue should execute all its jobs first or destroy immediately.

Example:

We have a shotgun that shoots 10 pellets at once with this code:

```
    void Fire()
    {
        for (int i = 0; i < 10; ++i)
            FireOnePellet();
    }
```

In order to turn this into a queued job, we will create a queue at start:

```
    void Start()
    {
        QueueHub.CreateQueue("ShotgunPellets", 2, false);

    }
```

That will create a queue that executes 2 jobs per frame and doesn’t loop.

Then, instead of calling FireOnePellet() directly, we will add it to the queue, thus Fire() becomes :

```
    void Fire()
    {
        for (int i = 0; i < 5; ++i)
            QueueHub.AddJobToQueue("ShotgunPellets", gameObject, FireOnePellet);

    }
```

With these changes, the Fire() function will shoot 2 pellets per frame, 5 times.

Notice that we only loop 5 times since the queue will run 2 jobs at a time.

Once the shotgun is thrown by the player, we can destroy the queue:

```
             QueueHub.DestroyQueue("ShotgunPellets", false)
```

In a real project, you could create a queue for every major task type at the start of the game, and add/remove jobs as needed.

Here’s an example of typical queue types:

AILow: a queue for AI that has to process a small number of jobs at any given moment

AIHigh: a queue that processes AI more frequently. As AI agents get close to the player, they’d leave AILow and join AIHigh for more frequent AI updates.

SpawningHeavy: a spawning queue that handles the spawning of costly objects (lots of components and heavy initializiation overhead)

SpawningLight: for ligther objects.

PhysicsQueries: a queue for all physics queries (raycasts, overlaps etc.)

If you have any questions, please e-mail me at Studiosthunderbytestudios@gmail.com