Setup

Setup is not required. Movement/Time will automatically add a Movement object to your uGUI event system GameObject (if you have one in your scene) or to a new GameObject named “Movement Effects”. This will happen at runtime the first time you use Movement.Run.

However, if you want to control your scene layout explicitly then feel free to add the Movement object to any object in your scene. Keep in mind that if the object you put it on is ever disabled then all movement effects will pause.

Principal

If you’ve ever used Unity’s coroutines before then the following block might look familiar:

```
public Transform StartItem;
public Transform EndItem;

void Start()
{
     StartCoroutine(MoveItem(StartItem, EndItem, 5f));
}

IEnumerator MoveItem(Transform start, Transform end, float timeframe)
{
     float startTime = Time.time;
     float startPos = start.position;
     float endPos = end.position;

     while(startTime + timeframe <= Time.time)
     {
         Start.position = Vector3.Lerp(startPos endPos,
                          (startTime - Time.time) / timeframe);

         yield return null;
     }
}
```

After years of developing in Unity, most of the people we know have used this general method to move things from one place to another hundreds of times. Even though there have been assets on the Unity asset store which can perform this same function we have always ended up falling back to coroutines for custom movement effects.

Movement/Time can be thought of as a way to take the above structure and customize it in any way you can imagine. Movement/Time takes the process of movement to the next level by redefining the lerp pipeline from the ground up, and giving the user complete control over every aspect of the effect, including time!

Usage

To use Movement/Time you define **Effects** and put them into **Sequences**. An Effect is a single movement and a sequence is a list of effects that run one after another. Once you have defined an effect or a sequence you can then start it going with Movement.Run().

Most of the fields that are defined in an effect use **actions** (which is another name for functions) rather than specific values. For instance, RetrieveEnd is an action which receives a class reference and returns the value that the effect should end at. The class reference can be anything. It is often set to the “this” pointer.

An Effect has the following fields:

float Duration – The number of seconds this effect should run for.
enum MovementSpace – Should the effect run in straight lines or around a sphere.

And the following actions which you can define. All actions except OnUpdate can be left as null to perform default behavior.

```
<TVal> RetrieveStart(<TRef > ref, <TVal> lastEndValue)
<TVal> RetrieveEnd(<TRef> ref)
void OnUpdate(<TRef> ref, TVal newValue)
void OnDone(<TRef> ref)
bool HoldEffectUntil(<TRef> ref, float TimeHeldSoFar)
float CalculatePercentDone(float RunningTimeSoFar, float EffectTimespan)
```

bool RunEffectUntilTime(float currentTime, float stopTime)
bool RunEffectUntilValue(<TVal> currentVal, <TVal> startVal, <TVal> endVal)

A sequence has the following fields:

<TRef> Reference – This is the value that is passed into all the functions.
bool StartWithVelocity – If you use Inertia or Elasticity this will “pre-warm” the sequence
bool SupressClosureWarnings – Don’t throw a warning if a closure is created.
bool IgnoreUnityTimescale – If set to true then the sequence will ignore the timescale variable in project settings.
float Inertia – This type of smoothing will round out any sudden changes in direction. Inertia will always lag behind the current position and will never overshoot.
float Elasticity – This type of smoothing will bounce around a bit and might overshoot.
enum SequenceTiming – Whether the sequence should run during the Update, FixedUpdate, or LateUpdate loop.
Effect[] Effects – An array holding the effects currently in the sequence.

It also has the following actions:

<TVal> RetrieveSequenceStart(<TRef> ref)
bool ContinueIf(<TRef> ref)
void OnComplete(<TRef> ref)

There is also a SequenceInstance variable which is returned by Movement.Run(). All variables on the SequenceInstance can be used to change the state of the instance as it is running (except for the PercentDone variable, which will just be overwritten with a new value if you change it.)

SequenceInstance has the following fields:

bool ExcludeFromPooling – If set to true then this object will be destroyed rather then recycled when it is finished.
bool SupressOnDoneCallbacks – OnDone will not fire for any effect in the sequence while this is set to true.
bool SupressOnCompleteCallbacks – OnComplete will not fire for any effect in the sequence while this is true.
bool Loop – The sequence will repeat until this is set to false.
float PercentDone – A value between 0 and 1 which represents the current percent done. Changing this value will have no effect, use the CalculatePercentDone action on the effect instead.
float Inertia – Changes the Inertia value on the effect while the effect is running.
float Elasticity – Changes the Elasticity value on the effect while the effect is running.
uint RecycleCount – Readonly. The number of times this SequenceInstance variable has been recycled.
float SequenceTimescale – Changes the timescale of the sequence while the sequence is running.
<TVal> Velocity – The current velocity of the sequence. Only used when elasticity is enabled.
<TVal> CurrentValue – The current position being passed into OnUpdate. Changing this will only have an effect if smoothing is enabled.
<TVal> StartValue – The starting position of the current effect.
<TVal> EndValue – The ending position of the current effect.
bool Initalized – Whether the effect has started yet.

Support

If you have any questions or experience any problems please contact trinaryllc@gmail.com