Phodex Framework

Anima

Installation

1. Extract the zip archive to your project folder.

2. Add the following line to your “Main.lua“ script file:

```
import("Scripts/Phodex Framework/Anima/Anima.lua")
```

3. Add “Animation:Draw()” to your “Main.lua“ script file, where the time and world gets updated (inside your main loop):

```
  --INSIDE YOUR MAIN LOOP
 if gamemenu:Hidden() then
       --Update the app timing
       Time:Update()

       --Update the world
       world:Update()

       --Draw Animations
       Animation:Draw()
   end
```

IMPORTANT: It is recommended, but not necessary, to give the animation system a reference to your main camera so animations of not visible objects don’t get rendered. To do so, simply use the camera variable of the animation class:

```
    Animation.camera = myCamera
```

4. You are ready to use the **Anima** commands!

Usage

The **Anima** animation system offers many use- and powerful commands to control your animations and to easily achieve complexes gameplay logic. Below you can find an explanation of every command. On the next page you can find examples for each command. Find an example script located in “Scripts/Phodex Framework/Anima/Anima Example.lua”. It should help you understanding the use of the **Anima Animation System.**

1. **Playing an animation:**

```
Animation:Play(model, bone, animation, speed, loop, blend, funcs, startFrame)
```

- **model:** The entity you want to play the animation to
- **bone:** The bone name as string you want to play the animation to (use the keyword “all” if you want to play the animation on all bones
- **animation:** The animation (animation name as string, or animation index as integer) you want to play
- **speed:** A float value for how fast the animation is being played
- **loop:** A Boolean value if the animation should get looped or not
- **funcs:** A two dimensional table containing information on which frame to call which function (explained in more detail below)
- **startFrame:** Sets the frame where the animation should start

2. **Play animations on specific bones (simultaneously):**

```
Animation:Play(model, bone, ...)
```

- Insert the bone name (as string) and the animation system will automatically find the corresponding bone entity. No need for referencing. If you want to play an animation on the whole model use the “all” keyword instead
- If an animation is played on a bone, all child bones will also play this animation. So if you, for example, play an animation on the left shoulder, the whole left arm will play this animation
- Using this will not override animation commands for parent (or unaffected) bones. So you can call Animation:Play(model, “all”, ...) and then override, for example, the left hand with Animation:Play(model, “LeftHand”, ...)
- This means you can basically play a different animation on every single bone of your entity

3. **Stopping an animation:**

```
Animation:Stop(model, bone)
```

- **model:** The entity you want to stop playing
- **bone:** The bone name as string you want to stop the animation on (use the keyword “all” if you want to play the animation on all bones. If this value is nil it is set to “all” automatically)
- **Hint:** If you stop an animation you can’t access it anymore with commands like Pause, or Set Animation. You have to start the animation again with Animation:Play()

4. Pause/Resume an animation:

```
Animation:Pause(model, bone)
Animation:Resume(model, bone)
```

- **model:** The entity you want to stop playing
- **bone:** The bone name as string you want to stop the animation on (use the keyword "all" if you want to play the animation on all bones. If this value is nil it is set to "all" automatically)

5. Using frame trigger functions:

```
 Animation:Play(model, bone, animation, speed, loop, blend,
           {
             {func, frame, tgt, arg1, arg2, arg3, arg4, arg5},
             {func, frame, tgt, arg1, arg2, arg3, arg4, arg5},
           })
```

- The trigger functions allow you to call a function on any subject you want, at a given frame. You can even add up to 5 arguments. As you can see the trigger functions argument is a two dimensional table, which gives you the powerful opportunity to add as many function calls, as you want
- **func:** The name of the function as a string
- **frame:** The frame on which the function is being called as an integer
- **tgt:** The function target (e.g. self) as reference
- **arg1/2/3/4/5:** the arguments the function should receive, as data type of your choice
- **Hint:** It is a good idea to use an indent as seen in the example above. If the usage is not 100% clear yet, see the example script

6. Change animation while playing:

```
Animation:SetAnimation(model, bone, anim)
```

- **model:** The entity you want to stop playing
- **bone:** The bone name as string you want to stop the animation on (use the keyword "all" if you want to play the animation on all bones. If this value is nil it is set to "all" automatically)
- **anim:** The new animation (animation name as string, or animation index as integer) you want to play
- **Hint:** The new animation start at the frame the current animation was

7. **Set animation frame while playing:**

```
Animation:SetFrame(model, bone, frame)
```

- **model:** The entity you want to stop playing
- **bone:** The bone name as string you want to stop the animation on (use the keyword “all” if you want to play the animation on all bones. If this value is nil it is set to “all” automatically)
- **frame:** The frame as integer to set the animation to
- **Hint:** The new animation start at the frame the current animation was

8. **Clear/Stop all animations:**

```
Animation:Clear()
```

- **Hint:** This method clears all active and inactive animations command and stops them from playing. It is a good idea to call it when you load a new map, because animations commands (even if they are not playing) will stay in the pipeline for as long as you stop them (which does not happen automatically if you reload a map). It is also useful, if you want to reset stuff.