Entities

General

Updating

Overview

The update functions can be used to make the entity run custom code at an interval. This can be used to give the entity custom behavior that is independent of the player.

HPL3 uses two different kind of update types. **OnUpdate** and **OnPostUpdate** runs at a fixed speed (60 times per second). Code that need to work the same regardless of the hardware and framerate of the user should be placed here. This would mainly be physics and gameplay related code. **OnUpdate** is called at the begining of a frame and **OnPostUpdate** is called at the end of it. It is not possible to know in which order the entities will be called, but **OnPostUpdate** will always be called after **OnUpdate** has been called for all entities in the scene.

The second update type is **OnVariableUpdate**. This will be called at a variable framerate and is not garanteed to work the same on all hardware. **OnVariableUpdate** is called right before the game is rendered. If the game is lagging **OnVariableUpdate** will be called less often than **OnUpdate** and **OnPostUpdate** that are forced to run at 60 FPS. Code that dont affect gameplay or that are purely aesthetic should be using the variable update function. **SetVariableUpdateRate** can be used to limit how often **OnVariableUpdate** is called, this should be used to stop performance heavy code from being run to often.

Code that is critical to the gameplay and that needs to work the same on all hardware should be using the fixed updates. All other code should be placed in the variable update since it gives better performance if the game is lagging.

Sleeping

Overview

When an entity no longer needs to be updated it should be put to sleep. This disables **OnUpdate**, **OnPostUpdate** and **OnVariableUpdate** from being called and saves performance. Entities that are asleep are still rendered and can be interacted with. An entity wakes up automatically if it is interacted with, collides with something, has an effect enabled (like Animation, MeshScale, BaseColorFade..) or if any of its bodies or attachments are woken up.

IMPORTANT
Props with active physics are currently set to be asleep when you enter a level. This means you can't put an object in mid-air and expect it to 'fall' to the correct ground position on startup of the level; you need to put it in the right place to start with. This is because otherwise there's a big performance hit

at level startup while all the physics are sorted out, and a collection of physics-triggered sounds.

To override this behaviour for a specific object, call **Entity_WakeUp()** on the entity.

Auto Sleep

Entities also have a setting for automatic sleep which is true as default. This automatically sleeps entities that have no effects enabled and if all of its bodies and attachments are asleep. AutoSleep should be disabled in the script class when it needs to ensure that Update/PostUpdate is called. AI entities do not sleep automatically (but can still be put to sleep manually).

Methods

These methods can be called in the Entity:

void Sleep()
void WakeUp()
bool IsSleeping()
void SetAutoSleep(bool)
**bool GetAutoSleep(**)

Entity Type

Player

This is a dummy object that's automatically created when the level starts. It's named “Player” and is useful for whenever you need to use the player as an entity, for instance to use it as a group for critters or attach something to it.

Camera

Like Player, this is a dummy object automatically created when the level starts. It's called “Camera” and is useful for, for example, attaching objects to the player's view (such as particle systems).

Prop

Overview

Props are used for any dynamic objects that do not any sophisticated AI. Boxes, closets, levers, etc all fit in here.

Creation notes

Must always have a mesh in the entity file. Apart from that pretty much anything goes.

Callback Methods

These are methods that are called from the prop:

void SetupAfterLoad(cWorld @apWorld, cResourceVarsObject @apVars, cResourceVarsObject @apInstanceVars)
void OnAfterWorldLoad()
void BeforeEntityDestruction()
void ResetProperties()
void OnSetActive(bool abX)
void GiveDamage(float afAmount, int alStrength, const tString&in asType, const tString&in asSource)
void OnUpdate(float afTimeStep)
void OnPostUpdate(float afTimeStep)
void OnVariableUpdate(float afTimeStep)
void OnStartMove()
void OnHealthChange()
bool CanInteract(int alType, iPhysicsBody@ apBody)
bool OnInteract(int alType, iPhysicsBody@ apBody, const cVector3f &in avFocusPos, const tString&in asData)
void OnConnectionStateChange(iLuxEntity@ apEntity,int alState)
void OnPhysicsCollision(iPhysicsBody @apBody, iPhysicsBody @apCollideBody, cPhysicsContactData&in apContactData)
float DrawDebugOutput(cGuiSet @apSet,iFontData @apFont,float afStartY)
void OnAttachmentUpdate(const cMatrixf&in a_mtxTransformAdd)
void GiveDamage(float afAmount, int alStrength, const tString&in asType, const tString&in asSource)

Agent

General

Agent is the building block for making anything needed more complex AI, like enemies or NPCs

Creation notes

Shall not have any bodies or joints in the entity file. Attach any effects either to nodes, sockets. If you skip attaching them to anything they will be attached to the mesh.

Callback Methods

These are methods that are called from the Agent

void SetupAfterLoad(cWorld @apWorld, cResourceVarsObject @apVars, cResourceVarsObject @apInstanceVars)
void OnAfterWorldLoad()
void BeforeEntityDestruction()
void ResetProperties()
void OnSetActive(bool abX)
void GiveDamage(float afAmount, int alStrength, const tString&in asType, const tString&in asSource)
void OnUpdate(float afTimeStep)
void OnPostUpdate(float afTimeStep)
void OnVariableUpdate(float afDeltaTime)
void OnHealthChange()
bool CanInteract(int alType, iPhysicsBody@ apBody)
bool OnInteract(int alType, iPhysicsBody@ apBody, const cVector3f &in avFocusPos, const tString&in asData)
void OnConnectionStateChange(iLuxEntity@ apEntity,int alState)
float DrawDebugOutput(cGuiSet @apSet,iFontData @apFont,float afStartY)
void SetupCharBody()
void GiveDamage(float afAmount, int alStrength, const tString&in asType, const tString&in asSource)
void HitByProp(float afAmount, float afMass, iPhysicsBody @apBody, cLuxProp @apProp)

Critter

Overview

Critters are meant to be smaller creatures that do not require advanced AI or behaviors. They are built up by boids and are controlled by having a lot of functions that add force to them.

Creation notes

A critter must have at least one body in the entity file. The mesh (or any other entities) may NOT be attached to this body. The shape of the body should almost always be a sphere. It does not need to cover the entire mesh and for long creatures it is probably best if it only covers the head. Do testing and see what works best.

Callback Methods

These are methods that are called from the Agent

void SetupAfterLoad(cWorld @apWorld, cResourceVarsObject @apVars, cResourceVarsObject @apInstanceVars)

void OnAfterWorldLoad()
void BeforeEntityDestruction()
void ResetProperties()
void OnSetActive(bool abX)
void GiveDamage(float afAmount, int alStrength, const tString&in asType, const tString&in asSource)
void OnUpdate(float afTimeStep)
void OnPostUpdate(float afTimeStep)
void OnVariableUpdate(float afDeltaTime)
void OnHealthChange()
bool CanInteract(int alType, iPhysicsBody@ apBody)
bool OnInteract(int alType, iPhysicsBody@ apBody, const cVector3f &in avFocusPos, const tString&in asData)
void OnConnectionStateChange(iLuxEntity@ apEntity,int alState)
float DrawDebugOutput(cGuiSet @apSet,iFontData @apFont,float afStartY)
void GiveDamage(float afAmount, int alStrength, const tString&in asType, const tString&in asSource)

Area

Overview

Used for various types of trigger areas

Callback Methods

These are methods that are called from the area:

void SetupAfterLoad(cWorld @apWorld, cResourceVarsObject @apVars)
void BeforeEntityDestruction()
void ResetProperties()
void OnSetActive(bool abX)
void OnUpdate(float afTimeStep)
void OnPostUpdate(float afTimeStep)
void OnVariableUpdate(float afDeltaTime)
bool CanInteract(int alType, iPhysicsBody@ apBody)
bool OnInteract(int alType, iPhysicsBody@ apBody, const cVector3f &in avFocusPos, const tString&in asData)
void OnConnectionStateChange(iLuxEntity@ apEntity,int alState)
float DrawDebugOutput(cGuiSet @apSet,iFontData @apFont,float afStartY)
bool OnStartCheckCollision(float afTimeStep, float afTimeSinceCheck)
Return true if collision should be check. Only available if SetCheckCollision has been set true.
bool OnCheckCollision(iPhysicsBody@ apBody, iLuxEntity@ apEntity)
Called for each body colliding. Only available if SetCheckCollision has been set true.
void OnEndCheckCollision(float afTimeStep
When all colliding bodies have been check. Only available if SetCheckCollision has been set true.
void GiveDamage(float afAmount, int alStrength, const tString&in asType, const tString&in asSource)

Liquid Area

Overview

A special kind of area hardcoded to make it extra fast. It basically gives anything in the area buoyancy. Normally the default should work, but you can use this to implement special types or add all kinds of effects easily.

Callback Methods

These are methods that are called from the liquid area:

void SetupAfterLoad(cWorld @apWorld, cResourceVarsObject @apVars)
void BeforeEntityDestruction()
void ResetProperties()
void OnSetActive(bool abX)
void OnUpdate(float afTimeStep)
void OnPostUpdate(float afTimeStep)
void OnVariableUpdate(float afDeltaTime)
bool CanInteract(int alType, iPhysicsBody@ apBody)
bool OnInteract(int alType, iPhysicsBody@ apBody, const cVector3f &in avFocusPos, const tString&in asData)
void OnConnectionStateChange(iLuxEntity@ apEntity,int alState)
void OnBodyEnterLiquid(iPhysicsBody @apBody)
void OnBodyExitLiquid(iPhysicsBody @apBody)
void OnCharBodyEnterLiquid(iCharacterBody @apBody)
void OnCharBodyExitLiquid(iCharacterBody @apBody)
void OnAttachmentUpdate()
float DrawDebugOutput(cGuiSet @apSet,iFontData @apFont,float afStartY)
void GiveDamage(float afAmount, int alStrength, const tString&in asType, const tString&in asSource)

Reverb Area

Overview

A special kind of area that is only used to set reverb properties for an area of the map. Does not allow variants to be created using script (like LiquidArea does).