Implementing Linked Lists

creationkit.com/index.php

This tutorial demonstrates how to implement Linked lists in Papyrus.

Linked Lists are a powerful programming tool that can accomplish some tasks with ease that would otherwise be quite difficult.

They are a bit like the linked references you see in the Creation Kit, but more versatile, as they can be dynamically created, used, and then deleted when they have served your purposes.

Essentially, each element in a linked list has references to the Next reference, possibly the previous reference and often, as a timesaver, a reference to the first reference in the list.

Contents

Difficulties

Coding Linked Lists in Papyrus is not an easy task, as a number of things work against it.

The primary difficulty is that there is no way to make a new instance of a user created type without already having an instance of it “in hand” - You cannot call a function that returns an instance of it from an empty variable, for example:

```
Function MyLinkedListFunction
    MyLinkedListType MyList = MyList.NewList()
; Do what you want
EndFunction
```

Will not work. You have to create a list element, in order to call the function that creates list elements. So just to use the above command, you need something more like:

```
GlobalVariableScript Property Gvar Auto

Function MyLinkedListFunction
    if !Gvar.MyLinkedListDummy ; At the start of any function wanting to use linked
lists
        Gvar.MyLinkedListDummy =
Game.GetPlayer().PlaceAtMe(GVar.MyLinkedListDummyType,1)

; That's two globals - one for the type for creation, and one for storage of the
dummy list item.

    endif
; Then, and only then,
    MyLinkedListType MyList = Gvar.MyLinkedListDummy.NewList()
; Do what you want
EndFunction
```

Be sure to link the global variable script property up to the quest script you're using to store these variables.

Controller Objects

And remember, local variables are not persistent unless they're in a persistent object, so you will need to put your linked list control code on a control object...and make sure there's only one of them. That's ANOTHER global of course:

```
Scriptname MyLinkedListController

Function MyLinkedListFunction
    if !Gvar.MyLinkedListDummy ; At the start of any function wanting to use linked
lists

; You'll want to replace Game.GetPlayer() with the Gvar.NodeSpawnPoint we define
later.

        Gvar.MyLinkedListDummy =
Game.GetPlayer().PlaceAtMe(GVar.MyLinkedListDummyType,1)
    endif
; Then, and only then,
    MyLinkedListType MyList = Gvar.MyLinkedListDummy.NewElement()
; Do what you want
EndFunction

Event OnInit()
    if !Gvar.MyLLController
        GVar.MyLLController = Self
        RegisterForUpdate(10)
    endif
EndEvent

Event OnUpdate()
; Periodic things you want to do with your lists here.
EndEvent

GlobalVariableScript Property Gvar Auto
```

You will need one controller for each different set of linked lists you need to use. Each set of lists might do something different on an update event, for example, and thus need separate OnUpdate events. For one set of lists that behaves differently in different circumstances, use states.

Global Variable Script

In order to have global references, you must attach a script similar to this one to an always running quest:

```
Scriptname GlobalVariableScript Extends Quest

LinkedList Property MyLinkedListDummy Auto ; The Dummy variable to create Linked
Lists

Activator Property MyLinkedListDummyType Auto ; The object type for the above
property.
```

```
ObjectReference Property NodeSpawnPoint Auto ; Where do we create nodes?

Activator Property NodeSpawnPointType Auto ; What is it?

FormList Property NodeTypeList Auto ; List of node types.

MyLinkedListController Property MyLLController Auto
```

Linked List prototype script

OK, once you have your controller more or less set up, it's time to create the actual node types. Here is a bare-bones Linked List Script, implementing a doubly-linked list with references to the first element (anything that can lower the number of operations needed in Papyrus is a good thing.)

You'll most likely also want a spawn point for the list nodes other than the player, particularly if your list nodes are actual physical objects...that's another global variable you will need.

And if your list nodes are physical items, you can make a FormList of them so you can have them look different (make sure to attach this script to all of them, and assign the property for the global variables!) - Anything you can attach a script to can be a linked list node.

Take a Deep breath:

```
Scriptname LinkedList extends ObjectReference
{Controls to store data and control links for Linked Lists. Special LinkedList types
should extend this}

import game
import utility

LinkedList property First Auto ; Reference to the First (Header)
LinkedList property Next Auto ; Reference to the Next Element.
LinkedList property Prev Auto ; Referebce to the previous element (Supports doubly-
linked lists.)

Form Property Data Auto ; Reference to the Data Item for the node. Can be anything,
the list doesn't care.

; Don't confuse the data reference with the list node, they are completely separate.

LinkedList Function MakeNewElement(int WhichType = -1)
    if !Gvar.NodeSpawnPoint ; We haven't got a place to put node lists yet!
        Gvar.NodeSpawnPoint =
FindClosestReferenceOfTypeFromRef(Gvar.NodeSpawnPointType,Game.GetPlayer(),100000) ;
Look for one.
        if !Gvar.NodeSpawnPoint ; None in the cell!
            Gvar.NodeSpawnPoint =
Game.GetPlayer().PlaceAtMe(Gvar.NodeSpawnPointType,1,true) ; Emergency...make one!
```

```
        endif
        Wait(2.0) ; Give a new Spawn Point time to settle in as the link holder.
    endif
    if Gvar.NodeTypeList
        if WhichType != -1 && WhichType <= Gvar.NodeTypeList.GetSize()
            return (Gvar.NodeSpawnPoint.PlaceAtMe(Gvar.NodeTypeList.GetAt(WhichType -
1),1,true) as LinkedList)
        else
            return
(Gvar.NodeSpawnPoint.PlaceAtMe(Gvar.NodeTypeList.GetAt(RandomInt(0,Gvar.NodeTypeList.
GetSize() - 1)),1,true) as LinkedList)
        endif
    else
        return (Gvar.NodeSpawnPoint.PlaceAtMe(Gvar.DefaultNodeType,1,true) as
LinkedList)
    endif
EndFunction

LinkedList Function NewList(form NewData, int Type = -1)
; Function to create a new list of a single element. Returns Reference to the
element.
    LinkedList FirstNode = MakeNewElement(Type)
    FirstNode.Data = NewData
    FirstNode.First = FirstNode
    return FirstNode
Endfunction

LinkedList function AddNewElement(form NewData, int WhatNodeType = -1,Bool
Append=false)

; Creates a new node holding the new data reference, and adds it to the List that
called it.

    LinkedList NewNode
    NewNode = NewList(NewData,WhatNodeType)
    NewNode.MoveToAnotherList(NewNode,self,Append) ; Insert NewNode into the list
this was called from.
    return Self ; Returns reference to the list.
Endfunction

; This next one may SEEM useless since you had a list and form reference already,
; but it is critical to be able to find out if a given reference is in a given list,
and where the actual
; node is, in case you want to move it to another list, remove it from the list, or
whatever.

; Again, don't confuse the node with the data...this only searches for the data
node, not any data IN the node.
; You have to write your own functions to do that.
```

```
LinkedList function FindListNode(Form FindThis)
; Search the list for a data item reference matching FindThis. Returns a reference
to the list element.
    if First.Prev
;        Debug.Notification("FindListNode: Bad First Link")
        Self.FixFirsts() ; Need to do this in case something external messed up the
reference to First.
    endif
    LinkedList CheckMe = First
    While CheckMe
        if CheckMe.Data == FindThis
            return CheckMe
        endif
        CheckMe = Checkme.Next
    Endwhile
    Return None ; Not in this list.
Endfunction

int function CountList(); Returns a count of the number of elements in the list it
is called on.
    if First.Prev
;        Debug.Notification("CountList: Bad First Link")
        Self.FixFirsts() ; Need to do this in case something external messed up the
reference to First.
    endif
    LinkedList Counter = First
    int Quantity = 0
    while Counter
        Quantity += 1
        Counter = Counter.Next
    endwhile
    return Quantity
Endfunction

LinkedList function FindElement(int N) ; Returns the Nth element of the list it is
called on.
    if N <= 0 ; If we were passed a zero or negative index
        Debug.Trace("Passed a negative index ("+N+") in FindElement Function")
        return None
    else
        if First.Prev
;            Debug.Notification("FindElement: Bad First Link")
            Self.FixFirsts() ; Need to do this in case something external messed up
the reference to First.
        endif
        LinkedList FindMe = First
        int Where = 1
        while Where != N &amp;&amp; FindMe
            FindMe = FindMe.Next;
            Where += 1
        endwhile
        if Where == N
            return FindMe
        else
            Debug.Trace("Passed Index larger than list size in FindElement
Function")
            return None
        endif
    endif
Endfunction


LinkedList function PickRandom() ; Returns a reference to a random element from the
list it is called on.
    return FindElement(RandomInt(1,CountList()))
Endfunction

function swap(LinkedList A,LinkedList B) global
; swaps the data items of the two nodes, effectively swapping the nodes.
    form HoldMe = A.Data
    A.Data = B.Data        ; Useful for sorting. Note you don't need to touch the
prev and next references at all.
    B.Data = HoldMe
endFunction

Function FixFirsts() ; used after activity that may alter the First node without
fixing all of the references.
;Use this if you do any external manipulation.
    LinkedList FixHere = Self
    While FixHere.Prev
        FixHere = FixHere.Prev
    endwhile
    FixHere.First = FixHere
    While FixHere.Next
        FixHere.Next.First = FixHere.First
        FixHere = FixHere.Next
    endwhile
EndFunction

; Here is a multi-use function. You can use it (as above) to insert a new element
into a list,
; remove an element from a list, or move an element between lists.

LinkedList function MoveToAnotherList(LinkedList FromList, LinkedList ToList,Bool
Append=False)

; Moves element it was called on from list FromList to list ToList - Returns
reference to ToList.

    if First.Prev
```

```
;        Debug.Notification("MoveToAnotherList: Bad First Link")
        Self.FixFirsts() ; Need to do this in case something external messed up the
reference to First.
    endif
    if Self.First == FromList.First ; If this evaluates to False, the item wasn't in
this list to begin with.
        if !FromList.Next ; There is only one element in FromList - special case.
            if ToList ; Last Element in previous list, New List has elements.
                Self.First = ToList.First
                if Append ; Put it at the end of the list, please.
                    LinkedList FindEnd = ToList.First
                    While FindEnd.Next
                        FindEnd = FindEnd.Next
                    endwhile
                    FindEnd.Next = Self
                    Self.Prev = FindEnd
                else ; Don't care where it goes.
                    self.next = ToList.First.next ; Link to Next Element in new
list.
                    self.prev = ToList.First; Link to header
                    if ToList.First.Next
                        ToList.First.Next.Prev = self ; Don't cut off the rest of
ToList!
                    endif
                    ToList.First.Next = self
                endif
                FromList = None ; !!! SEE NOTE BELOW
            else ; Last element in previous list, first element in new list.
                ToList = self ; !!! SEE NOTE BELOW
                FromList = None ; !!! SEE NOTE BELOW
            endif
            return ToList
        elseif self == self.first ; Another Special case, need to swap nodes first.
            Swap(Self,Self.Next) ; Put our data in the second node, and it's data
here.
            self.Next.MoveToAnotherList(FromList,ToList,Append) ; Take two, with
self in the second slot this time.
                    ; (Need to pass the same references in case one or the other
gets changed in the move)
        else
            if ToList ; Last List has more than one element, New List has elements.
                if Self.Next ; If there is a node after this one
                    Self.Next.Prev = self.Prev ; Link the next node's previous to
it's new previous node.
                endif
                Self.Prev.Next = Self.Next
; Link the previous node's Next to our next. Self is now removed from it's old list.
                if Append ; To the back of the line, you!
                    LinkedList FindEnd = ToList.First
                    While FindEnd.Next
```

```
                        FindEnd = FindEnd.Next
                    endwhile
                    FindEnd.Next = Self
                    Self.Prev = FindEnd
                else
                    self.next = ToList.First.next ; Link to Next Element in new
list.
                    self.prev = ToList.First; Link to header
                    if ToList.First.Next
                        ToList.First.Next.Prev = self ; Don't cut off the rest of
ToList!
                    endif
                    ToList.First.Next = self
                endif
                self.First = Tolist.First
                return ToList
            else ; Last list has more than one element, first element in new list.
                if Self.Next ; If there is a node after this one
                    Self.Next.Prev = self.Prev
; Link the next node's previous to it's new previous node.
                endif
                Self.Prev.Next = Self.Next
; Link the previous node's Next to our next. Self is now removed from it's old list.
                self.next = None ; No next item.
                self.prev = None ; No Previous Item.
                self.First = Self ; We are the header node of the new list.
                ToList = self ; !!! (See Note below) - New List now refers to us.
                Return ToList
            endif
        endif
    else
;         Debug.Notification("MoveToAnotherList: Item was not in FromList.")
    endif
Endfunction

Event OnInit()
    RegisterForUpdate(5000) ; Keep our data in existance until we are destroyed.
EndEvent

GlobalVariableScript Property Gvar Auto
```

```
!!! NOTE:  This will not actually work, since variables are Pass-By-Reference. Check
for these cases before calling this function.
```

Node Spawn Point Wrapup

Notice how the spawn point code waits a bit before making the first node after creating a new spawn point?

Just to make sure your spawn point item is working, It needs some code on it like this:

```
Scriptname LinkedListNodeSpawnPointScript extends ObjectReference
```

```
Event OnInit()
    if Gvar.NodeSpawnPoint ; Hey, there's one already!
        self.Disable()
        self.Delete()
    else
        RegisterForUpdate(5000) ; Keep our data in existance until we are destroyed.
        moveto (MyFavoriteNodeSpawnPointLocation)

; Change to the default location of your choice...preferably in the cell you'll be
using the linked lists in.
; Behind the statue of Wootamootazootagoota The Tense, or whatever.

        Gvar.NodeSpawnPoint = self
    endif
EndEvent

ObjectReference Property MyFavoriteSpawnPointLocation Auto
GlobalVariableScript Property Gvar Auto
```

Note that you only need one NodeSpawnPoint, but can have multiple controller objects.

Here is a video, demonstrating a very simple use of Linked Lists: To simulate a 2-dimensional array of objectreferences.