## TUESDAY, OCTOBER 13, 2009

### The Main Loop as a Graph Traversal

In an [earlier post](http://replicaisland.blogspot.de/2009/10/2009/09/aggregate-objects-via-components.html), I talked about treating game objects as a list of game components, with each component implementing a specific feature of the object. That way, I can call GameObject.update() and get a different result for each game object instance depending on which components have been inserted.  
  
In fact, in Replica Island I apply that same idea to the entire game loop. The whole simulation step can be completed with a single call to MainLoop.update(). Though it's not a game object, the MainLoop object is similar to GameObject it that it simply contains a list of things to update. Those things might be GameObjects, or they might be some other kind of object that can be updated--the main loop doesn't have to know. Things that are inserted into that list will be polled every frame; that's all the MainLoop object does.  
  
For example, there's a bit of code that needs to work out the current state of all the hardware inputs--touch screen, trackball, keyboard, d-pad, etc. That code needs to run every frame, but it's not a game object. As long as it has the right base class and implements update(), it just needs to be added to the MainLoop's list to be run every frame. Then there's the renderer; once all the game objects are finished this object needs to send the draw commands for the frame to the rendering thread (also described in a [previous post](http://replicaisland.blogspot.de/2009/10/2009/10/rendering-with-two-threads.html)), so it too can be inserted at the end of the MainLoop's list.  
  
And in fact, systems that need to update every frame can themselves manage other systems that need to update every frame.  
  
For example, game objects can be added directly to the MainLoop's list, but with one or two exceptions I never do that. Instead, a special system, called the GameObjectManager, is run by MainLoop every frame, and that object contains a list of GameObjects which it runs in its update() function. The reason that I have inserted this GameObjectManager class is that not all game objects should be updated every frame. Only a subset--the set that are within some radius from the camera--should be run. The rest should remain inactive to save CPU cycles. So the GameObjectManager, when updated by MainLoop, selects a subset of the GameObjects that it controls and updates them based on the position of the camera.  
  
If you hadn't guessed already, the structure I am describing here is a tree. The MainLoop is the root node of this tree, and its children are things like the input system, render system, and GameObjectManager--bits of code that need to run every frame. The GameObjectManager has all of the game objects as its children, but it stipulates that not all of its children will be visited every traversal. The game objects themselves contain game components as their children; the game components are the leaf nodes of this kind of tree. So, to run the simulation step for the current frame, I just traverse the tree.  
  
Actually, to be precise the structure that I am describing is a graph. The reason it must be a graph rather than a tree is that the structure allows for instancing of subgraphs and of cross-level traversals. For example, certain types of GameComponents that don't need to track state from frame to frame can be shared across multiple game objects; in that case, only one instance of the component exists but it is inserted into a number of different game object instances. In graph terms, shared GameComponents are nodes with multiple parents. However, for the general case the structure behaves like a tree, and so it's pretty safe to think about it that way.  
  
I like using a graph to describe all the work that must be done in a frame because it's an extremely flexible way to set up a main loop. The MainLoop object hasn't changed since I originally wrote it; though the number of objects that it contains has increased, the management code itself has remained the same. For the next game, I can rip out the individual systems that I don't need any longer and insert new ones without altering any of the main program architecture.  
  
This type of graph structure can also give you precise control over how your simulation step is run. Say you want to pause the game, but you need key systems (such as the renderer) to continue operating so that you can run the pause UI graphics. With a tree or graph system, you can insert "pausable" nodes into the tree and append to them children that should stop when the game is paused. At runtime these nodes will simply not traverse their children if the game is paused. This kind of control is hard to thread into a game that is already up and running using traditional hard-coded methods; it usually results in a lot of switches on the g\_paused variable littered throughout the code base. With a graph, none of the actual simulation code needs to change--only the graph structure is modified to accommodate pausing.  
  
Another advantage is that it's pretty easy to drive this sort of system with data. Though I haven't done this in Replica Island yet, on previous games I've worked with systems in which the entire runtime for the game is loaded from a file in the form of a main loop graph; you can see how such a structure would be pretty easy to describe in XML, and you could even use Java's reflective properties to automatically instantiate the various systems that live in the tree. Once the graph is described in data, you can change it easily from game to game, or even from level to level if necessary, all with general-purpose infrastructure code. I've not done that with Replica Island yet, but I will eventually--probably after the game ships.   
  
Game graphs are not specific to Android, but I use them a lot and I find them a pretty powerful (and generally underrated) pattern for managing real-time update loops. Like the GameComponent system, they leave the door open to future revision by separating data structures from code. This kind of system is also pretty simple to write (my entire graph is based on two core classes, a node and a group node). Of course, for small projects they are probably overkill--it is likely faster and less error prone to just write a traditional main loop and update the code every time you need to change something. But for medium or large projects, or projects based on a codebase that is intended to be reusable across many different titles, game graphs are a pretty neat way to structure your frame.

POSTED BY [CHRIS](http://www.blogger.com/profile/16079744366139636224" \o "author profile)AT [6:10 AM](http://replicaisland.blogspot.de/2009/10/main-loop-as-graph-traversal.html" \o "permanent link)

LABELS: [CODE DESIGN](http://replicaisland.blogspot.de/search/label/code%20design)

#### 13 COMMENTS:

1. http://img2.blogblog.com/img/b36-rounded.png

**[Folkert](http://www.blogger.com/profile/07666134146175312082)**[October 14, 2009 at 10:48 AM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1255542519693" \l "c5937148225009523390)

*This comment has been removed by the author.*

[Reply](javascript:;" \t "_self)

1. http://img1.blogblog.com/img/anon36.png

**Anonymous**[October 14, 2009 at 1:11 PM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1255551119304" \l "c4951985885739459080)

Can you show us the data structure you are using for the nodes?

[Reply](javascript:;" \t "_self)

**[Error323](http://www.blogger.com/profile/07666134146175312082)**[October 15, 2009 at 8:41 AM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1255621263743" \l "c5355421782204912298)

Hey Chris,  
  
Awesome I/O talk and blog! About the gamegraph, how do you handle communication between e.g. the InputSystem and your GamePlayerObject? I noticed from the I/O graph that it has a Movement component, but I don't see how its connected to the InputSystem.  
  
Keep it up!

[Reply](javascript:;)

1. http://img2.blogblog.com/img/b36-rounded.png

**[Chris](http://www.blogger.com/profile/16079744366139636224)**[October 15, 2009 at 9:55 AM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1255625714777" \l "c3067555135782098526)

> Anonymous  
  
I tried to post my BaseClass object, but it gets pretty munged in HTML format. It's very simple though--it just implements an empty update() function which takes a BaseClass parent and the current time delta from the previous frame as its argument. It's a basic node in the context of the graph. There's another, slightly longer (but still uncomplicated) class called ObjectManager which derives from BaseObject and implements an update() that walks a list of BaseObjects and calls update() on them. There are a few more variations as well--GameObjects are ObjectManagers with only GameComponents in them, the GameObjectManager is an ObjectManager that sorts and filters its child list before calling update(), and there is also a PhasedObjectManager which sorts its children before updating them. The core constructs are simple, but putting them together (and figuring out when to split a complex object up into multiple objects) is the hard part.  
  
> Error323  
  
This is a tricky area. My goal, as I mentioned earlier, is to reduce the number of cross-code dependencies. I also want to make sure that the structure of the code doesn't directly control the structure of runtime memory.  
  
I have a separate construct called the SystemRegistry which contains globally-accessible singleton systems. These systems might also be in the game graph if they need polling, or not if they don't (input needs to update every frame, so it's in there; background collision only runs when a GameComponent sends it a query, so it's not in the graph). The SystemRegistry itself is a singleton, but the systems it contains are not--I stipulate that the lifetime and exact class of those systems may fluctuate at runtime. But generally, when a component needs to talk to a high-level system like input, it looks at its SystemRegistry, finds the input system, and calls it directly if it exists.

[Reply](javascript:;" \t "_self)

1. http://img2.blogblog.com/img/b36-rounded.png

**[Ng Chee Wei](http://www.blogger.com/profile/18242275062065353050)**[October 15, 2009 at 7:10 PM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1255659029568" \l "c7403684183358887093)

I'm from an application development background with no game development experience before ... I'm currently working on an Android RPG game called Hexagon (still in POC, most likely will stick with 2D implementation) ...   
  
I have this idea of using SQLite for querying and storing the game world objects (coordinate, map, state, inventory etc). However it seems to be too much latency for my game loading (I'm using a non-OpenGL implementation now).  
  
I knew about using OpenGL to simulate a 2D scene and gain performance from your post. But I haven't try it yet.  
  
May I know from your game dev experience, do you use a DB in your game logic ever? how feasible to use it with acceptable game performance; or is it advisable to do so?  
  
by Avatar Ng

[Reply](javascript:;)

1. http://img2.blogblog.com/img/b36-rounded.png

**[Chris](http://www.blogger.com/profile/16079744366139636224)**[October 15, 2009 at 8:32 PM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1255663939664" \l "c4444426127788115163)

> Ng Chee Wei  
  
The questions you need to answer are:  
  
- What's the best framerate for my game. For a top-down Japan-style RPG you can probably get away with a sub-30 fps game without playability damage. For an action game you want to be as close to 60 fps as possible.  
  
- Given that frame rate, how much time can I spend per frame? See the post about rendering in two threads for some timing info. 60fps = entire frame in 16 ms.  
  
- How many database queries do I need to run per frame, and how long does each take? How's that going to fit into my 16 ms (or 32 ms or whatever) along with all my drawing and everything else?  
  
I think that querying a db is probably fairly slow (well, not slow, but not like 1ms or something). In the case where you need to query infrequently (say, only once every 20 frames or something), that might be fine. If you can further split it out into another thread so that the main game simulation continues to run (albeit at a slower rate), that's a potential solution too.  
  
If you are going to query more frequently, you need to know exactly how long that query is going to take, and exactly how much of your frame time budget that is going to eat. Test those things (it's pretty easy--just time some sample queries) and then make a decision. Same goes for any system. If you are deciding to draw with OpenGL or Canvas, my question is: do you need to redraw every frame? A color match game, for example, probably doesn't. A side-scroller like Replica Island does. If not, you can get away with a slower draw time. Test it and find out what you need before you start coding.

[Reply](javascript:;" \t "_self)

1. http://img2.blogblog.com/img/b36-rounded.png

**[Ng Chee Wei](http://www.blogger.com/profile/18242275062065353050)**[October 16, 2009 at 12:19 AM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1255677599932" \l "c2394514079056752557)

Chris,  
  
Thanks for your great input.   
  
Regards,  
Avatar Ng

[Reply](javascript:;)

**[Error323](http://www.blogger.com/profile/07666134146175312082)**[October 16, 2009 at 5:56 AM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1255697819205" \l "c7951233447193802557)

Hey Chris,  
  
Thanks for your answers, I have another set of questions since I'm fairly new to game design and threading. My mind still is stuck on a few things:  
  
\* Is every node within the graph part of the same thread?  
  
if yes: Then what is exactly updated within InputSystem as its 'touched' by the traverse since its input driven? And what does the Render component do?  
  
if no: How does the traversing work exactly? Where do the threads reside within the graph Input, Render, Rest? In this case I really need to wait for the code I think (or a snippet).  
  
\* Can you post a visualization of the graph showing parent child relations with arrows and every other relation with dashed arrows (e.g. InputSystem -> SystemRegistry -> Camera) and finally the threads (if any) using colored background areas.  
  
  
I'm sorry, I know I'm asking a lot, but it would really help me out. Given that, I hope I'm not the only noob wanting to know.

[Reply](javascript:;" \t "_self)

1. http://img2.blogblog.com/img/b36-rounded.png

**[Chris](http://www.blogger.com/profile/16079744366139636224)**[October 16, 2009 at 6:23 AM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1255699431993" \l "c2596322376426032594)

> Error323  
  
Sorry it's confusing. I should put together a real diagram image for this post. In the mean time, I'll just explain.  
  
The game graph is owned by a single thread. Traversal happens in that thread alone. However, other threads may communicate with things in the graph.  
  
For example, the input system receives input event from the main UI thread--the Activity that is running the game. At certain points (the beginning of the frame) all recent input events are passed from the UI thread to the game thread. When the input system is run by the game graph traversal, it uses this event information to figure out the current state of the inputs.  
  
The render system is another example. Render components, which are part of game objects, send draw commands (actually just an instance of a "drawable" object--texture, mesh, whatever) to the render system every frame. Both the components and the render system itself are part of the same thread, and both are part of the game graph (the render system is one of the last things to be visited by the traversal). The render system takes all the commands it's received throughout the frame and sends them to a different thread, the render thread.   
  
The system registry is basically orthogonal to the game graph. It's just a centralized location for useful snippets of code and utilities. Each node in the game graph has access to the system registry, so you can think of the graph and the registry as two separate but related systems. The tricky part is that certain systems, like the input and render systems, may be contained by both the registry and the game graph.

[Reply](javascript:;" \t "_self)

**[Error323](http://www.blogger.com/profile/07666134146175312082)**[October 16, 2009 at 7:02 AM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1255701755468" \l "c4402535152007322332)

Ahhh! This certainly clears things up for me, thanks!

[Reply](javascript:;" \t "_self)

1. http://img2.blogblog.com/img/b36-rounded.png

**[Matt Fowles](http://www.blogger.com/profile/08369073330803992975)**[October 30, 2009 at 10:09 AM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1256922566201" \l "c2293679313867763975)

You should consider using Syntax Highlighter for posting code. It will avoid most of the common mangling issues and every highlights it for you.  
  
http://alexgorbatchev.com/wiki/SyntaxHighlighter

[Reply](javascript:;)

1. http://img2.blogblog.com/img/b36-rounded.png

[**devcr**](http://www.blogger.com/profile/13679064054541277221)[October 30, 2009 at 2:15 PM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1256937335326#c8733292238632180507)

Thanks Chris, it's awesome, have you a twitter account?

[Reply](javascript:;)

1. http://img2.blogblog.com/img/b36-rounded.png

[**Chris**](http://www.blogger.com/profile/16079744366139636224)[October 30, 2009 at 11:27 PM](http://replicaisland.blogspot.com/2009/10/main-loop-as-graph-traversal.html?showComment=1256970474709#c1191245137455777768)

>Matt  
  
I actually have a syntax highlighter. You can see it at work in some of the other posts that include code. It doesn't work in the comments section, unfortunately, because blogger is pretty strict about tag stripping.  
  
>devcr  
  
Thanks! I don't have a twitter account--that's a system that I don't get. Maybe I am too old.