This is not an ªofficialº release because you cannot actually use it safely. To conserve space, *it does **not** include everything* in the regular release¼so don't delete your last version! The previous release is still on the ftp site and you should use it in any projects since it is tested and functional. What I've added shouldn't break anything (yet) since I'm using new names for the objects¼if you are using the old ones they are still here. This release revamps the animation section and adds several new objects. Everything compiles, but the objects are 100% untested and some are slightly incomplete (about 80% of the work is done not counting debugging) and will get a bit more tweaking. Some of the NX_Invaders code is

included to show you how the new animation stuff is used, but it is only about 50% complete. At least it gives an idea of what is to come. That is why this release exists; it will give you a chance to see where I'm headed since the next tested and ªrealº release will be at least a month away yet, since I'm going on vacation amongst other things.

Here's some late night ramblings¼basically the new stuff works/will work like this (look at the code for details; the comments are pretty good): The GameView is being split into a GKGameControl (the main state machine) and multiple GKStage objects, each with an associated GKGameView. A stage is a stack of buffers with a GKGameView on top of it and DirtPiles between the buffers. You add GKActors (your basic sprite) to a stage at the desired buffering level and then turn them loose. Each actor can move itself based upon how you subclass it to move. It has basic rendering built in and in many cases you won't even need to override it at allÐjust set a few parameters

and it all just works. Between the included GKTextActor (which will do what the text messages in PacMan doÐappear momentarily without moving and then go away) and the subclasses in NX_Invaders, you ought to get the idea of how easy this can be. The GKActorManager acts as a broker for ªout of workº actors, so that you don't have to free and alloc actors so much. Avoiding malloc() like this ought to allow a game to create and destroy sprites arbitrarily without much of a performance hit. The part that missing in all this is the GKGameControl object, which is unwritten at the moment. It will basically encapsulate the animation and timed/entry handling that used to be in the GameView.

In the case of actors colliding, you register a GKCollisionGroup with the GKStage. Notification of collisions are then automatically sent to the GKActors which collided as well as a delegate at the right times. The GKCollisionGroup describes which actors need to be checked against which other actors. This is done rather than checking all actors against each other because that would be an $O(n^2)$ algorithm and in most cases, only a few specific collisions are ever of interest. So it's up to you to tell the GameKit which collisions you want checked. Each actor can tell the collision machinery it's collision shape so

that it can be checked dynamically against the other actors in the GKCollisionGroup. Supported shapes are rectangle, triangle (arbitrary type/rotation), circle, and composite (ªorº operation) of any supported shapes. Not all possible types of intersection tests are implemented yet, and some are still a bit inefficient, but most (except the composites) are there. You can add new types of shapes in a subclass of GKCollider if these shapes are insufficient. I chose to implement these instead of generic polygons or arbitrary PostScript paths since they can be optimized via assumptions that can be made about each shape. Look at the mess of code in GKCollider and you'll see what I mean. (It would be nice in the future to allow arbitrary shapes, as drawn by PostScript paths, to be intersected. I don't think that would be too difficult to implement. This would be awfully slow, but with the upcoming PA-RISC port, at least some machines will want something to keep them busy¼)

Most of the above changes affect the ±autoUpdate method that is in the GameView. Look at PacManView's ±autoUpdate method and you'll see why I'd want to do something about this¼ now that method splits into 1) generic logic (GKGameControl) and 2) collision detection (spread throughout the various GKActor subclasses and the GKCollider/GKCollisionGroup objects) with the GKGameView only handling user events (forwarded on

to the forthcoming GKPlayer). You may also notice that the ±updateSelf:: of GameView is being replaced by the GKStage in such a way that in the majority of cases you won't have to write any rendering code at all!

One question: since this new architecture will easily support multiple GameViews (thinking ahead to things like xpilot where you have the play field and a thumbnail map view of everything; both would be GameViews) and each GameView needs a GKStage to manage it, there is a minor problem. This is as follows: Currently, each actor can only be on one stage at any given time. This means that in the above situation you either need ªshadowº actors or you need to hack them so that they can be two places at once. This problem happens becase right now a GKActor is (1) an agent in the game and (2) a sprite/rendering object. As a solution, I'm leaning toward splitting the object so that you have a GKActor which only does the agent stuff and is in ªagent spaceº, which each GameView is a window onto and then create a new object called GKSprite which an actor can have many ofÐand then the actor uses the sprite which is associated with a given GKGameView when asked to render itself. This way an actor can change it's rendering based upon where it is drawing. The only problem I have with this is that it adds yet another level of indirection to the kit, making things a

bit slower. On the other hand, it adds a lot more flexibility and it also will make it easier to write your own GKActor subclasses. It also seems to make better sense conceptually, since some actors are invisible agents which only serve to impose control over multiple actors on the stage. (ie. choreograph the motion of multiple actors simultaneously¼) What are your thoughts on this? Think about how you'd use the GameKit and intersect that with some of these design choices; which sort of an architecture do you think would better suit your application? (Feedback in any of the other kit areas is welcome as well. I mention this example explicitly because now is definitely the time to raise your voice if you want to be heard; once I commit to something, changes will be more difficult!)

Well, that's very brief, and I plan to document this fully and give examples, etc. to be placed in the next release. As well as test things. NX_Invaders is my test bench, so once they work, you'll have a new game to play, too! :-) Well, I'm headed home¼if you absolutely need to reach me between Aug. 14th and Aug. 28th, I'll be in the Chicago area at (708)392-7672 and without net access¼

Later,

DONALD

Y A C K T M A N

Here's the stuff from the regular README file:

First, here's how to install the stuff: (see notes in gamekit-1 for more info on installation and available options)

Put the gamekit-1/built-fat/libgamekit.a file into /usr/local/lib.

Copy the directory Headers/gamekit to /LocalDeveloper/Headers.

You will also need the CCRMA music kit installed on your system. I am currently using the Makefile supplied with the CCRMA Music Kit source distribution as the makefile for the gamekit, since that was the path of least resistance. I haven't set up the install target properly, though. To simplify things, I have compiled everything already so it should "just work" as is.

Well, this could be considered release 0.00. That's as low as I can go without going negative. Most of what's here works pretty well, but is nowhere near complete nor is it perfect. Much of what is described in the Concepts.rtf documentation is still missing, as you can easily see. I *will* be adding everything

that's there over time, however, and making frequent releases as the functionality improves and the bugs leave.

What's here will be useful to some, and very lacking for others. The best thing to do is to bug me about features that you need fixed or implemented. The areas most frequently requested will, of course, receive more (and/or faster) attention. If there's something you'd like added, let me know. If you find a bug, let me know. If you think anything at all about this, let me know.

If you want examples of how to use this stuff, right now there

are two ways to go. (1) Look at PacMan. It now uses gamekit objects and serves as a jumping off point. It's as good as the next option which is (2) become a registered user of Columns and then ask me for the source. Columns and PacMan are *right now* using these exact objects! They have a few subclasses of key gamekit objects and everything else is made up of stock gamekit objects, so they ought to be excellent examples! (If you're already a registered user of Columns, just ask me to NeXTmail you the current source files...) One big problem right now is that I haven't had time to include all the template .nibs and update the ones that are here. Without those, you'll

probably have a hard time figuring out how this all hooks together. The latest Columns or PacMan, in this case, can be very helpful. In fact, once I get them debugged to my satisfaction, their .nibs will be trimmed back to provide the templates¼ you can get the Columns (or PacMan) .nibs right from the beta binaries and munge them up, registered user or not¼and the PacMan source is in the GameKit release now.

Again, bug me about any questions and/or problems you have! Answers that would be interesting to anyone on the gamekit list will be posted there.

1

Share and enjoy!

±Don_Yacktman@byu.edu

[1]Douglas Adams, The Hitchhiker's Guide to the Galaxy. :-)