February 11, 2013

Agenda:

- Article of the Week – *Stop Trying to Learn Everything Before Starting*
- Game Design Document Discussion
- Game Design Document and Revised Timelines Due – **February 17**
- Dues
- Groups

Article of the Week

Stop Trying to Learn Everything Before Starting *by Chris DeLeon*

http://www.hobbygamedev.com/beg/stop-trying-to-learn-everything-before-starting/

Don't wait until you know everything there is to know before starting.

Learn just enough to get started. That includes having a rough idea of a realistic scope for a first project, so that you don't wind up lost on a fool's errand. (Tip: try 1970's.)

Then get started. Get your development environment set up and compiling empty/test/placeholder/example code. Just get anything on the screen that runs, meaning a .exe if your programming for Windows, a .swf if you're making a web game in AS3, etc.

At this point, you're already ahead of an untold number of people that have only ever thought about videogame development, but have been too caught up in waiting for the perfect idea, or trying hopelessly to fill their brains with everything ever written and said about it before actually getting started. So far so good.

Find a way in your programming language or environment to load an image file and get it on the screen, to respond to keyboard and mouse or whatever input's needed, to load and play back sound effect files and looping music. Depending on the platform, picking a library may be helpful for this (ex. SFML/SDL/Allegro/XNA, if in C/C++), or this may be functionality built into what you're working with (Unity). Often the easiest way to get to this point is to just find some simple example code that

already does these things, then twiddle settings in your development environment until you can compile and run it as expected.

Does the player character need to move like a truck? A spaceship? A tank? Mario? Get the input to move the player's graphic (probably an unanimated rough draft of the graphic at this point, concerned only with basic appearance and scale on screen) moving in the way that it ought to move.

Get the level structure put together, to position visuals and handle basic collision against the player. Save/load the level structure in some practical format – never mind getting caught up on the theoretically optimal compression of that data, as level files are virtually always tiny, and if that becomes a problem later, cross that bridge when you get there. Level files initially saved in ASCII can make debugging significantly easier anyhow. Is the game world based on freestanding obstacles? Grid-based tile collision? If the camera needs to pan, add some offsets to the player draw position and level draw origin to achieve scrolling, or a global transform if you're in 3D or using hardware acceleration, and update those offsets to move the view based on player position.

Add enemies (if needed), items (if needed), trigger puzzle elements (if needed), special powers (if needed), ammo limits (if needed). Add nothing that isn't needed. If not sure what else the basic engine might need – and I'm only using the word engine here only in the most minimal sense, meaning the game's core code, not the try-to-support-every-game-imaginable Titanic-seeking-an-iceberg undertaking – start putting together playable level content and see what additional features or tool/process improvements you find yourself wanting while doing that.

The game may not need anything else.

If and when a situation arises for which you're not sure how to proceed without digging and experimenting for solutions, that's fantastic, welcome to real programming.

If and when a situation arises for which you have to make a tradeoff between burning an uncertain amount of time on figuring out something tricky, or working out how to cut that feature from the design without the game completely falling apart, super cool, and welcome to production.

Once you finish the game and people don't feel about it the way you hoped they would, first off: that's awesome, because hey, you finished a videogame. Congratulations! Next time you'll be positioned to make incrementally more informed tradeoffs with consideration for implementation realities, production compromises, and potential impact on user experience. It's only at this point that someone

is beginning to really do videogame design, design of a full videogame, as opposed to talking about videogames, reading about programming, or cloning someone else's work as an exercise. Note too that design of a full videogame is different than level design, which though the two are sometimes conflated since often done by the same people, when done in isolation from other issues and options about a game's development level design is another field of content creation, another skilled technician's craft like animation, dialog writing, or sound editing. Those are hard things to do, deserving of respect and worth doing well, but they are not videogame design. Designing a videogame is different than designing for a videogame.

Learning is great. Books are important. Some amount of learning does need to happen before starting, all I mean to deal with here is that that amount of learning needed tends to be far less than people seem to assume. Learning, for this type of material, has to be situated in a context, demonstrable, useable, and practical. Contrary to the lay consumer impression that making something is a mere variation on being able to judge something, a maker has a fundamentally different set of concerns and practices than a critic.

Roger Ebert is brilliant, but isn't suited to making a decent film. (Not hypothetical.)

If your goal is to be a critic: spend all day studying the form and discussing it. If your goal is to be a maker: start making things, keep making things, and finish making things. Being reflective on practice can be helpful, but that requires actual practice to be reflective about.

Learning everything there is to know about programming or videogame development before starting is impossible. It simply won't – can't – all fit inside one head at the same time. Even if that information somehow could be learned, memorized in the abstract, detached, and rote sense, it would not be of any use without experience working through real problems with it.

This is not a decision between learning versus doing. What I am proposing is that, within this context, learning without doing isn't really learning. Meanwhile doing without learning is impossible, because doing will demand learning.

If you make some wrong assumptions early on, they're often easily corrected, that's the nature of digital stuff. We're not laying railroads or building skyscrapers, we're not cutting someone open while they're under anesthetic, we are dealing in digital text and other easily modified, easily backed-up files. The rework time is nothing compared to the vast but invisible damage from never starting, waiting an infinite amount of time until everything knowable is known. That rework is even when the

best learning happens, because the desire to not lose that amount time on the next attempt will help lead a deeper understanding of what caused it, so that it might later be avoided.

Be wary of excessive preparation serving as a disguise for procrastination.