Kurt Daill

Final Report: Jam Study

Speaking personally, this has been an incredibly valuable study in terms of data and experience. There are a lot of design or development lessons I have either learned or revisited and seen the practical application of. It is somewhat strange to be at the end of an "independent study" with entirely qualitative data, but some design lessons and experience was really the objective of the study. I even think a lot of my notes show a pattern that I would like to refine more, and also can put together into something of a framework I'll be taking with me to future projects. But before any of that, I think it would be worthwhile to review each of the individual projects that made up this study.

I'm perversely proud to say that *March of the Northmen* feels like it came from a different universe than the later projects. It was an odd, unfocused mess that was more designed around some theoretical larger game than any kind of core engagement. The core mechanic of beating a virtual drum to command an army is strong, but I'm now more aware than any attempt to revive that has to start with that drum and that army being engaging to interact with second by second, and then built from there.

Doppler was, in retrospect, the breakthrough that has informed the rest of development. The three hour time limit forced better design principles out of me. There is a difference between just hearing "focus on the primary gameplay loop", and building a game where that is all you have time for. It definitely was an influence in terms of the technical side of development throughout the rest of the project.

Nightrun was essentially another layer on top of the lessons picked up in *Doppler*. There was a core in the form of the platforming mechanics, and then the secondary loop of trying to get through as much as possible in one night. Ultimately I think it was also a lesson in managing time as I definitely came up a few hours short of where I should have been, and the controls did not get the final bits of polish they really should have. But a finished game is a finished game.

Call Backs was probably the largest leap in terms of design, and was definitely more valuable as a lesson than a finished project. I had, in many ways, failed to realize how the lessons from earlier applied to design as well as nuts and bolts technical development. So the design went untested, and ultimately didn't pan out. But it did provide the final hints that have led me to something of a framework I think I'll be applying to future development projects.

As this project draws to a close, I think I've had lessons that fit together into a coherent picture. Broadly it has been "development is chaos"; the largest source of that chaos being the gap between a project's design and its implementation in practice. In the broad strokes, game development is an additive art; one adds the art, sound, and code together to create something from a blank slate. But I'd argue that design is more of a subtractive art: the designer begins with an open possibility space, and each subsequent decision locks them into a more and more specific design that can't be changed without reversing those decisions. When creating assets in accordance with that design, reversing those decisions becomes a matter of throwing away a portion of the limited time one has to finish their game. It's like building a house out of concrete; you can go back and forth on ideas on blueprints, but once you start pouring any changes are going to need a wrecking ball. Given that, there are three broad considerations I know I will be

taking forward into the future. The developer must be mindful of their design's risks, solutions, and opportunities.

Risks are the reason projects could or do fail to ship, and are inherent to any new design. The developer does not know if a new mechanic is actually fun or resonant, or if they can properly code this new element, or if the art they are working on will turn out. If enough of those things go wrong, it is game over. Risks are first in this hierarchy because the time that they need to be addressed ranges from initial designs all the way to the end of development. One avoids those risks turning into massive problems by prototyping and testing. You build the smallest, fastest, dirtiest form of the thing and test it. You take that test data, refine the thing, test it again. I am two for four in terms of letting a risk go untested and having it turn out to be fatal once the project was already half built, so I feel that this the takeaway for someone around my general level of experience.

The next tier is solutions. Once you start fixing things in the testing phase, the quality of that solution has further impacts. Does that solution require extra development time that wasn't originally accounted for? Does it put a constraint on the creative side of the project (maybe it requires some lampshade hanging in the narrative)? Is it just a spot fix, or can you solve multiple problems with one elegant system? Most of my solutions across this project weren't sophisticated, but they satisfied the conditions of "this has to be done quickly" and "functional is more important than refined". *Nightrun* for example, had its original physics system discarded for the one packaged with unity, had a bunch of math related to the dash command replaced with a single line of script for reliability, and the 'keep building the project until the mysterious bug goes away' panic I detailed in that report rather than actually addressing the problem in the scirpt

or editor. Solutions are not being put together to make you look clever at GDC, they are there to make your project work.

The last I have found is opportunities. Not every unexpected thing in development is necessarily a disaster in the making. There are likely a few diamonds lurking in the darkness between your design and the realities of the game you're making. Being willing to take a page from jazz soloists and improvise a little when the opportunity presents itself can do wonders. Even in the time-crunched trenches of the jams I managed to find a few things, whether that be the accidental audio in *Doppler* or the tutorial text in *Nightrun*.

To summarize, development is about managing and exploiting chaos. If there is a broad "what I learned in my independent study is" moral at the end of the story, that's it. In a sense, game developers are of a kind with the wizards of folklore. We have nothing but some arcane language and whatever scattered knowledge available to draw something from the void, and then have to muster those skills again to contain the resulting chaos. If I have learned nothing else, it is that I will never be done learning how to do it better.