Doppler was created for the 59th Trijam, and in accordance with the rules was constructed within a three hour time limit. The development of *Doppler* clocked in at around 2:45, given that trying to make new builds with 15 minutes on the clock is a bad idea. Being a very short and productive development cycle, I managed to come out of the experience with a very short and productive lesson.

In *The Art of Game Design* Jesse Schell organizes his thoughts on game design into a chart that grows each chapter, and shows the relationships and flow between concepts. The centerline of this entire graph is this simple relationship: Designer->Game->Experience. The game itself is not the end of the process, it's merely a vehicle for an experience. Every art asset, piece of technology, or clever mechanic is just an elaborate trick to put an experience in the mind of the player.

I am a subscriber to the idea that good development of any kind of software comes in loops. You design, then you build, then you test, then test results lead you to a more refined design. Software development should be iterative.

Game development therefore, can be thought of as a loop built catering to that Designer->Game->Experience relationship. The developer plans for a theoretical experience, builds something she thinks will invoke that experience in an audience, then tests it to see if the artifact she built gives that experience.

Where these facts create problems for developers is in the 'building' step. The majority of time is spent building, i.e. writing code, creating art assets, solving technical problems. But testing can invalidate that work in a number of ways; aesthetics don't work or need to be changed for new technical requirements, mechanics need changes that require new scripting, etc.

The longer you stay building without sitting down, playing the game again, and asking "does the facilitate the experience" the more work you potentially waste.

Imagine creating a new enemy type. You create its animations and AI, you bug tested it's code and smoothed out all the technical issues, and only after it's implemented do you test it. On testing, you realize that it's just annoying to have to deal with or distracts from the better parts of the game. All of that technical work is wasted, because you didn't test it when you had a really primitive version of it.

I can't claim to be some great development genius in discovering this, mostly because it was a happy accident. Three hours is probably longer than I take to go through this loop when I'm development normally, so I developed as fast as possible. The order of the day was "make this an actual game as fast as possible", so I ran through the loop incredibly quickly and got more results than I'd expected. I even had more 'happy accidents' with misbehaving components actually adding to the game and being made permanent: an example being a drum beat sound effect that started playing in places that were actually quite satisfying.

Doppler is an incredibly primitive game, it boils down to "click the things to not lose". But it does a lot more than the projects already on this list to support a core experience. That experience is only "things are speeding toward me", but even then I'd wager it will stand as one of the projects with better "game feel". In summary: it's a game that took 3 hours to make, and it *works*.