What's the worst code you've seen in production? originally appeared on Quora: the place to gain and share knowledge, empowering people to learn from others and better understand the world.

Answer by Barry Holroyd, Staff Software Engineer at Quora (2017-present):

At one point in my career I joined an established game company to manage the engineering team for a web-based game. The original engineers that built it deserve a lot of credit. The game was still massively successful when I inherited it -- after three years, it was still bringing in over $80M annually. However...

The code base and deployment system were the worst I have ever seen, literally by at least an order of magnitude. For instance, although we had multiple sets of servers (production, continuous integration, testing, staging, etc.), no one knew how many there were (other than production, which had about 120 servers) or physically where they were; in addition, their configurations were nowhere near similar (different software packages on each, different versions of software, etc.). Furthermore, there were so many Perl-based deployment scripts and they were so convoluted that no one wanted to touch them. Any change risked serious undesired consequences (i.e., downtime). In desperation, I finally decided to force the issue by simply "starting somewhere" — I asked one of my senior engineers to pick one of the scripts and take the time to understand it thoroughly and then make a recommendation.

After spending the better part of a day analyzing a particular Perl script, we met to discuss his findings. He stared at me and I stared back for a minute. Finally, he said, "There are about 1,000 lines of code in this script. Of those, exactly two lines do anything at all. The rest of the lines do nothing and can be deleted."

One script down, 49 to go.

The data-driven aspect of our system was even more problematic. The concept was great -- game designers (non-engineers) could enter data into the system which would allow them to create new variants of armor, weaponry, etc., all without code changes. However, the mechanism was not the best. At the front end of the pipeline were about 50 Google Doc spreadsheets. The game designer would add/modify values to those spreadsheets. There was no data validation and Google Docs would auto-save any changes, so a simple typo (e.g., a 'w' instead of a '2') would enter the pipeline without the designer even being aware of it.

When the designer was done entering the data, they would "submit" the changes. The Google Docs would be converted to Excel spreadsheets and checked into the source code control system. Later, the back end servers would pull it out. At that point a few dozen Python scripts would start wending their way through the spreadsheets, consuming values from various cells and also getting redirected by special values in cells to other spreadsheets. Thus, there were numerous scripts crawling their way, independently and with no overriding architecture or design, through dozens of spreadsheets (Python was certainly an appropriate name for the language in use).

The output of the Python scripts took three forms, one of which was ActionScript. The ActionScript would be automatically compiled and then injected out into the wild, where it would run in thousands of browsers (the DAU of the game was about 300,000). If there was a problem with the ActionScript, it could blow up simultaneously all over the world. And tracing the problem back to its source could be extremely difficult.

Like the deployment scripts, the convoluted pipeline, along with the fact that we could tolerate very little downtime (official or otherwise), made any real architectural improvements to this pipeline challenging, to put it mildly.
At one point we tried to migrate to MySQL Cluster. After months of preparation, we finally started the conversion one night at about 10pm. By 3am the next morning the new system was up and running. There were about 15 of us in the War Room and we were all ecstatic. We decided to stay up for another hour or so to monitor the system. At about 3:30 a.m., the entire cluster crashed — went down hard. We frantically tried to analyze the problem but in the end decided to roll everything back (the rollback was successful). Subsequent analysis found the problem. Our code had generated a SQL query which was so long that, when printed out on computer paper and laid out on the floor, it measured about 15 feet in length. We did an Internet search and found one instance where someone else had a similar crash, apparently due to a bug in MySQL Cluster. I guess that few people tried to shove a 15 foot SQL query through MySQL Cluster!