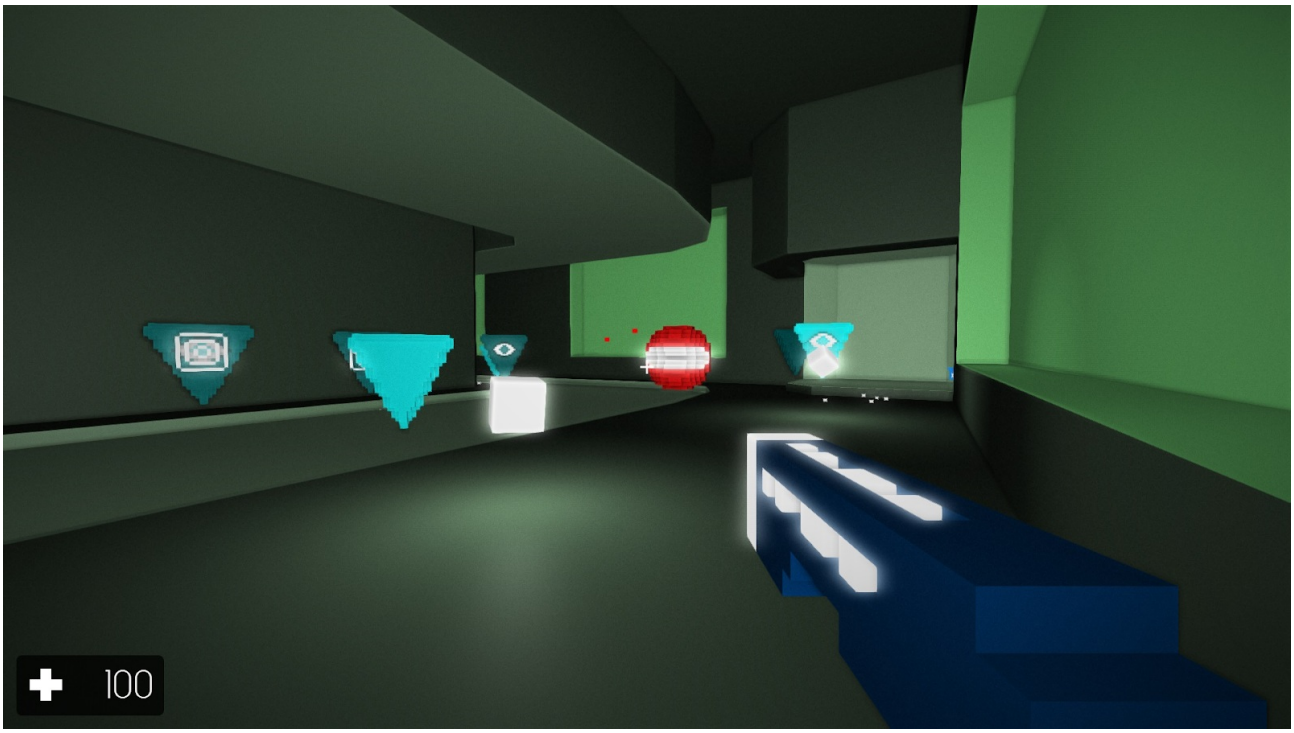
Post-Mortem: Gorescript Classic

Blogs
by Sergiu Bucur on 03/22/17 10:52:00 am Featured Post

Gorescript Classic is a fast-paced '90s-style first-person shooter built in JavaScript and released in 2014. It is the predecessor to Gorescript, the new 2017 version, a full revamp of the game, reworked from the ground up in C++ and currently on Steam Greenlight.

This article aims to explore the full history of Gorescript Classic, from the development stages to its initial release, to getting played by John Carmack himself, then headlined on HackerNews, and finally achieving lasting popularity on Chrome Web Store.



Building my own private Doom

In early 2014, I wanted to make a game. Specifically, I was messing around with some 2D stuff in three.js while relearning JavaScript the proper way. For some reason 2D never managed to keep me engaged for long periods of time, so after a few weeks I switched to 3D and started to build a shooter.

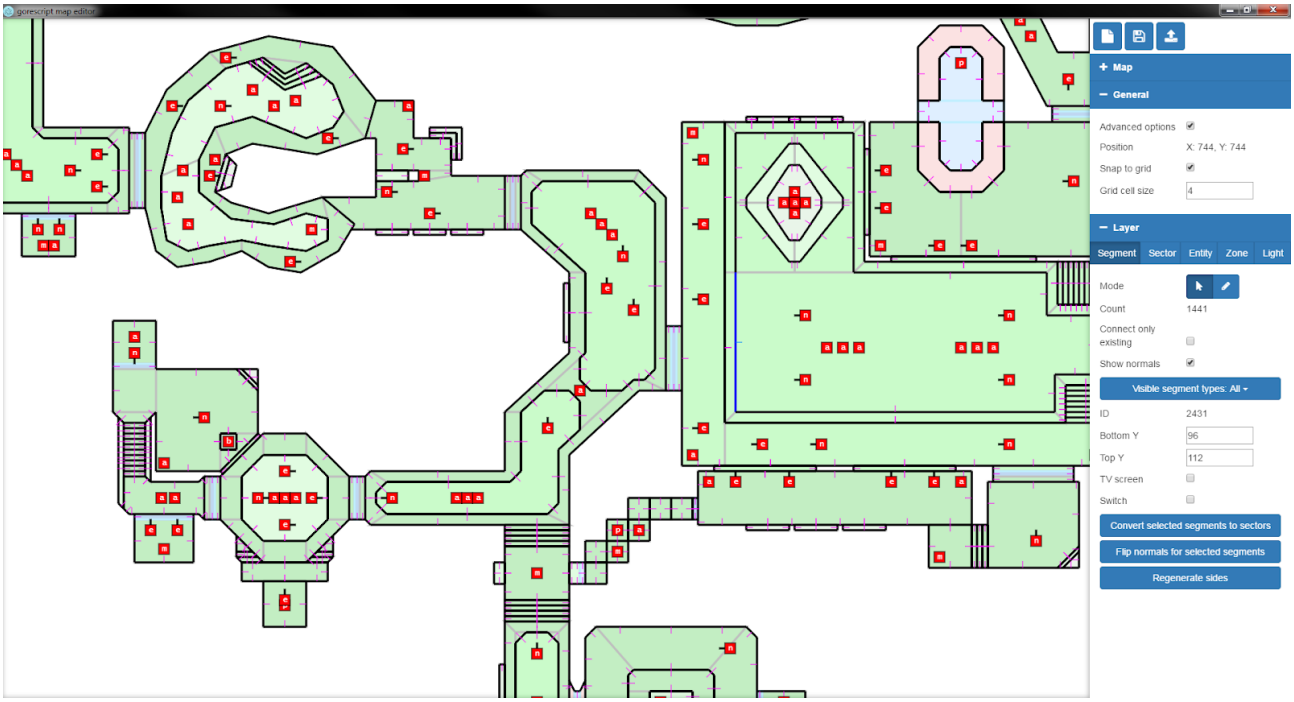
Now imagine you're in the following situation: you want to build a shooter, right? But you don't want to use an engine, nor do you want to learn Blender or something similar to make 3D assets. You also don't have any artistic skills whatsoever, and your team consists of you and you alone. What do you do? Well, you either choose a more orthodox path to pursue or you start to improvise. And improvise is what I did.

To build something that can be called a shooter you need the following things: some kind of weapons, some kind of enemies, and some kind of environment for these to interact with each other. Unless you're developing under a rock, at this point you're gonna start looking for inspiration. How did other people accomplish this?

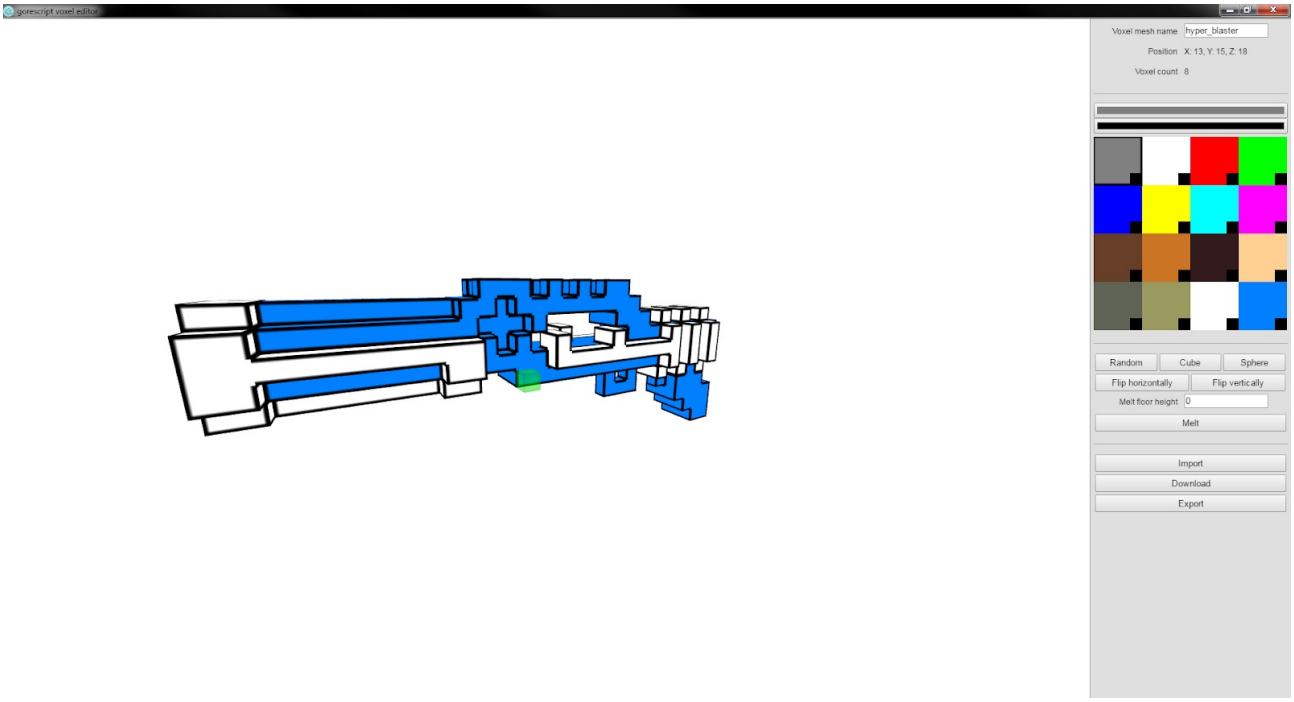
The best example I came up with is the original Doom. Doom was pretty much the top game of my childhood. I know every level like the back of my hand. I can't name more than a few Pokemon, but I can sure name every single monster "species" from Doom. The sound effects (particularly the PC speaker ones) are forever burned into my skull. I even read Masters of Doom twice. Doom was the perfect choice for me, both for personal and technical reasons.

So I started to reverse engineer Doom, not by way of reading the source code, but simply by looking at it and trying to break it apart. How does a map look like in a map editor? How does it render a map? What kind of scripting do the maps need to function? How do the weapons work? How does the monster pathfinding work? What monster types do you need?

Over a period of a few months, I built my own Doom-like content creation pipeline. First off, a map editor where you draw 2D maps which then get converted into 3D by the engine. While the engine renders them in full 3D, the levels themselves are old-school 2.5D. Because why would you want to place a room over another room in 2014?



Secondly, instead of sprites, I opted to use voxel meshes so I built a custom voxel mesh editor. Animating them the modern way was out of the question, so even though they're 3D, the enemies cycle through different meshes to walk, attack and die, in a way very similar to spritesheet animation.



Collision detection and response was another huge issue to tackle. After reading every article I could find, I finally pieced together a working version using ellipsoid to triangle tests for the player and projectile collisions and 2D AABB tests for the monsters. It's decent enough if you ignore the fact that monsters can't step over each other, even if one of them is flying 30 feet above. That's a Doom limitation which I gleefully added into my own engine, 20 years later. To tie it all together, I used a uniform grid scene graph instead of BSP trees.

AI was pretty basic: you make any kind of noise inside a room and all the monsters in that room and any adjoining area not separated by a door are instantly alerted to your presence. They don't come straight at you but they do add a random 2D vector to your position every few frames and instead head straight for that. If their movement is blocked for any reason, they "scatter" -- pick a random direction to run for a few moments before getting back to their original target. Same as in Doom, this works surprisingly well, considering they don't have any sophisticated pathfinding algorithm at their disposal.

Rendering was the most interesting part. I had to make it look decent while running reasonably fast. After a few attempts involving textured environments and lightmaps, I eventually chose against them and went for vertex-colored walls, floors and ceilings lit by point lights and shaded only by SSAO. It gets the job done, though it's not by any means a novel approach. A lot of three.js demos use this style, certain games too.

So after all that plus the usual boring to implement but necessary bits like UI, menus, rebindable keys, etc. the game engine was finally ready. For the game itself, which amounted to a demo content-wise, I made three scripted levels, two enemy types and three weapon types. And that was it. I released it for free in May 2014.

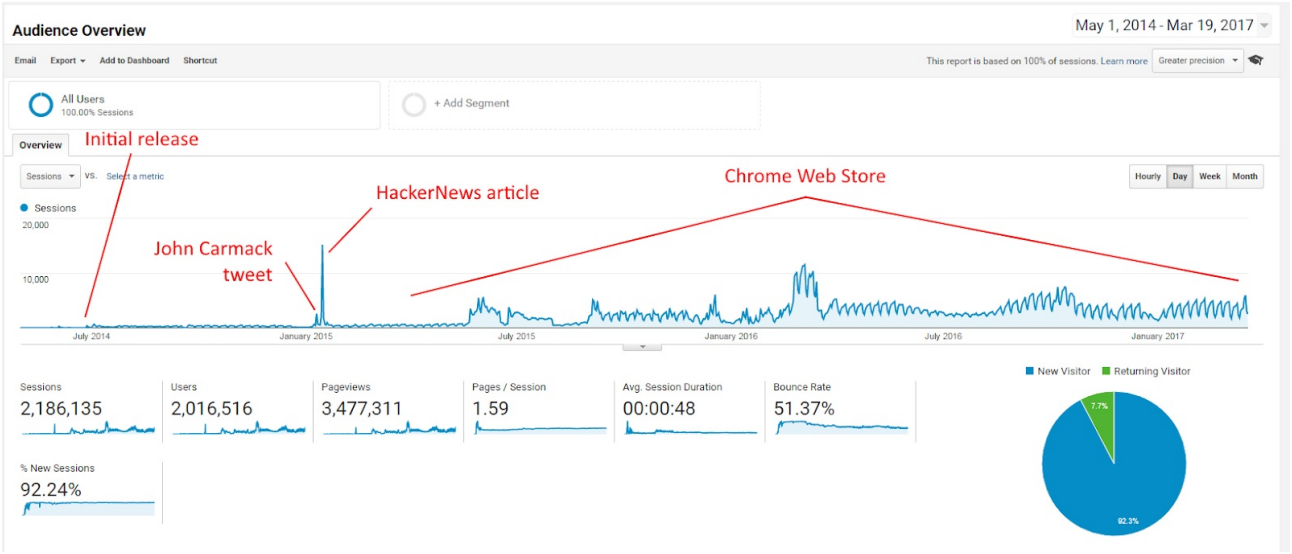
Aftermath

The initial response was lukewarm, although I probably saw it differently at the time. People seemed to like it for the most part, and it even got praised by Ricardo Cabello (aka Mr.doob), the creator of three.js. Apart from the game being showcased on threejs.org and Chrome Experiments, it pretty much went under the radar. Since I had no further plans to develop it at the time, I moved on with my life.

One of the main complaints in the first half-year since its release was performance. It was quite resource-intensive and didn't reach 60fps on most systems, so around Christmas 2014 I decided to pick it up again and try to improve the performance. Eventually, after a few weeks I implemented antiportal rendering, which proved to be a substantial boost. While not perfect, the game ran much better now on a significant slice of systems.

In January 2015, I started to promote it a bit more on Twitter andReddit. Somehow I even got the nerve to tweet it to John Carmack. To my surprise, he actually replied! He played it. It was an amazing moment for me. Soon after, HackerNews picked up the game, and its popularity soared for a while.

Come February, I got an e-mail from Google asking me to put the game on Chrome Web Store. I published it as a packaged app, which provided a few improvements over the original web version, most notably the fact that you no longer needed to click a browser dialog to enable mouse look every time you played.



The Chrome Web Store release proved to be the definitive one. It's currently at 250000+ users, with an average score of 3.5 stars out of 5. Not bad for my first ever game release.

The game is also quite popular on YouTube, getting weekly Let's Play's to this day. That's after basically no content patches since early 2015. People are still playing it. I still get e-mails about it from time to time. Overall, I'd say it was definitely worth my time to develop.

Gorescript Classic is completely free and the source code is available under the MIT license on GitHub. The new Gorescript, including a playable demo, is now on Steam Greenlight.


SQUARE ENIX


InnoGames