The Worst Language Ever Designed:
The Case for Better Programming Languages for 3D Environments

Crista Lopes (lopes@ics.uci.edu)
University of California, Irvine

In the past few years, a number of game engines (e.g. Unreal , Panda3D, among dozens), 3D scene editors (Unity3D, Blender, etc.) and programmable virtual environments (Second Life, OpenSimulator, Wonderland, etc.) have emerged, all having one thing in common: 3D objects need to be given behaviors, and that activity is to be performed by people who don't necessarily have a strong programming background – artists and game designers. In most cases, these systems are written in one language (C++, C# or Java, usually) and support end-user scripting in another (Python, JavaScript, Lua and a plethora of other DSLs). In most cases, too, the scripters aren't given full access to the game engine but, instead, access it through a more or less restrictive API that establishes what the objects can do. With the introduction of WebGL, the Web platform is now catching up with the work that has been going on for several decades in gaming. 3D is an increasingly important component of computing applications well beyond gaming (e.g. scientific visualization), and yet the challenges – both technical and social – of programming 3D scenes is unknown territory for mainstream, enterprise-focused programmers. In this position paper, I want to share my experiences with some of these 3D platforms, and I want to direct PL researchers' attention to this important application area that could use some help.

For the past 6 years, in parallel with my official research program, I have been working on an open source version of Second Life (SL), a server-side infrastructure called OpenSimulator [1]. OpenSimulator is an indie server, and as such, its use is hard to track, but we estimate several thousand independent servers exist out there; the number of downloads of each new release is in the tens of thousands. We recently held a very successful virtual conference [2] with 360 attendees, which I informally blogged about [3]. I am one of the core developers (under a pseudonym). So far, my main focus has been the server-side itself – its completeness with respect to the SL protocol, as well as a novel Federation architecture [4], assorted issues related to asset sharing, and server performance.

One of the neglected corners of OpenSimulator is its scripting engine, which supports the worst language ever designed: the Linden Scripting Language (LSL). LSL is an event-based, object-seeded, C-like language originally designed by Linden Lab (the makers of SL) that the users of these virtual environments use to add interesting behaviors to the 3D objects. The interface between LSL and the server consists of 30+ event types (raised by the server and handled by the scripts) and 400+ functions (called by the scripts to the server). The programming model is such that each individual object carries its own scripts: there is no concept of a class or type of object (aka "prefabs" in some game engines), only prototypes. There is also no concept of a reusable library that scripts can include; all reuse is done via copy-paste. Version control is non-existent, and it is hard to decouple the scripts from the 3D objects. Data types are limited (no dictionaries, for example), and the interface to the existing ones leaves much to be desired (read "horror!"). LSL is the opposite of the principle of least surprise, and it makes the most basic software engineering practices, such as testing and version control, quite difficult.

As bad as it sounds, thousands of users, many of them non-programmers, have been scripting amazing things in both SL and, now, in OpenSimulator virtual environments. It seems that a large enough engagement from a community to document a language, no matter how horrible it is, is enough for newcomers to be able to get things done! But the amount of time wasted by this community in trying to understand how LSL works, and in going around its limitations using all sorts of hacks, is certainly not something to be proud of, and we should aim for things to be greatly improved in the future.

SL/OpenSimulator are not alone in this state of affairs. Unity3D, for example, one of the most popular scene editors for multi-platform games, also suffers from several weaknesses regarding the association of behaviors with 3D objects. Unity3D provides C# and JavaScript as scripting languages, but in a relatively constrained manner: each object is given one or more behaviors each one described in a different script that, like LSL, responds to events from the scene and can call an API. Neither C# nor JavaScript are friendly to non-programmers, so the workflow in those games is such that either professional programmers also design the game or game designers need to acquire advanced programming skills, neither of which is ideal. Unity3D supports "prefabs" (a gaming word for a class/template), which is clearly an improvement over systems that don't support it. But, again, the strong coupling between 3D objects and scripts makes testing and version control quite difficult.

On the Web, the state of affairs is even worse: scene editors (clearly a good idea for these applications) are only now emerging, and they are still years behind the ones we see in the gaming arena.

3D environments are going to have an increasing importance in application development, and they present very interesting challenges for programming languages and software engineering. Let me try to summarize them: (1) these are **visual** environments, where the scenes – collections of 3D objects – are usually constructed by artists and designers whose skill sets don't include a deep understanding of programming; (2) the strong coupling between 3D objects and their behaviors is both a blessing (they're right there!) and a curse (very hard to do automated testing, version control, external processing, etc.); and (3) while programming individual objects tends to be relatively simple, there tends to be a generalized lack of support for programming collections of objects whose behavior needs to be coordinated. Overall, this niche is ripe for new ideas that will make the life of designers and developers a lot easier than what it is now.

[1] http://opensimulator.org

[2] http://conference.opensimulator.org/2013/ (Several videos of the sessions available here)

[3] http://tagide.com/blog/2013/09/the-future-of-conferences/

[4] Lopes, C. "Hypergrid: architecture and protocol for virtual world interoperability" IEEE Internet Computing 15(5) pp 22-29. September 2011.