The Goals of the Luau Type System

ANDY FRIESEN, ALAN JEFFREY, and OTHER PEOPLE?, Roblox, USA

A position paper about the goals Luau type system.

ACM Reference Format:
Andy Friesen, Alan Jeffrey, and Other People?. 2021. The Goals of the Luau Type System. In . ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Roblox [5] platform allows anyone to create shared, immersive, 3D experiences. At the time of writing, there are approximately eight million experiences available on Roblox, created by eight million developers. Roblox developers are often young, for example there are over 200 Roblox coding camps in over 65 countries listed at [4].

The Luau programming language [3] is the scripting language used by developers of Roblox experiences. Luau is derived from the Lua programming language [1], with additional capabilities, including a type inference engine.

This paper will discuss some of the goals of the Luau type system, and why those goals are slightly different from other type systems.

2 HUMAN ASPECTS

2.1 Heterogenous developer community

Quoting a 2020 report [2]:

- Adopt Me! now has over 10 billion plays and surpassed 1.6 million concurrent users in game earlier this year.
- Piggy, launched in January 2020, has close to 5 billion visits in just over six months.
- There are now 345,000 developers on the platform who are monetizing their games.

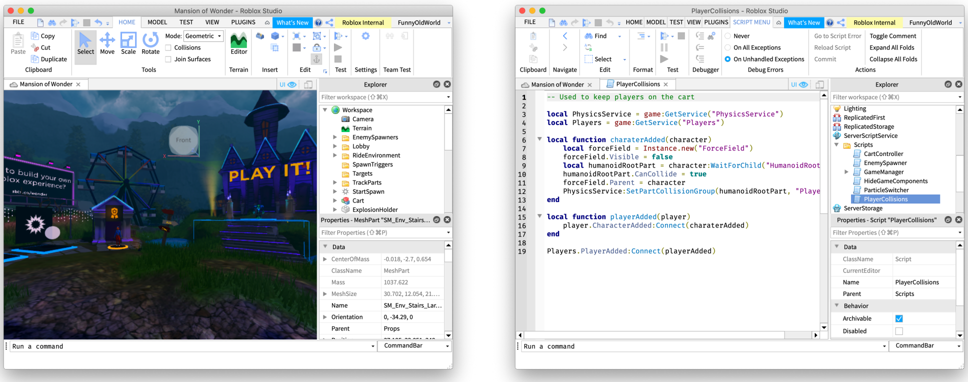
This demonstrates how heterogenous the Roblox developer community is: developers of experiences with plays measured in billions on the same platform as children first learning to code. Moreover, *both of these groups are important*, as the professional development studios bring high-quality experiences to the platform, and the beginning creators contribute to the energetic creative community.

2.2 Goal-driven learning

All developers are goal-driven, but this is especially true for learners. A learner will download Roblox Studio (the IDE) with an experience in mind, often designing an obstacle course (an “obby”) to play in with their friends.

The user experience of developing a Roblox experience is primarily a 3D interactive one, where the user designs and deploys 3D assets such as terrain, parts and joints, and provides them with

HATRA '21, October 2021, Chicago, IL
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn



```
-- Used to keep players on the cart

local PhysicsService = game:GetService("PhysicsService")
local Players = game:GetService("Players")

local function charaterAdded(character)
    local forceField = Instance.new("ForceField")
    forceField.Visible = false
    local humanoidRootPart = character:WaitForChild("HumanoidRoot
    humanoidRootPart.CanCollide = true
    forceField.Parent = character
    PhysicsService:SetPartCollisionGroup(humanoidRootPart, "Playe
end

local function playerAdded(player)
    player.CharacterAdded:Connect(charaterAdded)
end

Players.PlayerAdded:Connect(playerAdded)
```

Fig. 1. Roblox Studio's 3D environment editor, and script editor

physics attributes such as mass and orientation. The user can interact with the experience in Studio, and deploy it to a Roblox server so anyone with the Roblox app can play it.

At some point during experience design, the user of Studio has a need which can't be met by the physics engine alone. "The stairs should light up when a player walks on them" or "a firework is set off every few seconds." At this point they will discover the script editor, and the Luau programming language.

This workflow is different from many initial exposures to programming, in that by the time the user first opens the script editor, they have already built much of their creation, and have a very specific concrete aim.

This workflow suggests a Luau goal for helping the majority of creators: *support learning how to perform specific tasks* (for example through autocomplete suggestions and documentation).

2.3 Type-driven development

- Code refactoring
  - Code navigation
  - Detect detection

3 TYPES

3.1 Infallible types

Goal: support type-directed tools in all programs
  - All programs have a type (analogy with infallible parsers)
  - Used by autocomplete + goto-declaration
  - Still support red squigglies
  - Problem: stop the user being swamped by cascading errors
  - Problem: no "right" type, just heuristics

3.2 Strict types

Goal: no false negatives
  - Appropriate for experienced developers?
  - Variants of "usual techniques" apply, e.g. progress becomes "if you get stuck, there must be red squigglies"

- Related to blame analysis?

3.3 Nonstrict types

Goal: no false positives
- Appropriate for the majority of developers?
- Usual techniques do not apply, e.g. correctness becomes "code with red squigglies does not return a result"
- Related to success types?

3.4 Mixing types

Goal: support mixed strict/nonstrict development
- Strictness is per-script, so programs are mixed
- Can the correctness criteria be combined?
- Can success types be combined with regular types?
- Same types, different red squigglies?
- Related: incorrectness logic vs correcness logic?

4 FUTURE WORK

Draw the damn owl
- Mixing types
- Other interactions between types and IDEs, e.g. typed holes.
- Formalizations of all of this?

REFERENCES

[1] Lua.org and PUC-Rio. 2021. The Lua Programming Language. https://lua.org

[2] Roblox. 2020. Roblox Developers Expected to Earn Over $250 Million in 2020; Platform Now Has Over 150 Million Monthly Active Users. https://corp.roblox.com/2020/07/roblox-developers-expected-earn-250-million-2020-platform-now-150-million-monthly-active-users/

[3] Roblox. 2021. The Luau Programming Language. https://luau-lang.org

[4] Roblox. 2021. Roblox Education: All Educators. https://education.roblox.com/en-us/educators

[5] Roblox. 2021. What is Roblox. https://corp.roblox.com