Position Paper: Goals of the Luau Type System

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

A position paper about goals of the Luau type system.

ACM Reference Format:
Andy Friesen, Alan Jeffrey, and Other People?. 2021. Position Paper: Goals of the Luau Type System. In . ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Roblox [7] 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 creators are often young, for example there are over 200 Roblox kids' coding camps in over 65 countries listed at [6].

The Luau programming language [5] is the scripting language used by developers of Roblox experiences. Luau is derived from the Lua programming language [2], with additional capabilities, including a type inference engine.

This paper will discuss some of the goals of the Luau type system, focusing on where the goals are different from those of other type systems.

2 HUMAN ASPECTS

2.1 Heterogenous developer community

Quoting a 2020 report [4]:

- 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 are 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, seen in Fig. 1(a). The user designs and deploys 3D assets such as terrain, parts and joints, and provides them

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

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

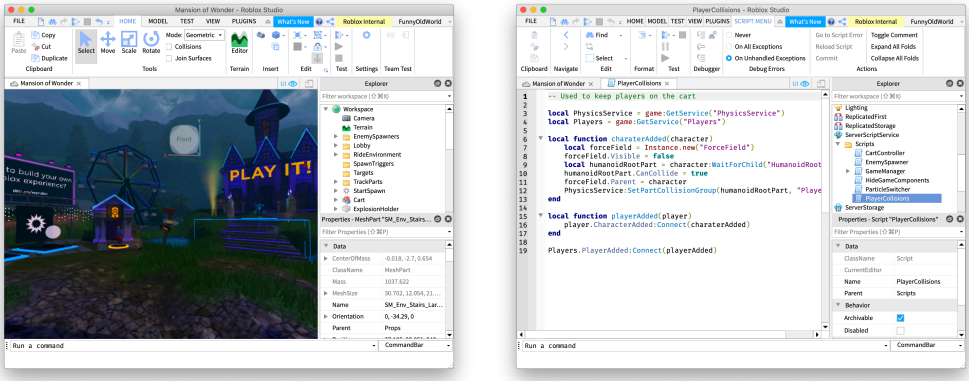

Fig. 1. Roblox Studio's 3D environment editor (a), and script editor (b)

with 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, seen in Fig. 1(b), and the Luau programming language.

This onboarding experience 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. It 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

Professional development studios are also goal-directed (though the goals may be more abstract, such as "decrease user churn" or "improve frame rate") but have needs that are less common in learners:

- *Code planning*: code spends much of its development time in an incomplete state, with holes that will be filled in later.
- *Code refactoring*: experiences evolve over time, and it easy for changes to break previously-held invariants.
- *Defect detection*: code has errors, and detecting these at runtime (for example by crash telemetry) can be expensive and recovery can be time-consuming.

Detecting defects ahead-of-time is a traditional goal of type systems, resulting in an array of techniques for establishing safety results, surveyed for example in [3]. Supporting code planning and refactoring are some of the goals of *type-driven development* [1] under the slogan "type, define, refine".

To help support the transition from novice to experienced developer, types are introduced gradually, through API documentation and type discovery. Type inference provides many of the benefits of type-driven development even to creators who are not explicitly providing types.

3 TYPES

3.1 Infallible types

Goal: *support type-driven tools in all programs.*

Programs spend much of their time under development in an incomplete state, even if the final arifact is well-typed. Type-driven tools should support this, by providing type information for all programs.

An analogy is infallible parsers, which perform error recovery and provide an AST for all input texts.

Program analysis can still flag type errors, for example with red squiggly underlining. Formalizing this, rather than a judgement $\Gamma \vdash M : T$, for an input terms M, there is a judgement $\Gamma \vdash M \Rightarrow M' : T$ where M' is an output term where some subterms are flagged $\underline{M}$. For example the usual type rules for field access becomes:

$$\frac{\Gamma \vdash M \Rightarrow M' : T}{\Gamma \vdash M.\ell \Rightarrow M'.\ell : U}[T = \{\overline{\ell : U}\} \text{ and } (\ell : U) \in (\overline{\ell : U})]$$

but there is also a rule for unsuccesful field access:

$$\frac{\Gamma \vdash M \Rightarrow M' : T}{\Gamma \vdash M.\ell \Rightarrow \underline{M'.\ell} : U}[T = \{\overline{\ell : U}\} \text{ implies } \ell \notin \overline{\ell}]$$

In this type rule, U is unconstrained.

Some issues raised by infallible types:

- Which heuristics should be used to provide types for flagged programs? For example, could one use minimal edit distance to correct for spelling mistakes in field names?
- How can we avoid cascading type errors, where a developer is faced with type errors that are artifacts of the heuristics rather than genuine errors?
- How can the goals of an infallible type system be formalized?

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?
- Problems with mutation and avoiding whole-program analysis.

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] Edwin Brady. 2017. *Type-Driven Development with Idris.* Manning.
[2] Lua.org and PUC-Rio. 2021. The Lua Programming Language. https://lua.org
[3] Benjamin C. Pierce. 2002. *Types and Programming Languages.* MIT Press.
[4] 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/
[5] Roblox. 2021. The Luau Programming Language. https://luau-lang.org
[6] Roblox. 2021. Roblox Education: All Educators. https://education.roblox.com/en-us/educators
[7] Roblox. 2021. What is Roblox. https://corp.roblox.com