Putting the Plane Together Midair

Andrew Woo
Gameplay Engineer, Riot Games

League of Legends

- **15 million+** registered users
- **500,000+** combined PCU NA & EU
- **2** week patch cycle

Great gameplay programming

embraces the **unknown**

Common programming wisdom

says you can optimize on

Time and **Space**

To find **Fun** you need to

optimize on **Life**

Designers need to do **crazy** stuff.

Flexible code allows you to say

"**Yes!**"

> Everything should be made as **simple** as possible, but **no simpler**.

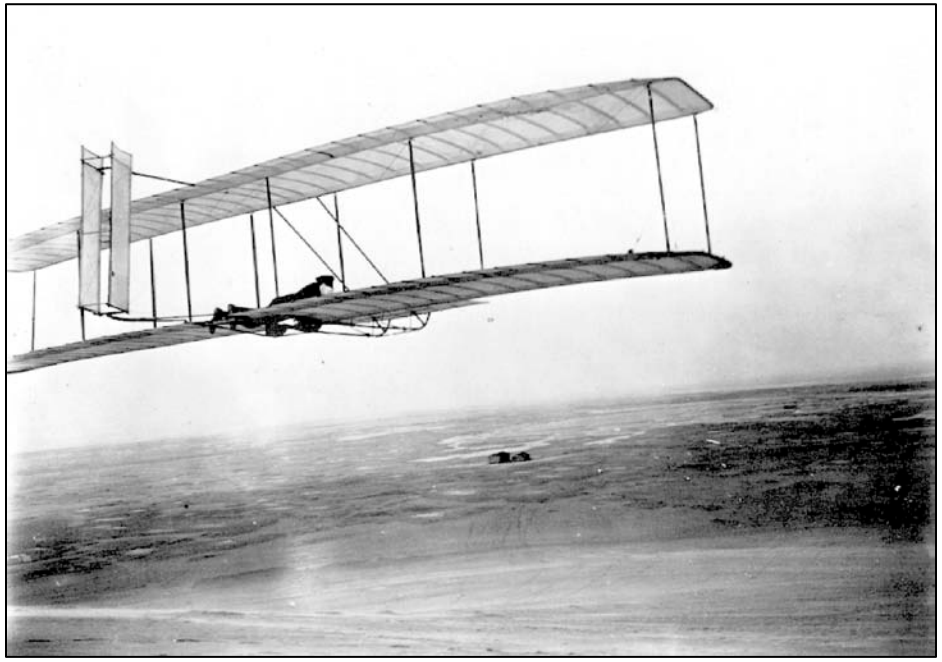
(Thanks, Albert Einstein!)

The Challenge

We want Moar!

- Advanced Tutorial
- More Intelligent AI
- New Game Modes

The Old and the New



Pain Points

- Hard to **debug**
- Hard to **optimize**
- Hard to **use**

Optimize on **Flexibility**
Efficiency
Iteration
Speed

Implementation Details

- Visual language based on **Behavior Trees**
- “Compiler” and “Virtual Machine” in **C++**
- Visual front-end tool in **C#**
- Underlying data format is clear-text **XML**
- **Co-exists** with our old scripting language!

The Secret Sauce

- Behavior Trees
- Design Patterns
- Storing State
- Event-Driven Trees

Behavior Trees

Scripting

- **Strengths**
  - Complete and direct gameplay control
  - Current systems already implemented in lua
- **Weaknesses**
  - Hard to debug and optimize
  - Requires a lot of engineering expertise from your Designers

Hierarchical FSMs

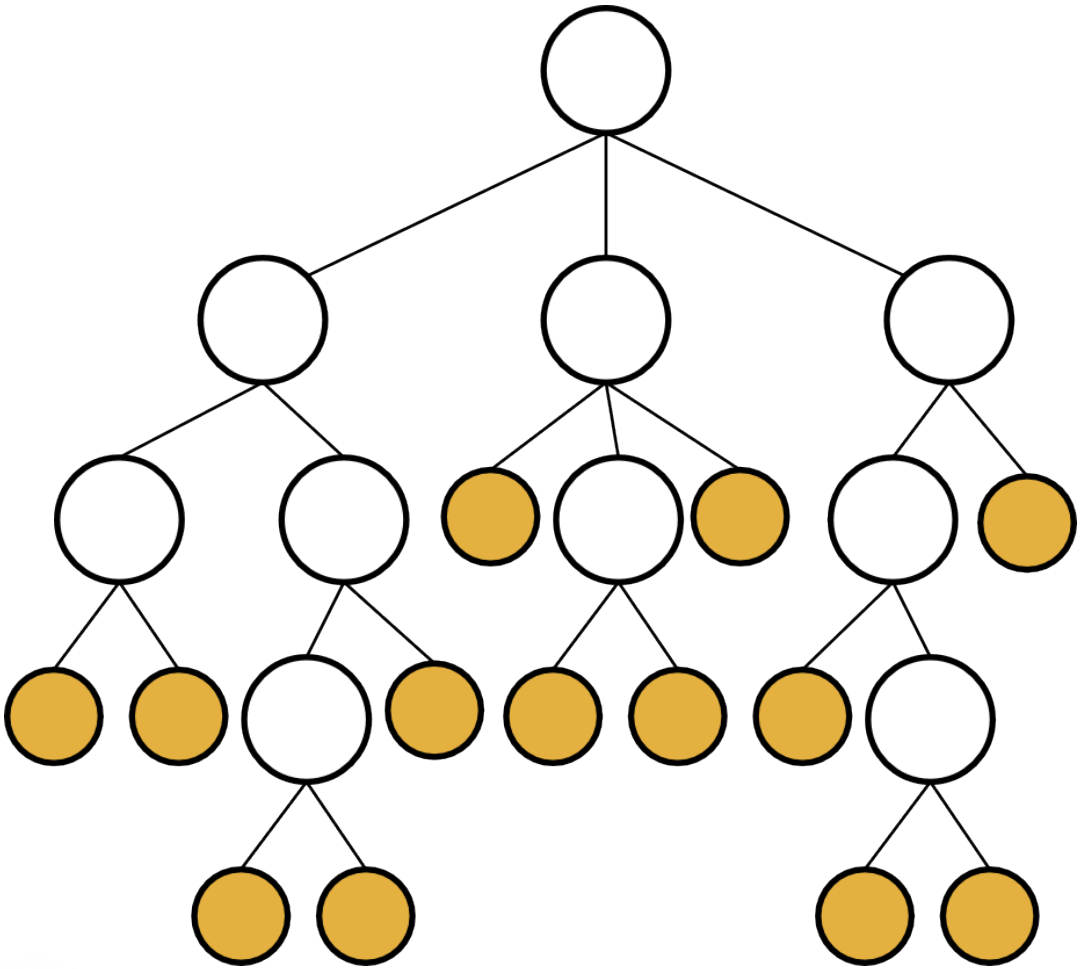
- **Strengths**
  - Intuitive for Designers
  - Pretty good low-level control
- **Weaknesses**
  - Difficult to scale and reuse
  - Hard to make goal-directed

Why Behavior Trees?

Takes the **power** and **flexibility** of scripting and makes it a **simple** visual language for Designers.

Takes the **intuitive** and **reactive** power of Hierarchical FSMs and makes it **reusable** and **goal-directed**

Behavior Tree



Key Node Types

Sequence: short-circuit AND (&&)

Selector: short-circuit OR (||)

Decorator: FOR loops

Condition: game state checks

Action: gameplay interaction

Flexible code allows you to say

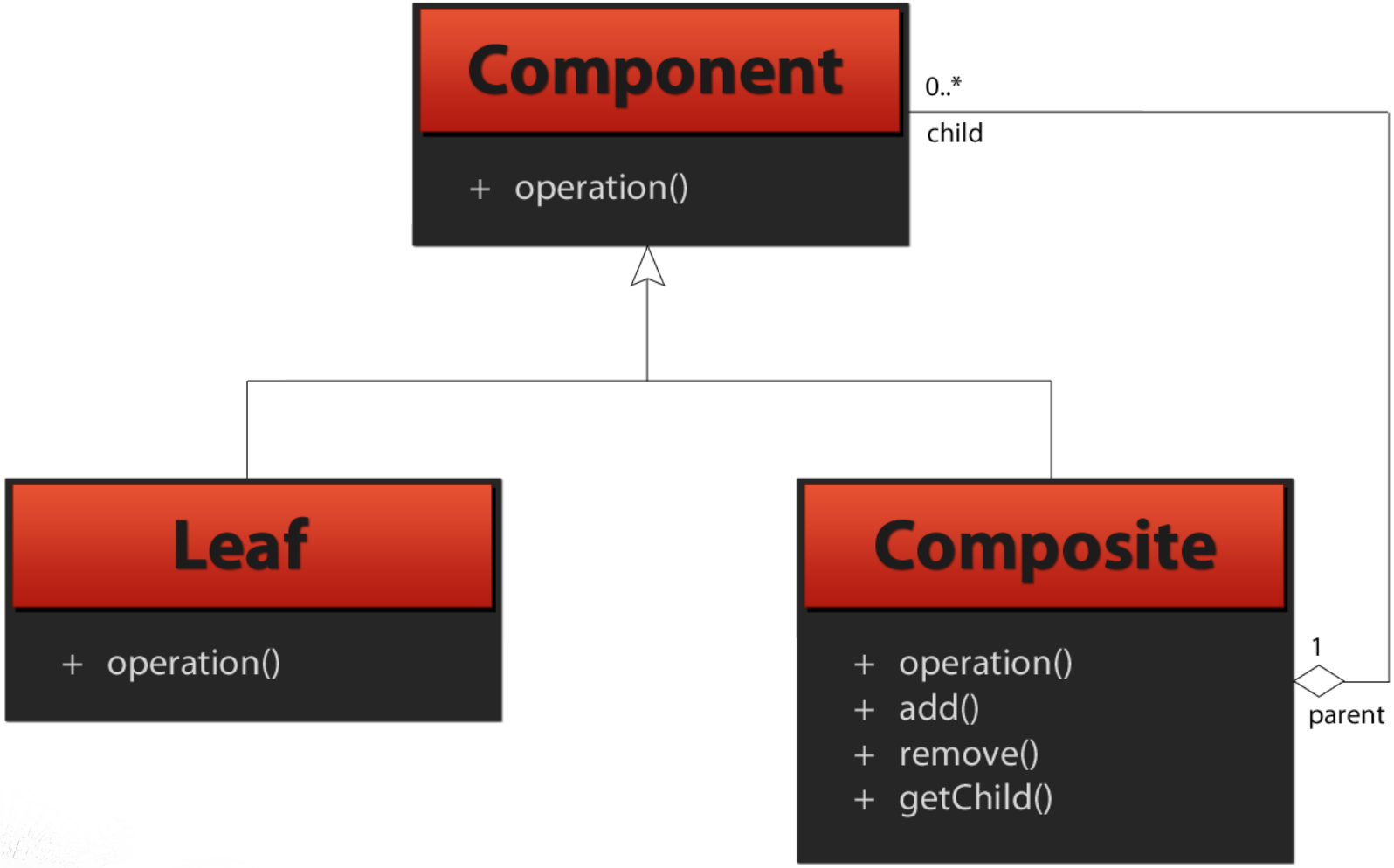
“**Yes!**”

Design Patterns

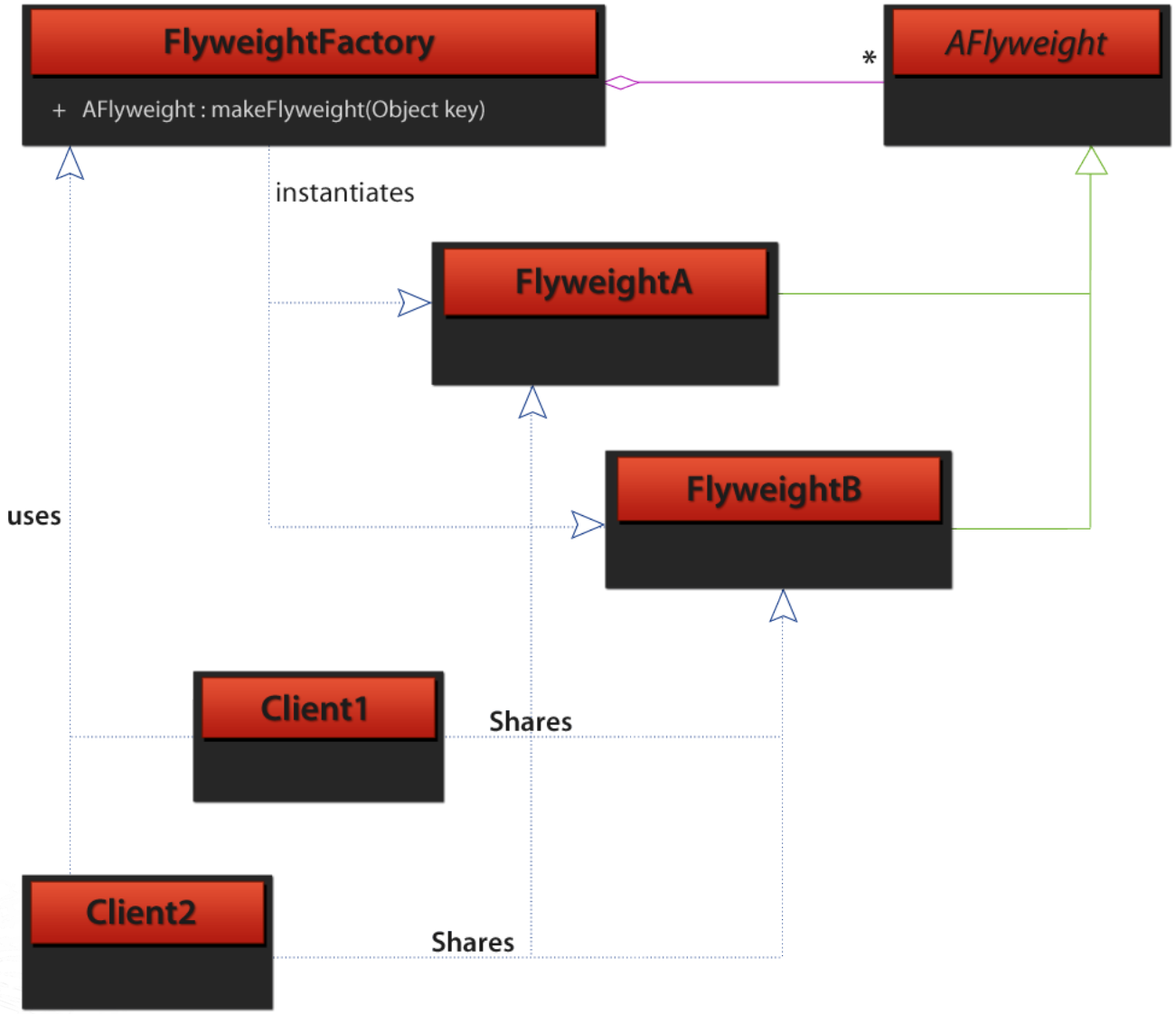
Goals

- User-defined nodes and Behavior Trees **[Composite]**
- Multiple AI Actors using the same Behavior Tree definitions **[Flyweight]**
- Need AI Actors to each be able to walk the same tree and keep their own state **[Visitor]**

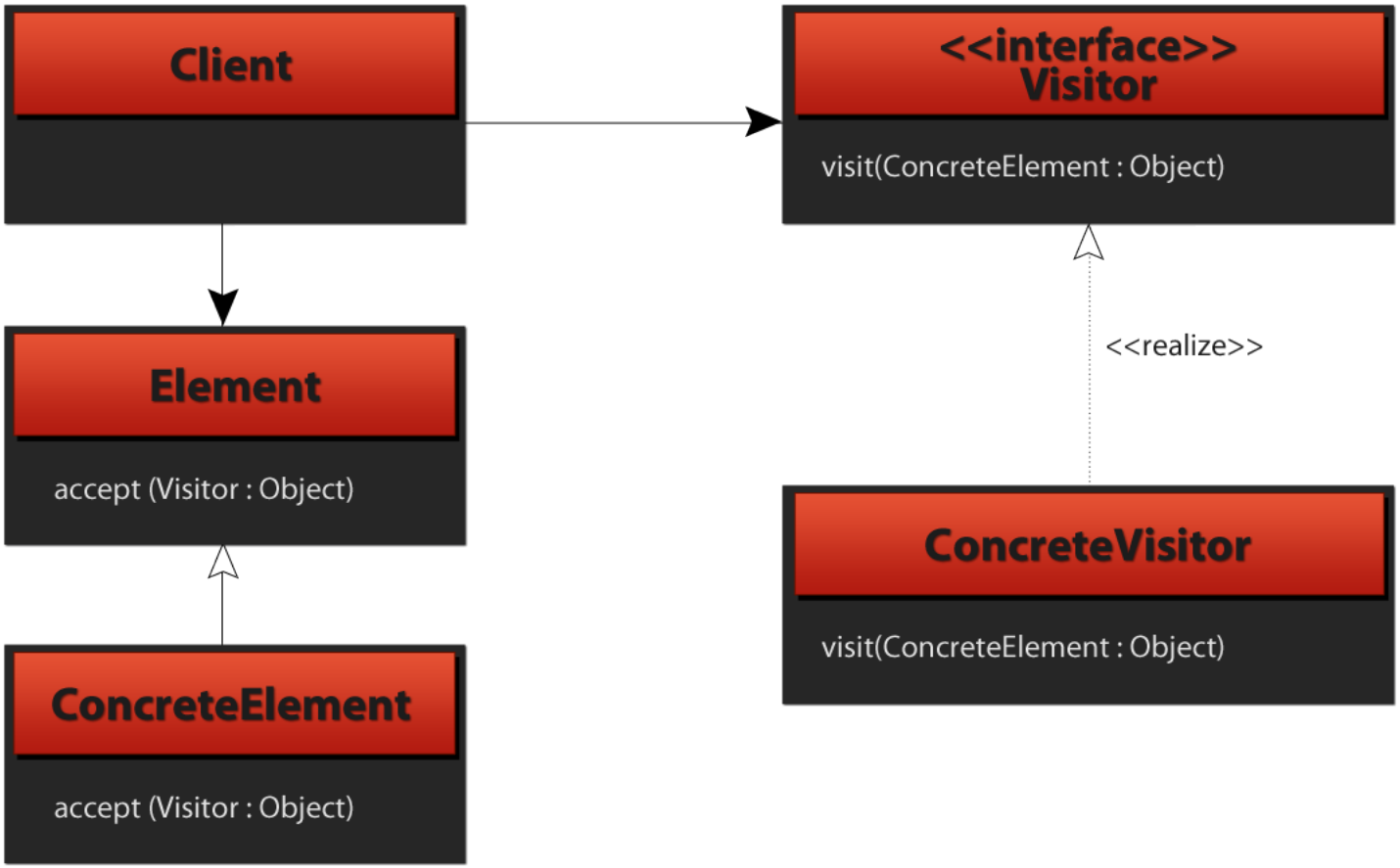
Composite


Component
+ operation()
0..*
child
Leaf
+ operation()
Composite
+ operation()
+ add()
+ remove()
+ getChild()
1
parent

Flyweight


FlyweightFactory
+ AFlyweight : makeFlyweight(Object key)
*
AFlyweight
instantiates
FlyweightA
FlyweightB
uses
Client1
Shares
Client2
Shares

Visitor


Client
<<interface>>
Visitor
visit(ConcreteElement : Object)
Element
accept (Visitor : Object)
<<realize>>
ConcreteVisitor
visit(ConcreteElement : Object)
ConcreteElement
accept (Visitor : Object)

Finding **Fun** means coding for

Change

Storing State

Goals

- Letting nodes and trees communicate with each other
- Allow nodes to be user-defined
- Type safety

Property Maps

- Map of “parameter name” and boost::any
- Let the Summoner Script execute or create any tree without caring about what nodes do
- Allowed us to enforce type safety both in the visual tool and the C++ code

Great gameplay programming

embraces the **unknown**

Event-Driven Trees

RIOT GAMES

Goals

- Allow the Designers to learn and become experts in a single language
- Have fast execution for both update-based AI scripting and event-based level-scripting

Event-Driven Trees

Event-Driven Trees are **just like** Update-Driven Trees except they only tick **once**.


LEAGUE of LEGENDS
DOMINION

Conclusions

Finding **Fun** means coding for

Change

Optimize on **Flexibility**
Efficiency
Iteration
Speed

awoo@riotgames.com