WHY (M)RUBY SHOULD BE YOUR NEXT SCRIPTING LANGUAGE?

Tomasz Dąbrowski / Rockhard

GIC 2016 Poznań

WHAT DO WE WANT?

WHAT DO WE WANT?

- fast iteration times
- easy modelling of complex gameplay logic & UI
- not reinventing the wheel
- mature tools
- easy to integrate

WHAT DO WE HAVE?

MY PREVIOUS SETUP

- Lua
- not very popular outside gamedev (used to be general scripting language, but now most applications seem to use python instead)
- even after many years I haven't gotten used to its weird syntax (counting from one, global variables by default, etc)
- no common standard - everybody uses Lua differently
- standard library doesn't include many common functions (ie. string.split)

WHAT DO WE HAVE?

- as of 2016, **Lua** is still a gold standard of general game scripting languages
- **C#** (though not scripting) is probably even more popular because of the Unity
- Unreal uses proprietary methods of scripting (UScript, Blueprints)
- **Squirrel** is also quite popular (though nowhere near Lua)
- **AngelScript**, **Javascript** (V8), **Python**… are possible yet very unpopular choices
- and you can always just use **C++**

MY CRITERIA

POPULARITY

- popularity is not everything
- but using a popular language has many advantages
- **most problems you will encounter have already been solved (many times)**
- more production-grade tools
- more documentation, tutorials, books, etc

- **most problems you will encounter have already been solved (many times)**

- this literally means, that you will be able to have first prototype of anything in seconds by just copying and pasting code

- (you can also use package manager to do that and then it's no longer frowned upon)

- Much easier for someone new to learn Python/Ruby than Lua
- Multiple books not only in English but also in Polish (and other languages)
- Even books for kids!

Javascript 1,234,435 questions tagged	Python 641,823 questions tagged	C# 229,926 questions tagged
C++ 474,773 questions tagged	Ruby 169,631 questions tagged	Bash 74,420 questions tagged not actually recommended, needed 9th one for aligned table
Lua 11,221 questions tagged	Squirrel 23 questions tagged	AngelScript 19 questions tagged

Javascript 1,234,435 questions tagged	Python 641,823 questions tagged	C# 229,926 questions tagged
C++ 474,773 questions tagged	Ruby 169,631 questions tagged	Bash 74,420 questions tagged not actually recommended, needed 9th one for aligned table
Lua 11,221 questions tagged	Squirrel 23 questions tagged	AngelScript 19 questions tagged

(ACTUAL) SCRIPTING

- while the compilation phase has its benefits, what I really want to have is **super fast iteration time**

- so I can hot-swap any code at any moment

- will prefer dynamic data structures over performance

- also important on platforms where you want to hotpatch code over the network (mobile, consoles)

Javascript 1,234,435 questions tagged	Python 641,823 questions tagged	C# 229,926 questions tagged
C++ 474,773 questions tagged	Ruby 169,631 questions tagged	Bash 74,420 questions tagged not actually recommended, needed 9th one for aligned table
Lua 11,221 questions tagged	Squirrel 23 questions tagged	AngelScript 19 questions tagged

Javascript 1,234,435 questions tagged	Python 641,823 questions tagged	C# 229,926 questions tagged
C++ 474,773 questions tagged	Ruby 169,631 questions tagged	Bash 74,420 questions tagged not actually recommended, needed 9th one for aligned table
Lua 11,221 questions tagged	Squirrel 23 questions tagged	AngelScript 19 questions tagged

SIZE

- size, and difficulty of embedding
- some languages were not designed for embedding - they require multiple files/bundles, custom build steps, etc.
- some have huge runtimes you can't easily get rid of, and some are just huge because they can
- you don't want 50MB of language runtime over your 10MB mobile game

SIZE

- Python2: ~4MB + runtime
- Python3: ~5MB + runtime
- mruby: ~0.5MB for everything
- Lua: ~0.25MB for everything
- V8 (Javascript): >10MB, super complicated build process

Javascript 1,234,435 questions tagged	Python 641,823 questions tagged	C# 229,926 questions tagged
C++ 474,773 questions tagged	Ruby 169,631 questions tagged	Bash 74,420 questions tagged not actually recommended, needed 9th one for aligned table
Lua 11,221 questions tagged	Squirrel 23 questions tagged	AngelScript 19 questions tagged

Javascript 1,234,435 questions tagged	Python 641,823 questions tagged	C# 229,926 questions tagged
C++ 474,773 questions tagged	Ruby 169,631 questions tagged	Bash 74,420 questions tagged not actually recommended, needed 9th one for aligned table
Lua 11,221 questions tagged	Squirrel 23 questions tagged	AngelScript 19 questions tagged

AND THE WINNER IS…

MRUBY

- Created in 2012, as an embeddable version of Ruby
- written in portable C (can be included in project just by copying sources)
- based on Ruby ISO spec
- **very** minimal standard library, with many common features extracted to optional gems
- there is also mrubyc - a version for microcontrollers with 1/10 of the size (~40 KB)

RUBY

- Object-oriented language
- Strong dynamic type system
- Optimized for programmer's performance
- Primitives: integers, floats, true/false/nil, strings, symbols, arrays, hashes, closures
- Clear separation of local, @instance and $global variables
- Exception handling

RUBY

- Any function can take a block of code as a parameter, and many builtin functions do that:
  - ```
    [“John”, “Тим”, “まつもと”].each do |name|
      print “Hello #{name}”
    end
    ```
- `each` is just an instance method of Array
    ```

RUBY

- you can easily create your own methods with blocks:

```
def twice
  yield
end

# two different syntax options:
twice do print “Banana” end
twice { print “Banana” }
```

- or in this case just use integer builtin:

```
2.times { print “Banana” }
```

MRUBY

- accepts most of the Ruby syntax
  (including 2.3 safe navigation operator)

- has a binary precompiled format so you don't need to provide sources with your game

- you can use the same sources with Ruby and mruby, which is very useful for integration with other services and testing

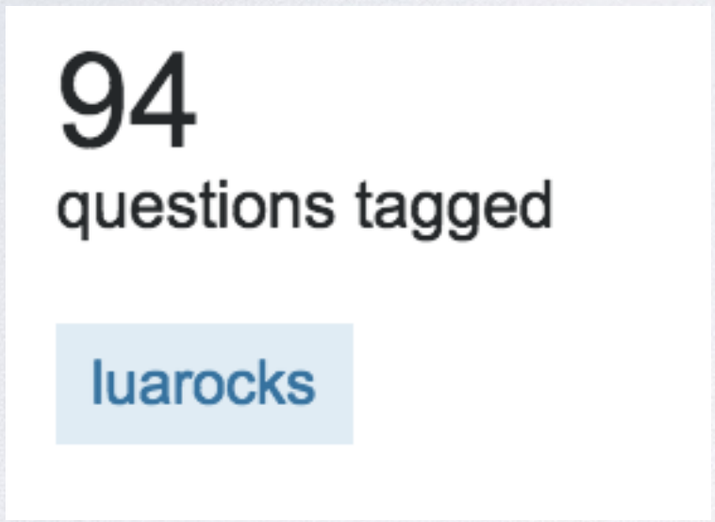
- ie. for multiplayer games, you could share your game logic with the backend written in Ruby on Rails

PACKAGES (GEMS)

- ready-to-use code packages, with dependencies
- mixed native and ruby code
- ruby: packages are dynamic
- mruby: packages are static (selected during build)
- a lot of ruby features are optionally available as gems, either included with mruby, or separate

PACKAGES (GEMS)

- Lua has luarocks, but it's not very popular



- none of gamedev developers I've talked (who uses Lua) is using luarocks (or any external code for that matter)

PACKAGES (GEMS)

- while the number of native mruby gems available now is limited, you can port "big" Ruby gems (though at this stage you will need to check and possible extend stdlib, depending on the kind of code you're looking for)

MRUBY & RUBY

- since mruby code is also valid Ruby, you can use any existing tools to create, debug, profile and test your code

MRUBY VS LUA

MRUBY VS LUA FORWARD AND REVERSE LOOPS

- Lua
  - ```
    for key, value in ipairs(table) do
      …
    end
    ```
  - ```
    -- http://lua-users.org/wiki/IteratorsTutorial
    function ripairs(t)
      idx={}
      for k,v in pairs(t) do
        if type(k)=="number" then idx[#idx+1]=k end
      end
      table.sort(idx)
      local function ripairs_it(t,_)
        if #idx==0 then return nil end
        k=idx[#idx]
        idx[#idx]=nil
        return k,t[k]
      end
      return ripairs_it, t, nil
    end

    for key, value in ripairs(table) do
      …
    end
    ```

MRUBY VS LUA
FORWARD AND REVERSE LOOPS

- but…
  - you need to know about the difference between pairs/ipairs
  - no built-in reverse iterator
  - key order is neither sorted nor preserved

MRUBY VS LUA
FORWARD AND REVERSE LOOPS

- Ruby
  - ```
    array.each do |value|
      …
    end
    ```
  - ```
    array.reverse_each do |value|
      …
    end
    ```
  - ```
    hash.sort.each do |key, value|
      …
    end
    ```
    ```

SOMETHING SIMPLER?

MRUBY VS LUA ARRAY SHALLOW COPY

- Lua

```
function shallowcopy(orig)
    local orig_type = type(orig)
    local copy
    if orig_type == 'table' then
        copy = {}
        for orig_key, orig_value in pairs(orig) do
            copy[orig_key] = orig_value
        end
    else -- number, string, boolean, etc
        copy = orig
    end
    return copy
end

new_array = shallowcopy(array)
```

- Ruby

```
new_array = array.clone
```

MRUBY VS LUA

- And I'm not even starting with OOP!
- While prototype-based system used in Lua *can* be quite powerful, it is very complicated for both non-programmers (ie. designers working on scripts) and most coders accustomed to typical OOP
- Ruby on the other hand has a powerful object engine, with easy inheritance, encapsulation, mixins (and even hotpatching on class and object level if required)

LUA

- While you certainly can do anything in Lua, there is a lot of code to write and test first
- There are many nuances in Lua coding that are easy to miss (like arrays with empty spaces)
- With (m)ruby I don't need to worry about this basics

DEMO TIME!

Q & A

t.dabrowski.ruby@rock-hard.eu