The Programming Language Céu

LabLua – PUC-Rio
www.lua.inf.puc-rio.br

Francisco Sant'Anna

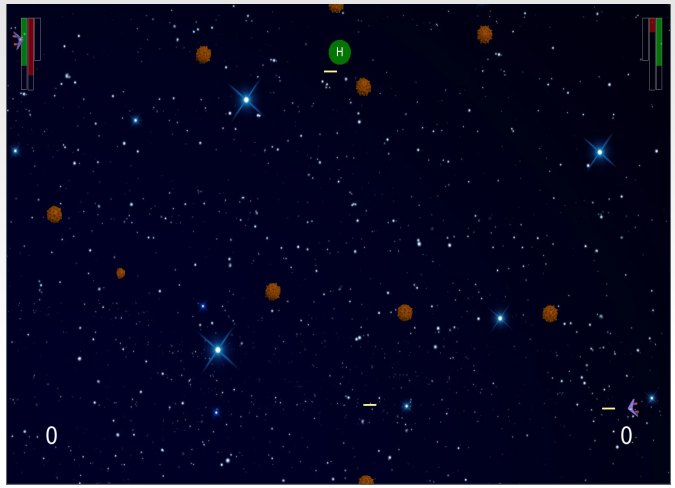
Who I am

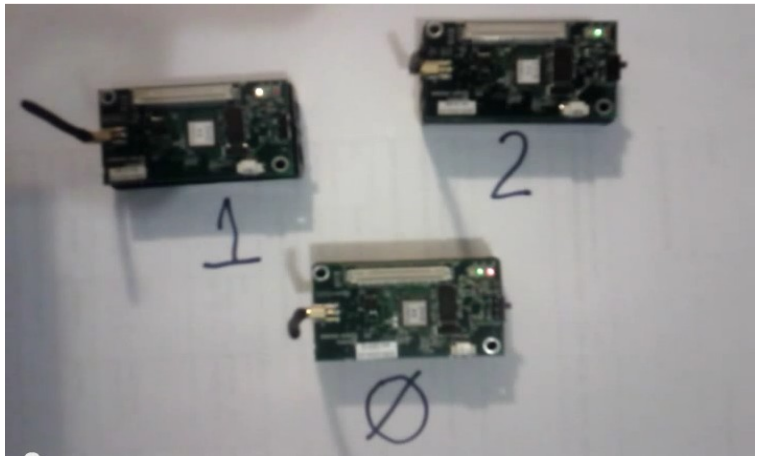
- Academic from Rio de Janeiro, Brazil
- LabLua @ PUC-Rio University
  - The home of Lua
  - Research in programming languages
- Single designer/developer of Céu
- Currently a Postdoc
  - considerable time on Céu (but I wish was full time)

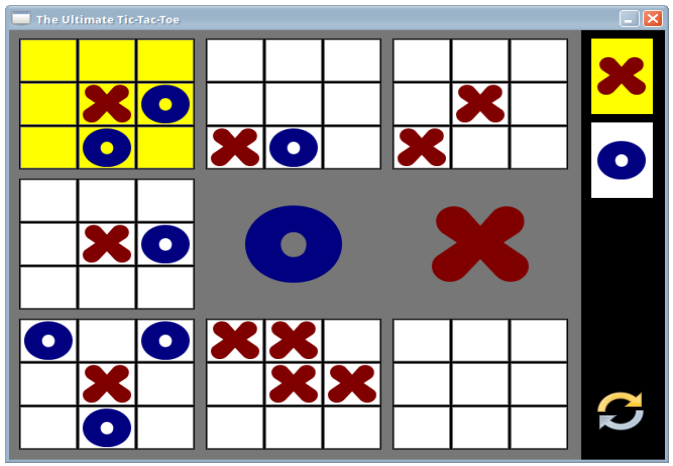
What is Céu?

- Alternative to C for embedded applications
  - more safety
  - more expressiveness
- Embedded
  - the application is embedded in an environment and has to react to events from it in real time
- Based on Esterel (a French language born in the '80s)
- Céu in keywords
  - synchronous, imperative, reactive, concurrent, deterministic
- Céu means “sky” in Portuguese (Lua means “moon”)

Embedded/Reactive/Real time







Why Céu (vs C)

- Safety
  - safe resource management
    - leaked/dangling pointers
  - finite bounds in programs
    - unbounded memory
    - unbounded execution
- Expressiveness (reasoning)
  - events (emit, await)
  - concurrency (par, par/and, par/or)
  - lexical scope

Céu in one feature: `par/or`

- `par`: lexical composition of activites
  - static reasoning, static memory
  - synchronous execution
- `or`: orthogonal abortion
  - safe/consistent termination
  - no need for garbage collection
- permeates all aspects of the language
  - organism expansion, reference watching, finalization

Timeline

- `[2007-2009]`: MSc., **LuaGravity**, reactive extensions to Lua
- `[2009-2013]`: PhD., **Céu**, new language from scratch
  - focus on sensor networks (drivers, protocols)
  - `[jan 2012]`: first public release
  - `[nov 2013]`: paper on *SenSys'13*
- `[2014-2015]`: PostDoc, evolving **Céu**
  - *organism* abstraction
    - towards general applications
  - `[mar 2015]`: paper on *Modularity'15*
  - `[apr 2015]`: latest release (v0.9)

Next

- Reactive data structures
  - From: enum, struct, union, pointers
  - To: algebraic datatypes, arrays, hash tables
- Requests
  - bi-directional communication
    - deals with multiple requests, errors, cancellations

```
input int X_DONE;
_request_x(v1);     // makes the request
v2 = await X_DONE; // awaits the answer
```

```
output int X_REQUEST;
input  int X_DONE;
emit X_REQUEST=>v1; // makes the request
v2 = await X_DONE;  // awaits the answer
```

```
output/input int=>int X;
v2 = (request X=>v1);
        // makes the request and awaits the answer
```

Non-academic “wish list”

(aka *mea-culpa*)

- Error messages
- Documentation
- Debugger!

Comments on “libaqm”

Comments on “libaqm”

- Favor `every` over `loop`
  - easier to reason
  - refuses awaits inside
    - i.e., ensures that **every** occurrence is handled
  - less lines of code :)

```
// run.ceu
loop do
  var _context context;
  ... // other vars
  (context, ...) = await START_IA_PROC;
  ... // non-awaiting statements
end
```

```
// run.ceu
var _context context;
... // other vars
every (context, ...) in START_IA_PROC do
  ... // non-awaiting statements
end
```

Comments on “libaqm”

- Use `finalize` for “post execution”
  - easier to reason
  - ensures execution regardless of external termination
  - less lines of code :)

```
// ia_prep.ceu
do
  _setdedicatedtoIA(true);
  par/or do
    ... // code that awaits
    _setdedicatedtoIA(false);
  with
    ... // code that awaits
    _setdedicatedtoIA(false);
  end
end
```

```
// ia_prep.ceu
do
  _setdedicatedtoIA(true);
  finalize with
    _setdedicatedtoIA(false);
  end
  par/or do
    ... // code that awaits
  with
    ... // code that awaits
  end
end
```

Comments on “libaqm”

- `do T` can return value
  - easier to reason
  - makes the organism anonymous
  - less lines of code :)

```
// ia_prep.ceu
var SealCheck first_sealcheck with
  this.context = context;
end;
var bool ok = await first_sealcheck.ok;
...
```

```
// ia_prep.ceu
var bool ok =
  do SealCheck with
    this.context = context;
  end;
...
```

Comments on “libaqm”

- Favor **`output`** events over **`native`** calls
  - analogous to **`input`** events
  - maps to **`native`** calls at the very end (configurable)
- Convert **`emit/await`** to **`request`**
  - matches **`emit/await`** with session IDs
    - i.e., no need for **`await-until`**

```
// bindings.ceu
#define MOVEABS( dev, pos )   \
do                            \
  _moveabs( dev, pos );       \
  var _dev_ids_t resp_dev =   \
    await MOVE_DONE until     \
      dev == resp_dev;        \
end

... // many other similar macros
```

```
// bindings.ceu
output/input (int,float)=>void MOVEABS;

#define MOVEABS(dev, pos)   \
  request MOVEABS=>(dev,pos)
```

Questions for the team

Questions for the team

- Motivations do use Céu?
- Process to adopt Céu?
  - Many *no* reasons: still pre-1.0, lack of documentation and tools, not backed by a company, primarily academic goals
- Expectations and Reality?
- Development of "libapm"?
  - APP=*1568 LoCs*, TST=*3182 LoCs*
    - Any estimates for these numbers in C/C++?
- Missing features in Céu?

Thanks!

LabLua – PUC-Rio
www.lua.inf.puc-rio.br

Francisco Sant'Anna