Structured Synchronous Reactive Programming for Game Development

Case Study: On Rewriting Pingus from C++ to Céu

www.ceu-lang.org

Francisco Sant'Anna

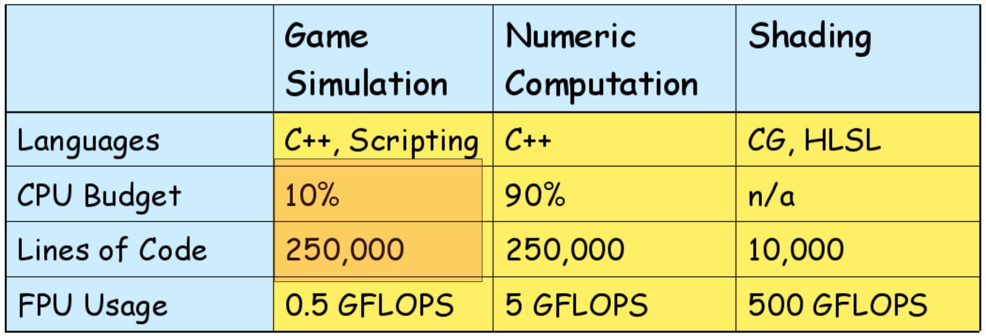
Rio de Janeiro State University

francisco@ime.uerj.br

@_fsantanna

Game Logic / Simulation

- About 50% of complexity in games
- But only 10% of CPU budget
- “Will gladly sacrifice 10% of our performance for 10% higher productivity.”
- **Appeal for programming alternatives concerning game logic productivity**



	Game Simulation	Numeric Computation	Shading
Languages	C++, Scripting	C++	CG, HLSL
CPU Budget	10%	90%	n/a
Lines of Code	250,000	250,000	10,000
FPU Usage	0.5 GFLOPS	5 GFLOPS	500 GFLOPS

[Tim Sweeney, POPL’06]

"Hello world!" in Céu

- Blinking a LED
    1. *on ↔ off every 500ms*
    2. *stop after "press"*
    3. *restart after 2s*
- Compositions
    - seq, loop, par *(trails)*
        - At any level of depth
    - ~~state variables / communication~~

```
loop do
    par/or do
        loop do
            await 500ms;
            _leds_toggle();
        end
    with
        await PRESS;
    end
    await 2s;
end
```

Lines of execution
=
Trails (in Céu)

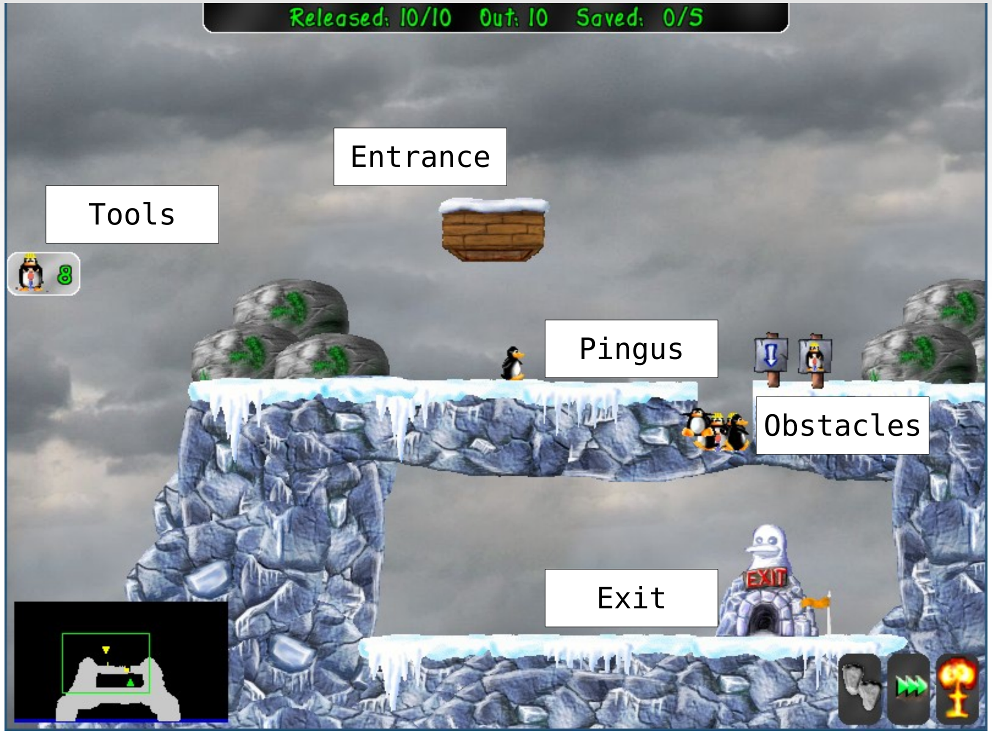
From “Structured Programming” To “Structured Reactive Programming”

- Control Structures
  - Sequences, Loops, Conditionals
- Blocks, Scopes, Locals
  - Automatic memory management
- What about reactivity?
  - Environment event → Short-lived callback
    - No more loops, scopes, etc.
    - Breaks structured programming
      - “Callbacks as our Generations' `goto`”

- The `await` statement
  - *Imperative-reactive* nature
- Compositions
  - Control structures + parallels
- Synchronous execution model
  - Time ~ Sequence of events
  - Deterministic behavior
- Sensor Networks, Embedded Systems, Multimedia, **Games**

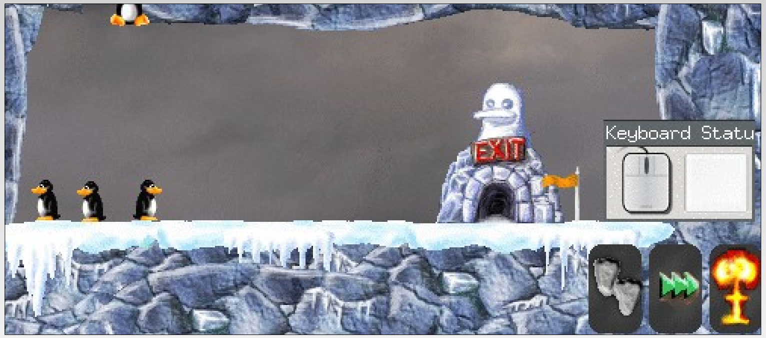
SSRP in Games

- Case Study: Pingus
  - 40k LoC in C++
  - 20k Game Logic
    - 10k config., serial., strings, etc.
    - **10k of reactive code (25%)**
- Qualitative study



```
ArmageddonButton::ArmageddonButton(<...>):
    RectComponent(<...>),
    <...>
    pressed(false); // initial button state
    press_time();   // how long since the 1st click?
    <...>
{
    <...>
}

void ArmageddonButton::draw (<...>) {
    <...>
}

void ArmageddonButton::update (float delta) {
    <...>
    if (pressed) {
        press_time += delta;
        if (press_time > 1.0f) {
            pressed = false;    // giving up, 1st click was
            press_time = 0;     //            too long ago
        }
    } else {
        <...>
        press_time = 0;
    }
}

void ArmageddonButton::on_click (<...>) {
    if (pressed) {
        server->send_armageddon_event();
    } else {
        pressed = true;
    }
}
```

40 LoC



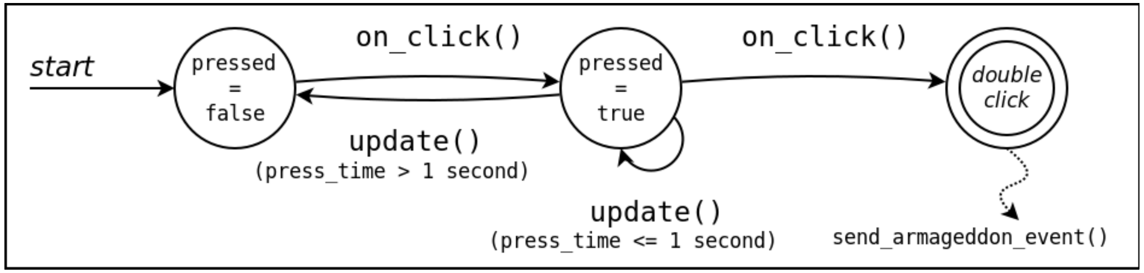
```
class ArmageddonButton with
    <...>
do
    var RectComponent component = <...>;
    <...>
    loop do
        await component.on_click;
        watching 1s do
            await component.on_click;
            break;
        end
    end
    <...>
    emit global:go_armageddon;
end
```

Control-flow patterns in Pingus

1. Finite State Machines
2. Dispatching Hierarchies
3. Lifespan Hierarchies
4. Continuation Passing
5. Signaling Mechanisms *(not in the paper)*

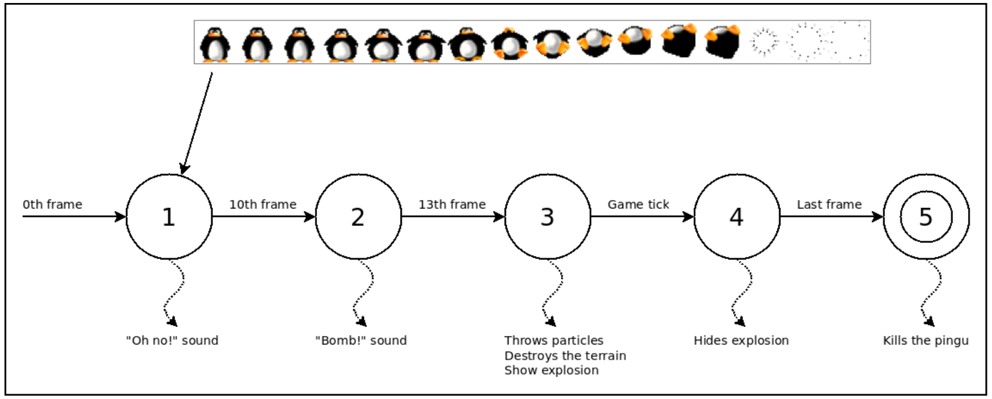
Finite State Machines

- Event occurrences lead to transitions between states and trigger actions comprising the behavior of a game entity.



Finite State Machines

- Event occurrences lead to transitions between states and trigger actions comprising the behavior of a game entity.
- SSRP
  - encode states with sequential code (awaits)
  - eliminate callbacks and shared variables
  - handle states (and only them) in the same contiguous block
- Pingus
  - 30 FSMs in 25 files



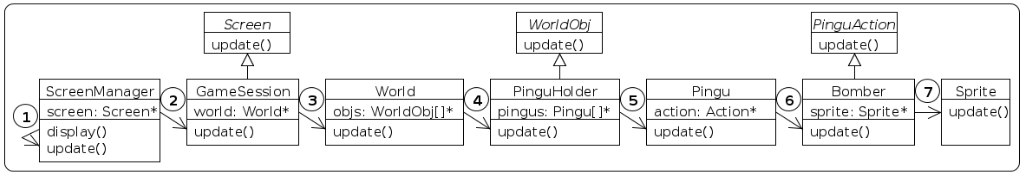
Dispatching Hierarchies

- Entities form a dispatching hierarchy in which a container that receives a stimulus automatically forwards it to its managed children.
  - goes through dozens of files
  - interleaves between game, engine, and possibly third-party classes
- SSRP
  - bypass the program hierarchy entirely
- Pingus
  - `update()/draw()/resize()` touch 50 files and 500 LoC just for dispatching

```
class Bomber : public Action {
    <...>
    Sprite sprite;
}

Bomber::Bomber (<...>) : <...> {
```

```
code Bomber (void) -> ActionName do
    <...>
    var Sprite sprite = spawn Sprite(<...>);
    <...>
end
```

```
    sprite.update();
}

void Bomber::draw () {
    <...>
    sprite.draw();
}
```



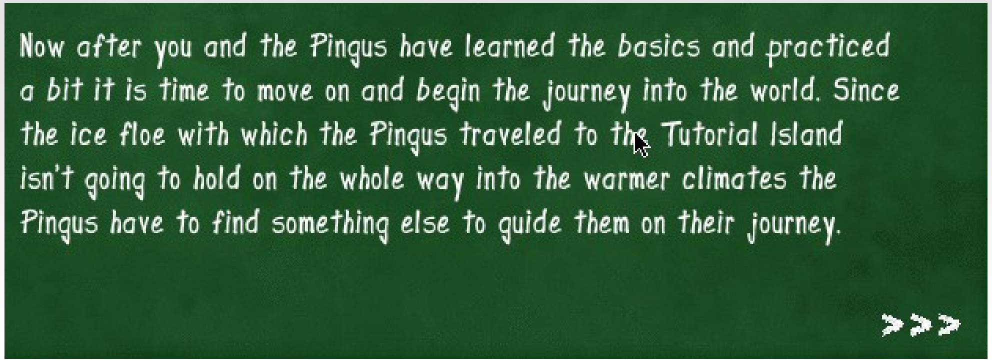
Lifespan Hierarchies

- Entities form a lifespan hierarchy in which a terminating container entity automatically destroys its managed children.
- SSRP
  - lexical scope for abstractions
  - natural termination susceptible to immediate reclamation
- Pingus
  - 200 static and 100 dynamic instantiations

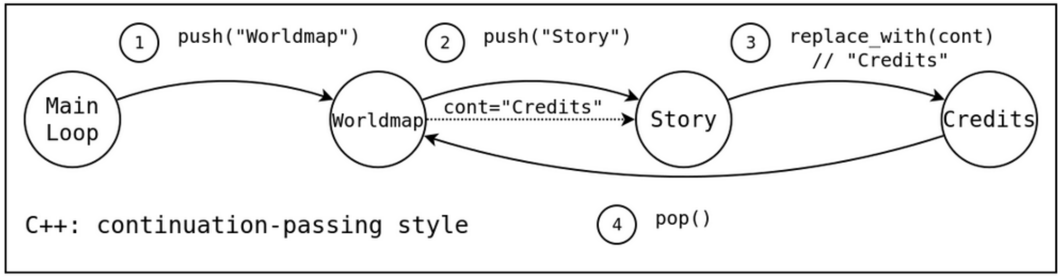
```
Pingu* PinguHolder::create_pingu (<...>) {
    <...>
    Pingu* pingu = new Pingu (<...>);
    pingus.push_back(pingu);
    <...>
}

void PinguHolder::update() {
    <...>
    while(pingu != pingus.end()) {
        (*pingu)->update();
        if ((*pingu)->status() == DEAD) {
           pingu = pingus.remove(pingu);
        }
        <...>
        ++pingu;
    }
}
```

Continuation Passing

- The completion of a long lasting activity in the game may carry a continuation, i.e., some action to execute next.
- SSRP
  - structured control (sequences and loops) vs. data structures and continuation variables (stacks and flags)
  - decoupled activities (no pointers)
- Pingus
  - Screen transitions, menus, level progressions



Now after you and the Pingus have learned the basics and practiced a bit it is time to move on and begin the journey into the world. Since the ice floe with which the Pingus traveled to the Tutorial Island isn't going to hold on the whole way into the warmer climates the Pingus have to find something else to guide them on their journey.



Signaling Mechanisms

- Entities often need to communicate explicitly through signaling mechanisms, especially if there is no hierarchy relationship between them.
- SSRP
  - convenient syntax (emit, await, every)
  - never create infinite dependency loops
- Pingus
  - 39 events in 20 files with 200 invocations in over 50 files

Structured Synchronous Reactive Programming for Game Development

Case Study: On Rewriting Pingus from C++ to Céu

www.ceu-lang.org

Francisco Sant'Anna

Rio de Janeiro State University

francisco@ime.uerj.br

@_fsantanna