Synchronous Retrogames in HTML5

Doussot Alexandre

05/02/17

Introduction

Retrogames were played in the dark arcade rooms of the past millenium. Such games were written in assemby so as to squeeze every bit of performance from hardware that weren't clocked at more than a few Megahertz. Nowadays, a mere smartphone harnesses several Megabytes of RAM and multiple processor cores clocked at several Gigahertz each. Such power allows us to revisit game programming: ease of development takes the lead while performance is sidelined.

Web browsers are ubiquitous - and are quickly becoming the defacto platform to run multiplatform apps. Furthermore, the new web standard HTML5 [1] greatly simplifies the deployment of dynamic application through the Javascript language. Notably, the `canvas` [2] interface allows to draw objects, animate them, and intercept inputs from mouse or keyboards in a Web page.

The canvas API imposes a reactive programming paradigm: in order to update the canvas, one should register a callback function through `requestAnimationFrame()` [3]. One has to do the same to process input.

The reactive limitations imposed by the canvas API is similar to what is used in control-command systems. Several languages, based on the synchronous dataflow paradigm, have been created in order to ease development of such systems. A program written in such a language processes a flow of events - player inputs - and produces a flow of actions corresponding to instructions that will be transmitted to the actuators.

Application

In this internship, we aim at applying this programming model to the implementation of gameplay code. It allows us to express invariants directly in code. We will focus on the application of the synchronous dataflow formalism to retrogame programming:

- We will first implement a Snake [4] clone in pure Javascript.
- We will then design a synchronous dataflow language integrating the HTML5 canvas API.
- We will devise a compiler from this language to Javascript. We aim to follow a similar approach to the one explored in [5]. This compiler should be written in OCaml and will be bootstraped through `js_of_ocaml`.
- We will validate this second approach by implementing an other retrogame - this time using this brand new language.

A simple example

Consider the following Javascript code, which simply moves a square from left to right:

```
var x = 0;
var y = 150;

var dir_enum = Object.freeze({
    LEFT: 37,
    RIGHT: 39,
});

dir = 0;

document.addEventListener("keypress", keyPressHandler, false);

function keyPressHandler(e) {
    dir = e.keyCode;
    switch (dir) {
        case dir_enum.RIGHT:
            x += 20;
            break;
        case dir_enum.LEFT:
            x -= 20;
            break;
        default:
            break;
    }
}

function draw() {
    requestAnimationFrame(draw);
```

This code suffers from several limitations. First, the callback registration is cumbersome. Second, it uses several global variables, which is considered to be poor craftmanship since 1973 [6]. Other problem that may arise when using global variables include non-locality, the absence of access control, implicit coupling and namespace pollution [7].

Now compare it to the following synchronous dataflow program:

```
#static diff

enum Dir = { Up = 37 , Down = 39 }

node move_x (key: Dir, x: int) return (x': int)
let
    x' = match key with
         | Up -> x - diff
         | Down -> x + diff
         end
tel

node main (key : Dir) return (x : int)
let
    x = 0 fby move_x(key, x)
    /* `fby` creates a memory: x's value will be saved for the next call */
tel;
```

Programming with events is abstracted away by the notion of stream, which enables a purely equational approach to programming. Each node is then called on a `tick` defined for each node.

Here, the main node takes a stream of Directions and produces a stream of positions, whose first value will be 0. On each tick - which is here a keypress - `main` will read the next value of `dir` and return a new value of x.

Such code is then compiled to the following Javascript:

```
var dir_enum = Object.freeze({
    LEFT: 37,
    RIGHT: 39,
});

function Main() {
    this.move_x_node = new Move_x();
    this.x = 0;
}

main.prototype.step = function (key) {
    var tmp_x = this.move_x_node.step(key, this.x)
    this.x = tmp_x;
    return tmp_x;
}
```

```
function Move_x() {}

Move_x.prototype.step = function (dir, x) {
    switch (dir) {
        case dir_enum.Left:
            x_1 = x - diff;
            break;
        case dir_enum.Right:
            x_1 = x + diff;
            break;
        default:
    }
    var new_x = x_1;
    return new_x;
}
```

While this code seems bigger than the first excerpt, it also has quite a few benefits. Firstly, we got rid of global variables, which is a net plus. Next, switch-statement completeness is checked at compile-time, which guarantees that no case is left unmatched. Furthermore, this program is guaranteed to be deterministic.

References

[1] Mozilla Developper Network. Html5 developer guide, November 2016. https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

[2] Mozilla Developper Network. Canvas api, November 2016. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas

[3] Mozilla Developper Network. requestAnimationFrame api, November 2016. https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

[4] Wikipedia. Snake, November 2016. https://en.wikipedia.org/wiki/Snake_%28video_game%29

[5] Dariusz Biernacki, Jean-Louis Colaço, Grégoire Hamon & Marc Pouzet (2008): *Clock-directed modular code generation for synchronous data-flow languages.* In Krisztián Flautner & John Regehr, editors: Proceedings of the 2008 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'08), Tucson, AZ, USA, June 12-13, 2008 , ACM, pp. 121–130, doi:10.1145/1375657.1375674.

[6] W. Wulf and Mary Shaw. 1973. *Global variable considered harmful.* SIGPLAN Not. 8, 2 (February 1973), 28-34. DOI=http://dx.doi.org/10.1145/953353.953355

[7] WikiWikiWeb. *Global Variables Are Bad*
http://wiki.c2.com/?GlobalVariablesAreBad