# The potential of declarative programming languages to support user interface programming: the case of Elm

Simon Buist
simon.buist@gmail.com
The University of Bath

## 1. Abstract

*The research area of this project is concerned with investigating the cognitive load a programmer experiences while programming. Two programming languages, Elm and JavaScript will be used to identify differences in cognitive load for a given task. User studies and quantitative analysis will be done to determine these differences.*

## 2. Motivation

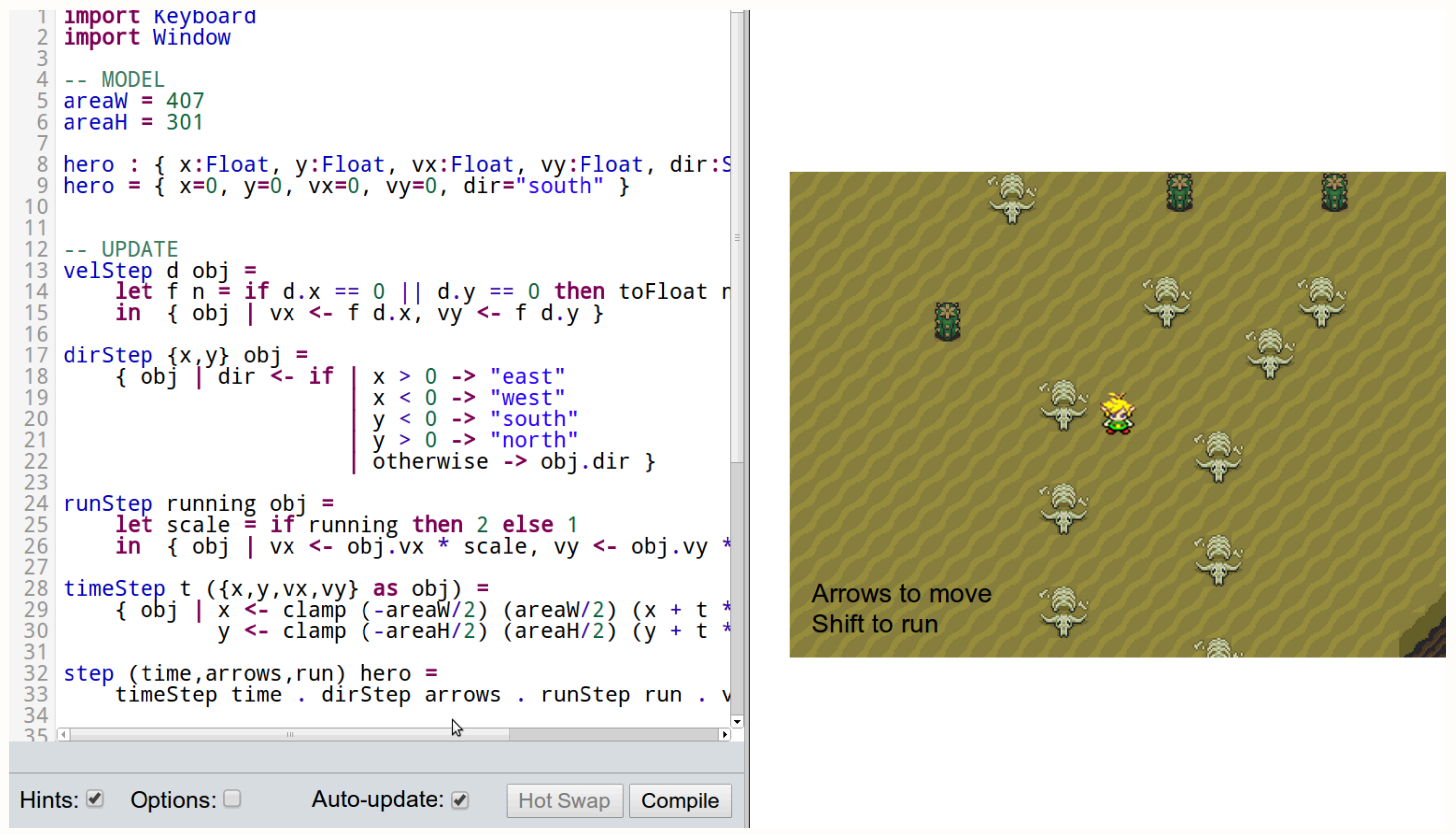
Figure 1: Elm logo

Evan Czaplicki, in his thesis of which the Elm is the product [1], makes claims — "[Elm] makes it *quick and easy* to create and combine text, images, and video into rich multimedia displays". While the evaluation of Elm's usability is not the focus of the thesis, rather, it is to establish a context for Functional Reactive Programming and describe the implementation details, he makes other usability claims without evidence — "[non-declarative frameworks for graphical user interfaces] mire programmers in the many small, nonessential details of handling user input and modifying the display.", "FRP makes GUI programming much more manageable", and in a section entitled *The Benefits of Functional GUIs*, "In Elm, divisions between data code, display code, and user interaction code arise fairly naturally, helping programmers write robust GUI code". If these claims are true, there is all the more evidence that Elm should be a language of choice for GUI programmers, but experiments must be done to determine this.

## 3. Background

The catalyst to this project was a talk about the concept of a highly reactive development environment — where changes in the code result in instantaneous updates to the runtime, 'on-the-fly'. This was presented in Bret Victor's "Inventing on Principle" [2]. He makes several assertions about the 'traditional' style of coding, one statement of which is that "most of the developer's time is spent looking at the code, blindly without an immediate connection to the thing they're making". He argues that "so much of creation is discovery, and you can't discover anything if you can't see what you're doing" — alluding to his earlier statement that the compile–run–debug cycle is much like this. Perhaps this rapid development cycle is not always suitable — in their 2013 paper [3] show that novices tend to "thrash" about, trying out many ideas that may or may not be a solution, and executing "poorly directed, ineffective problem solving . . . failing to realise they are doing it in good time, and fail to break out of it", whereas experts think much more about the problem at hand before proceeding with a solution.

## 4. Experimental Design



```
import Keyboard
import Window

-- MODEL
areaW = 407
areaH = 301

hero : { x:Float, y:Float, vx:Float, vy:Float, dir:S
hero = { x=0, y=0, vx=0, vy=0, dir="south" }


-- UPDATE
velStep d obj =
    let f n = if d.x == 0 || d.y == 0 then toFloat n
    in  { obj | vx <- f d.x, vy <- f d.y }

dirStep {x,y} obj =
    { obj | dir <- if | x > 0 -> "east"
                      | x < 0 -> "west"
                      | y < 0 -> "south"
                      | y > 0 -> "north"
                      | otherwise -> obj.dir }

runStep running obj =
    let scale = if running then 2 else 1
    in  { obj | vx <- obj.vx * scale, vy <- obj.vy *

timeStep t ({x,y,vx,vy} as obj) =
    { obj | x <- clamp (-areaW/2) (areaW/2) (x + t *
            y <- clamp (-areaH/2) (areaH/2) (y + t *

step (time,arrows,run) hero =
    timeStep time . dirStep arrows . runStep run . v
```

Hints: ✓ Options: ☐ Auto-update: ✓

Figure 2: A platform game in the Elm IDE

In the Elm IDE picture, there is the Elm code on the left pane, and the runtime on the right, which updates as the code is written. To determine whether Elm is 'easier' (for some operationalisation of easy) than JavaScript, I am designing the same task in both Elm and Javacript and am conducting user participant studies, video recording people completing the task of writing a small GUI. I am then going to work through a series of research — design — research — design . . . iterations in order to find out what may be causing the differences in cognitive load that I have observed speculatively in preliminary studies.

The task has to be designed in a way that is not idiomatic of either language – it is unfair to, say, choose a task that leverages the unique advantages of Elm's `Signal` type. A line such as `main = lift asText Mouse.position` will not be anywhere near as easy to achieve the same thing in JavaScript.

## 5. Further Work

In the interest of comparing research methodologies, I am curious as to the reliability of thematic analysis to highlight themes within such an activity as programming. I would like to extend the Elm IDE to log keypresses and mouse co–ordinates and obtain a different perspective on the same user studies to see if the themes identified correlate with the objective measures of cognitive load.

### References

[1] E. Czaplicki, "Elm: Concurrent FRP for Functional GUIs," 2012.

[2] B. Victor, "Inventing on principle," in *Proceedings of the Canadian University Software Engineering Conference (CUSEC)*, 2012.

[3] T. Lopez, M. Petre, and B. Nuseibeh, "Thrashing, tolerating and compromising in software development," in *Psychology of Programming Interest Group Annual Conference (PPIG-2012)* (Y. Jing, ed.), (London Metropolitan University, UK), pp. 70–81, London Metropolitan University, 21-23 November 2012.