Analysing cognitive load by extending an IDE to support input device logging of users during the activity of user–interface programming

Simon Buist
The University of Bath

April 18, 2014

Abstract

This dissertation provides a browser-based Integrated Development Environment [IDE] that logs input device data — for the purpose of performing user participation studies — whose implementation is described herein. The IDE is then used to conduct studies comparing the cognitive load experienced with two languages, Elm and JavaScript, during the completion of a basic task: modifying code to restrict the movement of a 2–D box from leaving the bounds of a frame. In order to do this comparison, the metric of `number of mouse clicks per code region` is selected, as an operationalisation of the concept of *thrashing* (Lopez et al., 2012), as being indicative of cognitive load during task completion. **The study found...**

Contents

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

Submitted by: Simon Buist

COPYRIGHT

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:

Acknowledgements

In no particular order, I would like to thank my dissertation supervisor, Dr. Leon Watts, for guiding me along the way — suggesting relevant literature, helping me stay on track and for lifting my spirits. I thank my parents Gail and Joseph for being a constant inspiration in my life, in what it means to be determined, and for believing in me. I am grateful to Remco Meeuwissen for his experience and input in writing the JavaScript task. I thank the lecturers and staff at The University of Bath for their help and time in providing me with the foundation and support that made this dissertation possible. My regards go to all those that kindly gave of their time to participate in the pilot studies and user studies.

1 Introduction

2 Project Plan

I will now explain my current plan for the project. Notice that I say current here – this may change throughout the course of the project: I may narrow in on a topic of interest, or branch out to investigate anomalous research findings.

I will be building the end product – the dissertation and software – via a process of iterations, much like an iterative Software Lifecycle. The Literature Survey is ongoing – throughout the whole project from beginning to end – feeding into all parts of the dissertation, and indeed this Proposal, as shown in the Gantt chart (Figure
1). The literature I choose is sometimes chosen to support points I wish to make, sometimes acting to guide my next area of research, reinforce findings, compare or contrast with other research, and probably many other things I have not yet thought of. Most importantly, I will be looking at who the paper/article etc. is cited by, preferring sources that are peer-reviewed.

As well as this literature research, I will also have an ongoing Product Literature Survey – looking at existing software out there that is related to my current area of interest.

Central to this idea of iteration is my desired method of performing user studies: I will first do what I have called a "Pilot" – a short and shallow trial User Study that focuses not on the research I'm concerned with, but instead the particular experimental design I would like to use in my actual User Study. By employing a Pilot I can hopefully get an idea of the nature of the experimental design – perhaps discovering any variables I had not previously considered that will require me to increase my sample size or simplify the experiment in order to mitigate their effect on the dependent variable I wish to test for. These are all problems discovered in Yates (2012) – including basic teething problems in getting the experiment to flow smoothly. In an even less detailed aspect, the pilot may allow me to look at what is out there. It may help to not look for anything in particular initially, and see what happens.

At this stage, with the help of discussion with my Project Supervisor, I have

some ideas about how to gather data in User Studies and these pilots could prove to be a useful testbed for such tools. I have a hypothesis that the novice developer "thrashing" Lopez et al. (2012) can be observed by shorter pauses between editing and experimentation, and I could measure this by way of measuring the mouse position relative to the IDE, clicks, and key-presses, using tools built-in to Elm and a bit of extension to stream this over the Internet to my storage facilities Czaplicki (2013b).

As you will see in the Gantt chart (Figure 1) I have included Testing & Implementation under the same heading as I will be doing Test Driven Development. My experience on Placement at PicoChip, my job as a Software Engineer at Altran and readings have helped me realise that this way of developing is time-saving and improves code quality by enforcing modularity in order to test it Martin (2008) and Hunt & Thomas (1999).

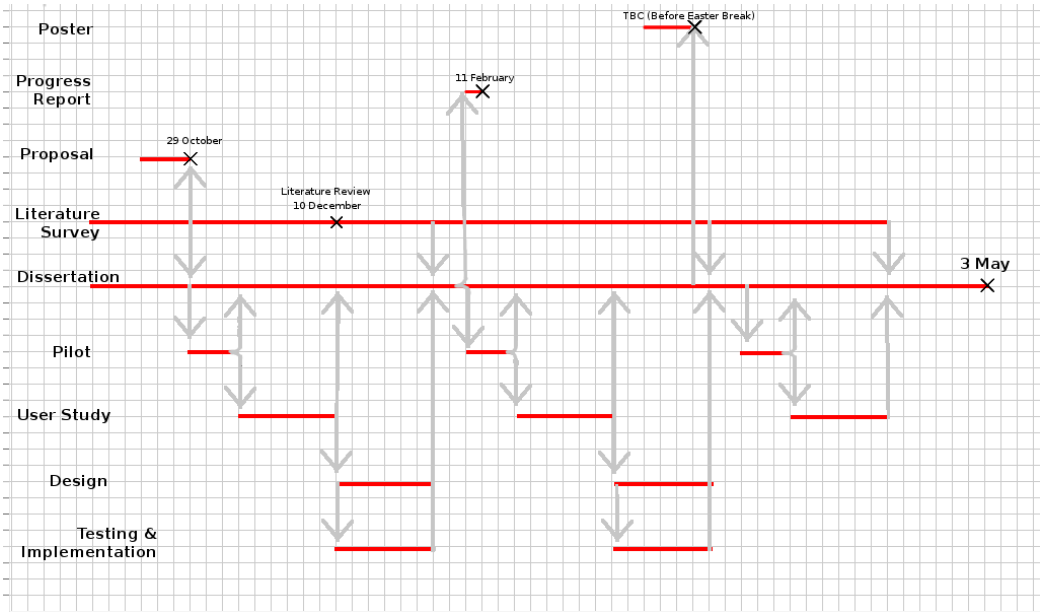

Figure 1: Gantt Chart

2.1 Required Resources

I will now talk about the resources I require for the completion of this dissertation, including the availability of these resources.

I will require users for my user study. These users must be proficient in at least one programming language (declarative programming languages are niche in and of themselves, never mind the discipline of programming, so some basic knowledge is required in order to see useful patterns in User Interface Design). Suitable candidates are First and Second Year Computer Science students from most Universities in the UK. Their availability is limited – Christmas holidays and coursework deadlines may mean that certain periods of the term are particularly busy for them. At Bath, suitable periods are therefore November, January to Mid February (inclusive), Mid-March to April (inclusive). It will be useful to procure free periods for other nearby Universities to hedge my bets, and to have a decent random assignment of users so I can make equivalent groups in my experiments.

The ACM Digital library, accessible via the Bath portal either from University or from home via Single-sign-on is a valuable resource for research papers, articles and references. The Cited-By feature will allow me to assert the popularity/ranking of each resource. Another valuable resource is the Psychology of Programming Interest Group, a "[group of] people from diverse communities to explore common interests in the psychological aspects of programming and in the computational aspects of psychology", with peer reviewed papers on particularly relevant topics to my area of research.

I will require regular access to the Internet, Emacs with haskell-mode installed and Elm version 0.10 Czaplicki (2013a). I will also need git for software source control, and bitbucket.org for online, private backups of my work. I require LaTeX to type up my dissertation, and have chosen texlive on Ubuntu 12.04.3 as my development environment of choice. The full development environment is installed at the house I am staying in, in Bath, on my laptop. I am also able to replicate this environment to a satisfactory level at Bath University on any computer with access via Putty/SSH or similar to LCPU, as all the above software can be installed and run on my Bath University account.

I am using Chromium Version 28.0.1500.71 Ubuntu 12.04 (28.0.1500.71-0ubuntu1.12.04.1) to run the Elm IDE, which is an important dependency that may cause problems in getting Users in User Studies to run a functionally equivalent browser. Only recent editions of Chrome, Chromium, Firefox, Opera and Safari (not Internet Explorer) support Elm web programs.

3 Ethical considerations

In conducting User Studies, I will be interacting with people and collecting data from them, so I must be considerate and mindful of those I talk to and the information I handle.

An Ethical Checklist such as the one Bath University uses as it's template Bath (2013) may assist my research such that I treat each participant with care and respect. I may learn from the discoveries made by others – in my reading, I came across a paper (also mentioned earlier) that highlighted concerns that participants under study had, and the paper detailed ways to mitigate these concerns so as to make the participant feel that are informed and safe Yates (2012).

4 Literature Survey

4.1 Introduction to the problem area

The problem area of user-interface programming, and more generally, the activity of programming in a context such as a software engineering environment, encompasses certain realms of interest. Through my survey of literature, my research has touched upon the above-mentioned terms, and I have discovered some thought-provoking problems that exist in the field of programming. The concept of 'Programming' embodies other concepts – art-forms, engineering processes, science, language, and mathematics, among others. To me, programming is a creative endeavour unlike any other – in which the programmer wields materials of no substance – the code – by manipulating symbols on a screen, which represent states in the machine being used. There are so many programming languages, and all languages (all that are Turing-complete) reduce to the same language – that of a Turing Machine. So, *why do we have so many programming languages?*.

Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy. (Perlis, 1982)

Different languages lend themselves to different ways of thinking about problems. They may place emphasis on one feature, for example list manipulation and hide others such as types. The language or programming environment

may make explicit the effect of changes as they are encoded, as opposed to queuing up a block of changes and the programmer having to initiate an update manually.

I would like to draw your attention in particular to the terms **Abstraction**, **Cognitive offloading**, **Feedback**, **Loss of information?/Augmented reality?**, **Thrashing**, and **"Programming blind"**. These, at current, are my topics of interest, and my literature review has up to this point been inextricably and heavily influenced by this.

4.2 What does it mean to be 'easy to use?'

In the process of surveying relevant (and sometimes irrelevant) literature to this dissertation, recurring conceptual patterns were observed – one particular instance of this is that several authors seem to lay victim to the trap of claiming their creation is "easy to use", "better", "simpler than x" without providing any supportive evidence of this.

Perhaps these are incidents of 'experimenter bias' – where the evaluator is naturally predisposed to a positive appraisal of their own findings. One way to avoid this is to have one set of people perform the data capture and another set perform the data analysis. Nevertheless, these patterns emerge, and present numerous opportunities for experimentation and subsequent evidence supporting or contradicting these claims. Experiments may see if the same conclusions are reached as the above-mentioned authors, accounting for the 'evaluator effect' (Hertzum & Jacobsen, 2001).

Whether this particular route is taken for experimentation hinges on pilot studies that will be conducted concurrently to the Literature Survey, each inextricably shaping the other's direction of investigation and inquiry.

The catalyst to this whole dissertation 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" (Victor, 2012). In his presentation Bret 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.

Evan Czaplicki, in his thesis of which Elm is the product (Czaplicki, 2012), makes similar 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.

And perhaps this rapid development cycle is not always suitable – in their 2012 paper, Lopez et al. 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 (Lopez et al., 2012).

4.3 Running User Studies

Perhaps a further direction of investigation may be running an experiment to spot whether or not Elm's auto-updating IDE lends to a lack of critical thinking – some operationalization may be *pauses reported as 'thinking' made during development* – where a pause is disambiguated as 'thinking' by the experimenter asking the participant why they did not perform any interaction with the computer for more than 10 seconds, and the participant reports that they were planning/designing/other similar activity. Along this line of thinking, a paper studying the relationship between speech pauses and cognitive load (Khawaja et al., 2008) found through studying 48 mixed gender participants that there is statistically significant indicators of cognitive load

through analysing pauses in speech. Perhaps this concept of pauses can be applied to the activity of programming. However, the planned method of disambiguating pauses via self-reporting (previously mentioned) would not be suitable according to these authors – "such measures can be either physically or psychologically intrusive and disrupt the normal flow of the interaction", although a paper cited by (Khawaja et al., 2008) itself claims that "although self-ratings may appear questionable, it has been demonstrated that people are quite capable of giving a numerical indication of their perceived mental burden (Gopher & Braune, 1984)". Indeed a pilot study by Fraser and Kölling (McKay & Kölling, 2012) structures the self-reporting by getting the users to evaluate an IDE as they use it using a set of subject-specific heuristics that they have designed. They showed that this customised set of heuristics helped guide the user more effectively than Nielsen's heuristics in evaluating usability, so one could develop a custom set of heuristics for evaluating the usability of Elm.

From the Elm thesis (Czaplicki, 2012), the language syntax and rapid feedback seem simple enough that it is conceivable (or at the very least, possible and of experimental interest) to allow the user to customise the UI layout to their liking. Letting the user shape the UI in concert with a UI programmer is covered the study of the interface development environment "Mobi-D" in millitary and medical applications (Puerta, 1997), with success in those fields. It may be worth speculating how Elm would fit into the development cycle that Puerta's paper outlines, as this may lend inspiration to potential user interface enhancements to the Elm IDE for A/B testing. It must be noted that there does not seem to be a re-emergence of Mobi-D since the paper was written, however.

My goal is to answer these questions. By way of conducting user studies, leveraging Elm with extensions to do A/B testing to illustrate it's effectiveness (or ineffectiveness) at enhancing User Interface Design.

Central to this idea of iteration is my desired method of performing user studies: I will first do what I have called a "Pilot" – a short and shallow trial User Study that focuses not on the research I'm concerned with, but instead the particular experimental design I would like to use in my actual User Study. By employing a Pilot I can hopefully get an idea of the nature of the experimental design – perhaps discovering any variables I had not previously considered that will require me to increase my sample size or simplify the

experiment in order to mitigate their effect on the dependent variable I wish to test for. These are all problems discovered in (Yates, 2012) – including basic teething problems in getting the experiment to flow smoothly. In an even less detailed aspect, the pilot may allow me to look at what is out there. It may help to not look for anything in particular initially, and see what happens.

At this stage, with the help of discussion with my Project Supervisor, I have some ideas about how to gather data in User Studies and these pilots could prove to be a useful testbed for such tools. I have a hypothesis that the novice developer "thrashing" (Lopez et al., 2012) can be observed by shorter pauses between editing and experimentation, and I could measure this by way of measuring the mouse position relative to the IDE, clicks, and key-presses, using tools built-in to Elm and a bit of extension to stream this over the Internet to my storage facilities (Czaplicki, 2013b).

5 Experimental methodology

5.1 AB Testing of the languages with the same IDE?

The primary direction I mentioned (as echoed in my Proposal) was doing AB testing of Elm vs. another language (e.g. JavaScript) (i.e. the language is the dependent variable) using the same Concurrent FRP IDE (the independent variable).

5.2 Test just the paradigm?

He also suggested a potential experiment to test just the paradigm, eliminating the IDE from the experiment above. Perhaps for a Pilot study.

5.2.1 Experiment process

1. Study question (e.g. Is it easy?)
2. Measurement concept (e.g. "Easy")

3. Operationalisation – taking a measurement concept and mapping it to something concrete (e.g. if completing a pre-defined task the user must complete takes < 5 steps, it is 'easy' – we can then compare instances of these studies given our definition of easy). This is much like mapping a design to an implementation, and there is a risk of losing information, or ending up with a mismatched concrete instance that does not represent the concept we wish to convey.
4. Do another operationalisation of our measurement concept – this allows us to get a different perspective of the same concept. (e.g. if total length of pauses during a 1 hour experiment is < 10 minutes, it is 'easy'). We do this to get 'coverage' of the measurement concept. It is a form of cross validation. If we see an overlap in the correlational results after analysis, we can make a stronger assertion that e.g. "language A is easier than language B.". The idea I am describing here is methodological decision-making.
5. Predict what will be the likely results of our experiments on the operationalised measurements. This is "feed forward validation".
6. Do the experiement.
7. Analyse the data. See if the data has patterns that correlate with the assertion I wish to make. I will be representing the raw data in some outcome measure – that is turning the raw data into a set of (or a single) value for comparison.
8. Does the data answer the study question I set out to ask? This is now "feed backwards validation".
9. Write-up including the 'nitty-gritty' of the user study, and a statement like "Given our definition of easy, our multiple operationalisations of the concept of easy show that this is in fact objectively true/false".

5.2.2 Pilots

We also spoke about ideas for pilot studies – asking "What might be surprising insights into declarative programming languages for User Interface Design – the case of Elm?".

Speak-aloud protocols where you prompt/facilitate the user to say what is on their mind when that e.g. pause for more than 10 seconds – a measurement I set out to look for during an experiment.

I might ask

- I notice you have paused for at least 10 seconds – why did you?
- I thought the code would do X, but it did Y.
- Why did you think it would do X?
- ...

I must ask the participant questions designed in a way that they are not leading.

Leon suggested I gather a rich data set, as it's difficult to take notes AND prompt the user during an experiment. SO difficult. Perhaps record video.

5.2.3 Actions for next meeting

Devise a Pilot study, answering these 3 questions:

1. What might I ask people to do?
2. How will I gather data?
3. How will I analyse the data?

Also see paper Leon will send me on "Thematic analysis & Psychology"

6 Pilot Study 1

Using per-participant questionnaire (See 9.5), I captured video & audio data of participants while the completed the task of extending a mario game to make mario fly

6.1 Hypotheses

- H1. ...
- H2. ...

6.1.1 Method

Using Thematic analysis (Braun & Clarke, 2006) to code the data...

6.2 Results

6.2.1 Observation 1

- Prompting *"What are you thinking about?"* etc. seemed to place additional cognitive load on the user as they spent longer resuming than when not prompted. This caused noise in assessing the actual cognitive load incurred during the completion of the **task**. Were the signs of struggling/undergoing difficulty due to simply not understanding the language, or were they due to the difficulty of the task?
- In particular, the majority of instances where the users paused turned out to be confusion as to the semantics & syntax of the language.

6.2.2 Model Adjustment 1

- Add tooltips that appear as the user places the keyboard cursor to the right of a token in the language.

6.2.3 Observation 2

- Sifting through 1-hour+ of video data capture for incidences of cognitive load is *HARD!*. Is there some programmatic way of narrowing the video data to points of interest?

6.2.4 Model Adjustment 2

- Track the user mouse and keyboard movements in a 3-tuple: `(Time t, (Mouse.x, Mouse.y), Keypress k)`
- It doesn't have to be implemented this way. I could extend **Model Adjustment 1** to define blocks of code as tokens in themselves, and capture how long the cursor is static on that particular token.

- Leon suggested a further refinement of this idea in order to further narrow the data (in fact, just capturing mouse & keyboard movements will result in an explosion of the volume of data – countrary to what I intend to achieve). His refinement was to define regions of interest in the code pane, and *only when the mouse/key cursor is in the region, do I capture data.*
- Use the `if cursor in region then log (Time t, (Mouse.x, Mouse.y), Keypress k)` functionality as a *lens* to focus on significant portions of video capture.

6.3 Further discussion

We then discussed some questions that might lead my direction of study in the next steps of my research:

- Is the mouse/cursor position a proxy for someone's attention as they carry out the task?
- Often when I'm coding I'll leave the cursor where it is but think about other regions of code. I don't necessarily move the keyboard/mouse cursor to the section of code I'm thinking about. Instead, I use it as a 'bookmark' to track what I'm currently implementing, and may scroll around to other parts.

6.3.1 We also discussed...

The result of the dissertation will be a list of observed cognitive easing/loading that each language produces for users, much like an advantage/disadvantage comparison:

Elm	JavaScript
+ ...	+ ...
+ ...	- ...
- ...	- ...

Elm	JavaScript
- ...	+ ...
+ ...	_

6.4 Actions

1. Design a task in JavaScript to go inside this adjusted model (incorporating Model Adjustment 1 and 2).

   This will require a degree of *"implementation juggling"* in order to find a balance of code-length/difficulty over the same task in Elm in such a way that is not creating noise in the thing being studied: Cognitive load.

   Keep the reactivity constant, compare the differences in ease between JS and Elm.

2. If time available, run another Pilot study on this task + adjusted model

6.4.1 Modifications to be made to the experimental methodology

Needs to be more objective! Why? What will I modify?

7 New Study! First, implement an IDE that logs input

7.1 Requirements

I will now identify what the requirements are for the project.

7.1.1 Functional Requirements

1. Write software to assist the capture of objective data to inform me of the user's activities as they use the Elm IDE.

1. The program must be able to work offline and later transfer collected data to me once a connection is resumed, collecting mouse and keyboard activity
   Priority: High

2. Perform Pilot and User Studies

    1. I must perform Pilot and User Studies in an iterative fashion, each one learning and building upon discoveries made in prior ones, starting vague and getting more and more focused on a particular facet of User Interface Design and/or Declarative programming as an activity.
       Priority: High
    2. I must use these studies to inform experimental and software design to disambiguate and filter data collected in the experiment, and to exercise hypotheses.
       Priority: High

7.1.2 Non-Functional Requirements

1. Source code

    1. The software must be written clearly and simply.
       Priority: High
    2. The software must have suitable, concise comments which explain the programs intent, but only where the code alone is not enough.
       Priority: High

2. Activity recording

    1. The program activity recording feature must not slow down the user's use of the IDE more than 1ms difference than without it.
       Priority: High
    2. There should be software to visualise the usage data
       Priority: Medium

7.2 Design

This is the chapter in which you review your design decisions at various levels and critique the design process.

More detail on what I will modify. How will I modify?

7.2.1 Experimental Design

Discussed progress made and what hypotheses to form that may usefully model cognitive load.

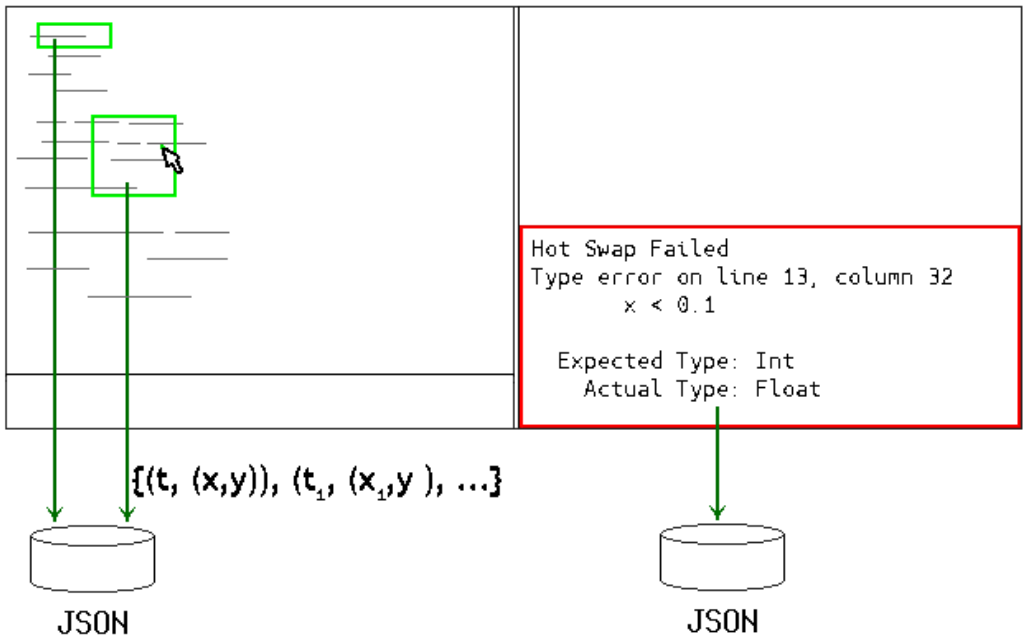

Figure 2: Extensions made to the Elm IDE

7.2.1.1 Progress since last meeting I have implemented full-screen mouse tracking that stores to a database a tuple:

```
(t, (x, y))
```

for every mouse move, producing a list in JSON (so it's more like `{{uniq-userid: {125125, (67, 321)}}, {uniq-userid: {125126, (67, 322)}} ...}`)

I am ready to demo this (See Action 1.)

The only issue worth tweaking is that user activity data is captured separately from the error output, so I will need to collate the data afterwards or find some way to feed it into the same data store.

7.2.1.2 Meeting Discussion 2 Hypotheses

1. Why the regions (*see green boxes in figure above*) I define in the code (to mouse-track e.g.) are meaningful
2. Frequency of semantically or syntactically incorrect errors made will differ as a function of the language under study

These need narrowing as they are too broad to test. Explode them into multiple, tighter hypotheses.

They are *valid* because they are *well-founded* – i.e. I have good reason to believe that # of errors made is an indication of cognitive load. I have good reason to believe that the selected regions will have more mouse activity (or whatever activity I suspect indicates higher cognitive load) as they are harder regions of code OR they pertain to achieving the set task.

7.2.1.3 Actions

1. Refine Mouse logging
    1. **DONE** Make it so that I can run arbitrary Elm code in the editor via a fileOpen operation
    2. **DONE** Make an Elm file that logs mouse movements ready to be loaded into the editor (See 9.4.4)
    3. **DONE** Load it into the editor and test it uploads to Firebase
    4. **DONE** Modify Generate.hs (See 9.4.5)

```
case (Elm.compile elmSrc) of
    Left jsSrc -> ...
    Right _ -> error "blah"
```

So that when we get an error, we timestamp and append it to a log file so this can later be collated with the Firebase to determine when errors were made

I'll need to insert a layer between `compile :: Snap()` and `serveHtml :: MonadSnap m => H.Html -> m ()` that performs the logging. It will have type signature `TypedHtml -> H.Html`

See the functions `compile` and `serveHtml` in Server.hs (See 9.4.3).

5. Make it so I can define regions in the mouse tracking – i.e. ONLY within a defined region is the mouse movement tracked e.g. `if mouse(x,y) in some2by2Square then Just mouse(x,y) else Nothing`

See https://github.com/spanners/laska/blob/master/Signals.elm

2. **DONE** Demo to supervisor

    1. Install on VPS (See 9.4.2)
    2. Run these:

```
git clone https://github.com/spanners/elm-lang.org
cd elm-lang.org
cabal install --bindir=.
```

3. **DONE** Design a task in JS and Elm

4. Define regions to select for logging activity. Why? Because:

    - Complex logic in code, OR
    - Relevant to task
    - Captures Thrash (keep on going over the same thing, e.g.). Errors made also captures thrash!

5. **DONE** Determine what to do with mouse (for example) data.

What makes code difficult to understand and work with?

- Bit twiddling?
- Declaring and defining simultaneously?
- Compound if/then/else statements?

[Programming is] manipulating symbols blindly ~ Bret Victor

Do a 2×2 study, defining regions in the code monitoring mouse clicks. Regions can either be simple/hard in complexity (exhibiting/not-exhibiting one of the above 'difficult' properties). Or code can be task-oriented or not, that is *the code does/does not need to be changed to achieve the completed task set for the user*:

Elm	-
Simple/Task	Hard/Task
Simple/Not-Task	Hard/Not-Task
JavaScript	-
Simple/Task	Hard/Task
Simple/Not-Task	Hard/Not-Task

Table 2: 2 × 2 study between-subjects

7.2.1.4 Study method Look at total and/or mean time in each of these areas for comparison.

My study will be **between-subjects** instead of within-subjects.

That is, I will study *different users* for different languages. If a user has completed the task in Elm, I can not have them complete the task in JavaScript, and vice-versa.

I will necessarily make a compromise here:

Between-subjects:

- I lose the ability to keep programmer competence as constant, thus it is a confounding variable
- I gain the ability to ignore learned-experience in completing the task – the participant is different every time so will not have done this task before, thus this is not a confounding variable.

Within-subjects is the converse of the above methodological properties

7.2.1.5 Actions

1. **DONE** Reorder divs so embedded div is on top of editor div.

   This turned out (I am fairly certain) to be due to codemirror.js binding mouse clicks. It was solved by using Elm's `Mouse.isDown`. Using `Mouse.isDown` has the added benefit of tracking mouse selects and drags, because it logs `(x,y)` when the mouse is down and `(x,y)` again when it is up.

2. **DONE** Create a task that features *Hard/Simple x Task/Not-task* (See 2)

3. ~~Implement *Region filtering* functionality so mouse activity is only logged when the clicks occur within defined region(s)~~

   I have instead defined bounding boxes that pertain to the regions I want to track as a mouse-data filter – that is, I capture all click data for the whole frame, and then filter it by comparing x,y co-ordinates with my bounding boxes. If it's in the box, keep it, otherwise discard.

4. **DONE** Integrate JS task into IDE

5. **DONE** Perform pilot study

6. **WIP** Visualise mouse data

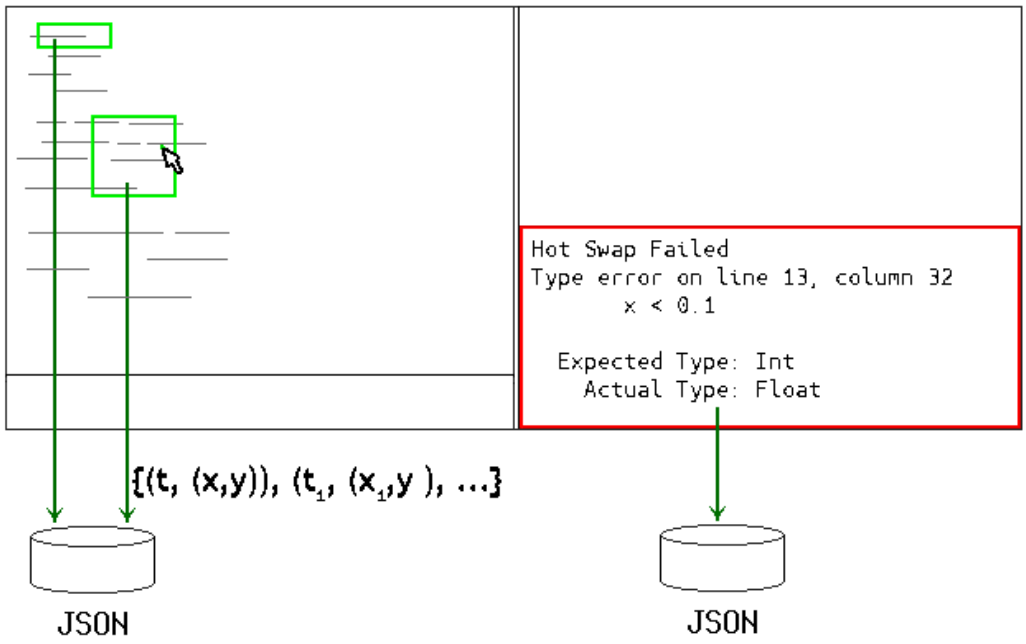

Figure 3: Extensions made to the Elm IDE

7.3 Implementation

Describe how I extended the Elm IDE

**This is the chapter in which you review the implementation and testing decisions and issues, and critique these processes. Code can be output inline using `some code`. For example, this code is inline: `public static int example = 0;` (I have used the character | as a delimiter, but any non-reserved character not in the code text can be used.) Code snippets can be output using the environment with the code given in the environment. For example, consider listing 5.1, below. Listing 5.1: Example code

Code listings are produced using the package "Listings". This has many useful options, so have a look at the package documentation for further ideas.**

7.4 Pilot Study 2

Using the Elm IDE

7.4.1 Observations

The task I chose for Pilot Study 1 was too difficult to capture the cognitive load incurred by the language itself for a given task, due to the difficulty of the task itself creating noise. I could improve this by simplifying the task, in a way that is 'language agnostic', i.e. that is not idiomatic of Elm or JavaScript (the two languages that I am comparing). Something like the following will never be that easy in JavaScript:

```
1 main = lift asText Mouse.position
```

Saw some things in Pilot Study 1, also in the use of the Elm IDE I extended, I saw some things before Pilot Study 2.

7.4.1.1 Hypotheses 1H.

7.4.2 Experiment

7.4.2.1 Method A 2×2×2 study, that is 2 Languages (Elm and JavaScript), 2 Region difficulties (Hard and Simple) and 2 Region relevances (Relevant and Not relevant) will be done to determine if the number of mouse clicks per region differ across variables.

7.4.3 Results

See Figure 4 for the visualisation of participant 15 completing the Elm version of the task.

Operationalisation of thrash (the concept), i.e. cementing the concept by a metric that models cognitive load (does it? we don't know – further work after the analysis of this may determine if it is a plausible indicator of cognitive load)

Leon suggested an improvement over this experimental method is to take people who are new, and train them up either in JS or Elm, and then run the same task. That way, their level of ability is comparable. (New as in never having used JS or Elm)

My current method creates quite a bit of noise in the data, because I rely on self-reported level of expertise in JS/Functional languages. I don't know how to modify the data to account for this. I could group the analyses into categories? I.e those who reported being experts at JS, those who reported never having used it, those who reported being experts in at least one FP language, and those who reported being new.

Talk about "phases" in a programmer's activities during task-completion:

(Not necessarily distinct and in sequence — more often interleaved)

1. Familiarisation – Where is the bit I need to change?
2. Narrowing in on the task once discovered – Oh I need to change `X`, but how?
3. Solved task
4. Playing (?)

Figure 4: Participant 15, Elm task (Overlaid with mouse clicks)

7.4.3.1 Mention the ways in which the study is flawed:

1. Self-reported expertise
2. Self-reported task completion
3. No way to be sure which error log pertains to which compile
4. Unique participant ID per Surveymonkey
5. Surveymonkey has taken my data hostage
6. window dimensions?!
7. Syntax reference 404
8. I did not capture window resizing

Not capturing window resizing is problematic – participant 15 (See Figure 4) very likely had a much shorter window height than I have used here. I suspect this is the case because of the cluster of mouse clicks in the same range of the x axis as the Compile button, but much futher up in the y axis, but I have no way to be sure as I did not log window dimensions.

7.4.3.2 Results

1. Describe data collected
2. How it was analysed (I aggregated regions and looked at number of clicks per region (Hard/Task, Hard/Not-Task, Simple/Task, Simple/Not-Task)*(Elm, JavaScript))
3. Presentation of data (summary means std dev.)
    1. χ^2 frequency analyses
    2. $2 \times 2 \times 2$ making 8 cells. My expected is an even distribution of clicks in each category, i.e. if I have 80 clicks in total across all groups, I expect to find 10 in each cell if there is no correlation.

Time (min)	Clicks
38.717217	183
8.034583	130
7.878533	39

Time (min)	Clicks
23.672500	25
29.754533	391
14.993517	78
48.960367	769
6.354050	71
7.878533	39
29.698267	501
40.302217	803
12.319317	65
17.106933	79
12.958300	119

Table 3: Session time and clicks per session for Elm task

Instead of χ^2, consider just using multiple regression with dummy variables (binary predictors) (See Table 4)

Condition	d_1	d_2	d_3	d_4	d_5	d_6	d_7
relevant × hard × Elm	1	0	0	0	0	0	0
relevant × hard × JS	0	1	0	0	0	0	0
relevant × easy × Elm	0	0	1	0	0	0	0
relevant × easy × JS	0	0	0	1	0	0	0

Condition	d_1	d_2	d_3	d_4	d_5	d_6	d_7
irrelevant × hard × Elm	0	0	0	0	1	0	0
irrelevant × hard × JS	0	0	0	0	0	1	0
irrelevant × easy × Elm	0	0	0	0	0	0	1
irrelevant × easy × JS	0	0	0	0	0	0	0

Table 4: Multiple regression with dummy variables (d1, d2..) (binary predictors)

- The d_n in the top row are the dummy variables, the values are the codes you assign them
- There are $n-1$ dummy variabes; one group is coded as all zeros – that's your reference group
- Why $n-1$? If there are 8 dummy codes, there are the same number of variables as conditions. the model's fully saturated and there are no degrees of freedom
  - Similar to $x_1 = 2, x_2 = 4, x_3 = 1$, and $y = 2$. then trying to solve for y

This is the chapter in which you review the outcomes, and critique the outcomes process.

You may include user evaluation here too.

8 Conclusions

This is the chapter in which you review the major achievements in the light of your original objectives, critique the process, critique your own learning and identify possible future work.

Bibliography

Bath, U. (2013) 'Research ethics framework checklist', [online] Available from: http://www.bath.ac.uk/research/pdf/ethics/EIRA1ethicsform.doc.

Braun, V. and Clarke, V. (2006) 'Using thematic analysis in psychology', *Qualitative Research in Psychology*, 3(2), pp. 77–101, [online] Available from: http://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa.

Czaplicki, E. (2013a) 'Elm 0.10', [online] Available from: http://elm-lang.org/blog/announce/0.10.elm.

Czaplicki, E. (2012) 'Elm: Concurrent FRP for Functional GUIs',

Czaplicki, E. (2013b) 'What is functional reactive programming?', [online] Available from: http://elm-lang.org/learn/What-is-FRP.elm (Accessed 1 October 2013).

Gopher, D. and Braune, R. (1984) 'On the psychophysics of workload: Why bother with subjective measures?', *Human Factors: The Journal of the Human Factors and Ergonomics Society*, SAGE Publications, 26(5), pp. 519–532.

Hertzum, M. and Jacobsen, N. E. (2001) 'The evaluator effect: A chilling fact about usability evaluation methods', *International Journal of Human-Computer Interaction*, Taylor & Francis, 13(4), pp. 421–443.

Hunt, A. and Thomas, D. (1999) *The pragmatic programmer: from journeyman to master*, Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc.

Khawaja, M. A., Ruiz, N. and Chen, F. (2008) 'Think before you talk: An empirical study of relationship between speech pauses and cognitive load', In *Proceedings of the 20th australasian conference on computer-human interaction: Designing for habitus and habitat*, OZCHI '08, New York, NY,

USA, ACM, pp. 335–338, [online] Available from: http://doi.acm.org/10.1145/1517744.1517814.

Lopez, T., Petre, M. and Nuseibeh, B. (2012) 'Thrashing, tolerating and compromising in software development', In Jing, Y. (ed.), *Psychology of Programming Interest Group Annual Conference (PPIG-2012)*, London Metropolitan University, UK, London Metropolitan University, pp. 70–81.

Martin, R. C. (2008) *Clean code: A handbook of agile software craftsmanship*, 1st ed. Upper Saddle River, NJ, USA, Prentice Hall PTR.

McKay, F. and Kölling, M. (2012) 'Evaluation of subject-specific heuristics for initial learning environments: A pilot study', In *Proceedings of the 24th Psychology of Programming Interest Group Annual Conference 2012*, London Metropolitan University, pp. 128–138.

Perlis, A. J. (1982) 'Epigrams on programming', *SIGPLAN Notices*, 17(9), pp. 7–13.

Puerta, A. R. (1997) 'A Model-Based Interface Development Environment', *IEEE Softw.*, Los Alamitos, CA, USA, IEEE Computer Society Press, 14(4), pp. 40–47, [online] Available from: http://dx.doi.org/10.1109/52.595902.

Victor, B. (2012) 'Inventing on principle', In *Proceedings of the canadian university software engineering conference (CUSEC)*, [online] Available from: http://vimeo.com/36579366 (Accessed 15 March 2014).

Yates, R. (2012) 'Conducting field studies in software engineering: An experience report', In Jing, Y. (ed.), *Psychology of Programming Interest Group Annual Conference (PPIG-2012)*, London Metropolitan University, UK, London Metropolitan University, pp. 82–85.

9 Appendices

Appendix A

9.1 Design Diagrams

Appendix B

9.2 Mouse click visualisation

9.3 Raw results output

9.3.1 firebase-mouseclick-data.json


```
{
  "js" : {
    "12" : {
      "-JKVwXgnfg4POT9MArjy" : {
        "t" : 1397490853983,
        "y" : 0,
        "x" : 0
      },
      "-JKVwXl8I_bLZNqxP36c" : {
        "t" : 1397490854551,
        "y" : 444,
        "x" : 417
      },
      "-JKVwXj0LD-uRKDsT2Om" : {
        "t" : 1397490854503,
        "y" : 444,
        "x" : 417
      }
    }
  },
  "elm" : {
```

Figure 5: Paricipant 15, Elm task (Overlay)

```
    "33" : {
      "-JKRgqJIRPH2EG-edZb5" : {
        "t" : 1397419631953,
        "y" : 249.59999084472656,
        "x" : 48.79999923706055
      },
      "-JKRhRQi31pr9AP1p1Rr" : {
        "t" : 1397419787709,
        "y" : 294.3999938964844,
        "x" : 152.8000030517578
      },
      "-JKRffOszNGfgwNdnO_X" : {
        "t" : 1397419324585,
        "y" : 608,
        "x" : 346.3999938964844
      }
    }
  }
}
```

9.3.2 error_log.json

```
{
   "2014-04-11 21:14:32.141743994+01:00":{
      "Parse error at (line 37, column 44):",
      "unexpected 'a'",
      "expecting \"{-\", \" \" or end of input"
   },
   "2014-04-11 21:35:41.694436974+01:00":{
      "Type error on line 27, column 16 to 77:",
      "        (min (max x (-hHeight)) hHeight,",
      "         (min (max y (-hWidth)),hWidth))",
      "",
      "   Expected Type: Float",
      "     Actual Type: (Float -> Float, Float)"
   },
   "2014-04-12 00:19:14.945129550+01:00":{
```

```
        "Parse error at (line 1, column 1):",
        "unexpected \"<\"",
        "expecting reserved word 'module', reserved word 'import'
                or at least one datatype or variable definition"
    },
    "2014-04-12 00:19:21.553633974+01:00":{
        "Parse error at (line 1, column 1):",
        "unexpected \"/\"",
        "expecting reserved word 'module', reserved word 'import'
                or at least one datatype or variable definition"
    },
    "2014-04-12 00:19:27.053901481+01:00":{
        "Parse error at (line 1, column 1):",
        "unexpected \"/\"",
        "expecting reserved word 'module', reserved word 'import'
                or at least one datatype or variable definition"
    },
}
```

Appendix C

9.4 Code

- All code available here: https://github.com/spanners/elm-lang.org.
  - This is a modifed version of Evan Czaplicki's elm-lang.org code, available here: https://github.com/elm-lang/elm-lang.org
- Elm task here: http://mouth.crabdance.com:8000/edit/task/MovingBox.elm
- Javascript task here: http://mouth.crabdance.com:8000/_edit/task/MovingBox.js

9.4.1 LICENSE

```
Copyright (c) 2012-2013 Evan Czaplicki
```

```
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.

    * Redistributions in binary form must reproduce the above
      copyright notice, this list of conditions and the following
      disclaimer in the documentation and/or other materials provided
      with the distribution.

    * Neither the name of Evan Czaplicki nor the names of other
      contributors may be used to endorse or promote products derived
      from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

9.4.2 install_elm.sh

```
1 #!/bin/bash
2
3 sudo apt-get install libgl1-mesa-dev libglc-dev freeglut3-dev
4 sudo apt-get install libedit-dev libglw1-mesa libglw1-mesa-dev
5 sudo apt-get install ghc
6 wget http://www.haskell.org/ghc/dist/7.6.3/ghc-7.6.3-src.tar.bz2
```

```
tar xjf ghc-7.6.3-src.tar.bz2
cd ghc-7.6.3/mk
cp build.mk.sample build.mk
sed -i 's/^#BuildFlavour = quick/BuildFlavour = quick/' build.mk
cd ..
./configure
make -j 8
sudo make install
cd
platform="2013.2.0.0/haskell-platform-2013.2.0.0.tar.gz"
wget "http://lambda.haskell.org/platform/download/"$platform
tar xzvf haskell-platform-2013.2.0.0.tar.gz
cd haskell-platform-2013.2.0.0
./configure
make
sudo make install
cabal update
cabal install cabal-install
cabal install elm
cabal install elm-server
exit 0
```

9.4.3 Server.hs

```
{-# OPTIONS_GHC -W #-}
{-# LANGUAGE OverloadedStrings, DeriveDataTypeable #-}
module Main where

import Data.Monoid (mempty)
import qualified Data.ByteString as BS
import qualified Data.ByteString.Char8 as BSC
import qualified Elm.Internal.Utils as Elm
import Control.Applicative
import Control.Monad.Error

import Text.Blaze.Html5 ((!))
import qualified Text.Blaze.Html5 as H
```

```
14  import qualified Text.Blaze.Html5.Attributes as A
15  import qualified Text.Blaze.Html.Renderer.Utf8 as BlazeBS
16  import Text.Regex
17
18  import Snap.Core
19  import Snap.Http.Server
20  import Snap.Util.FileServe
21  import System.Console.CmdArgs
22  import System.FilePath as FP
23  import System.Process
24  import System.Directory
25  import GHC.Conc
26
27  import qualified Elm.Internal.Paths as Elm
28  import qualified Generate
29  import qualified Editor
30
31  data Flags = Flags
32    { port :: Int
33    } deriving (Data,Typeable,Show,Eq)
34
35  flags :: Flags
36  flags = Flags
37    { port = 8000 &= help "set the port of the server"
38    }
39
40  -- | Set up the server.
41  main :: IO ()
42  main = do
43    setNumCapabilities =<< getNumProcessors
44    putStrLn "Initializing Server"
45    getRuntimeAndDocs
46    setupLogging
47    precompile
48    cargs <- cmdArgs flags
49    httpServe (setPort (port cargs) defaultConfig) $
50        ifTop (serveElm "public/Empty.elm")
51        <|> route [ ("try", serveHtml Editor.empty)
```

```
                  , ("edit", edit)
                  , ("_edit", jsEdit)
                  , ("code", code)
                  , ("_code", jsCode)
                  , ("compile", compile)
                  , ("_compile", jsCompile)
                  , ("hotswap", hotswap)
                  ]
      <|> error404

error404 :: Snap ()
error404 =
    do modifyResponse $ setResponseStatus 404 "Not found"
       serveElm "public/build/Error404.elm"

serveElm :: FilePath -> Snap ()
serveElm = serveFileAs "text/html; charset=UTF-8"

logAndServeJS :: MonadSnap m => H.Html -> m ()
logAndServeJS = serveHtml

logAndServeHtml :: MonadSnap m => (H.Html, Maybe String) -> m ()
logAndServeHtml (html, Nothing)  = serveHtml html
logAndServeHtml (html, Just err) =
    do timeStamp <- liftIO $ readProcess "date" ["--rfc-3339=ns"] ""
       liftIO $ appendFile "error_log.json" $ "{\"" ++ (init timeStamp)
                                                    ++ "\","
                                                    ++ (show (lines err))
                                                    ++ "},"
       setContentType "text/html" <$> getResponse
       writeLBS (BlazeBS.renderHtml html)


embedJS :: MonadSnap m => H.Html -> String -> m ()
embedJS js participant =
    do
       elmSrc <- liftIO $ readFile "EmbedMeJS.elm"
       setContentType "text/html" <$> getResponse
```

```
90          writeLBS (BlazeBS.renderHtml (embedMe elmSrc js participant))
91
92  embedHtml :: MonadSnap m => H.Html -> String -> m ()
93  embedHtml html participant =
94      do elmSrc <- liftIO $ readFile "EmbedMeElm.elm"
95         setContentType "text/html" <$> getResponse
96         writeLBS (BlazeBS.renderHtml (embedMe elmSrc html participant))
97
98  serveHtml :: MonadSnap m => H.Html -> m ()
99  serveHtml html =
100     do setContentType "text/html" <$> getResponse
101        writeLBS (BlazeBS.renderHtml html)
102
103 hotswap :: Snap ()
104 hotswap = maybe error404 serve =<< getParam "input"
105     where
106       serve code =
107           do setContentType "application/javascript" <$> getResponse
108              writeBS . BSC.pack . Generate.js $ BSC.unpack code
109
110 jsCompile :: Snap ()
111 jsCompile = maybe error404 serve =<< getParam "input"
112     where
113       serve = logAndServeJS . Generate.logAndJS "Compiled JS" . BSC.unpack
114
115 compile :: Snap ()
116 compile = maybe error404 serve =<< getParam "input"
117     where
118       serve = logAndServeHtml . Generate.logAndHtml "Compiled Elm" . BSC.unpack
119
120 edit :: Snap ()
121 edit = do
122   participant <- BSC.unpack . maybe "" id <$> getParam "p"
123   cols <- BSC.unpack . maybe "50%,50%" id <$> getQueryParam "cols"
124   withFile (Editor.ide cols participant)
125
126 jsEdit :: Snap ()
127 jsEdit = do
```

```
128   participant <- BSC.unpack . maybe "" id <$> getParam "p"
129   cols <- BSC.unpack . maybe "50%,50%" id <$> getQueryParam "cols"
130   withFile (Editor.jsIde cols participant)
131
132 code :: Snap ()
133 code = do
134   participant <- BSC.unpack . maybe "" id <$> getParam "p"
135   embedWithFile Editor.editor participant
136
137 jsCode :: Snap ()
138 jsCode = do
139   participant <- BSC.unpack . maybe "" id <$> getParam "p"
140   jsEmbedWithFile Editor.jsEditor participant
141
142 embedee :: String -> String -> H.Html
143 embedee elmSrc participant =
144     H.span $ do
145       case Elm.compile elmSrc of
146         Right jsSrc -> do
147             embed $ H.preEscapedToMarkup (subRegex oldID jsSrc newID)
148         Left err ->
149             H.span ! A.style "font-family: monospace;" $
150             mapM_ addSpaces (lines err)
151       script "/fullScreenEmbedMe.js"
152   where addSpaces line = H.preEscapedToMarkup (Generate.addSpaces line) >> H.br
153         oldID = mkRegex "var user_id = \"1\";"
154         newID = ("var user_id = " ++ participant ++ "+'';" :: String)
155         jsAttr = H.script ! A.type_ "text/javascript"
156         script jsFile = jsAttr ! A.src jsFile $ mempty
157         embed jsCode = jsAttr $ jsCode
158
159 embedMe :: String -> H.Html -> String -> H.Html
160 embedMe elmSrc target participant = target >> (embedee elmSrc participant)
161
162 jsEmbedWithFile :: (FilePath -> String -> H.Html) -> String -> Snap ()
163 jsEmbedWithFile handler participant = do
164   path <- BSC.unpack . rqPathInfo <$> getRequest
165   let file = "public/" ++ path
```

```
166   exists <- liftIO (doesFileExist file)
167   if not exists then error404 else
168       do content <- liftIO $ readFile file
169          embedJS (handler path content) participant
170
171 embedWithFile :: (FilePath -> String -> H.Html) -> String -> Snap ()
172 embedWithFile handler participant = do
173   path <- BSC.unpack . rqPathInfo <$> getRequest
174   let file = "public/" ++ path
175   exists <- liftIO (doesFileExist file)
176   if not exists then error404 else
177       do content <- liftIO $ readFile file
178          embedHtml (handler path content) participant
179
180 withFile :: (FilePath -> String -> H.Html) -> Snap ()
181 withFile handler = do
182   path <- BSC.unpack . rqPathInfo <$> getRequest
183   let file = "public/" ++ path
184   exists <- liftIO (doesFileExist file)
185   if not exists then error404 else
186       do content <- liftIO $ readFile file
187          serveHtml $ handler path content
188
189 setupLogging :: IO ()
190 setupLogging =
191     do createDirectoryIfMissing True "log"
192        createIfMissing "log/access.log"
193        createIfMissing "log/error.log"
194     where
195       createIfMissing path = do
196         exists <- doesFileExist path
197         when (not exists) $ BS.writeFile path ""
198
199 -- | Compile all of the Elm files in public/, put results in public/build/
200 precompile :: IO ()
201 precompile =
202   do setCurrentDirectory "public"
203      files <- getFiles True ".elm" "."
```

```
204       forM_ files $ \file -> rawSystem "elm"
205                                ["--make","--runtime=/elm-runtime.js",file]
206       htmls <- getFiles False ".html" "build"
207       mapM_ adjustHtmlFile htmls
208       setCurrentDirectory ".."
209     where
210       getFiles :: Bool -> String -> FilePath -> IO [FilePath]
211       getFiles skip ext directory =
212           if skip && "build" `elem` map FP.dropTrailingPathSeparator
213                                        (FP.splitPath directory)
214             then return [] else
215             (do contents <- map (directory </>) `fmap`
216                                                  getDirectoryContents directory
217                 let files = filter ((ext==) . FP.takeExtension) contents
218                     directories  = filter (not . FP.hasExtension) contents
219                 filess <- mapM (getFiles skip ext) directories
220                 return (files ++ concat filess))
221
222 getRuntimeAndDocs :: IO ()
223 getRuntimeAndDocs = do
224   writeFile "resources/elm-runtime.js" =<< readFile Elm.runtime
225   writeFile "resources/docs.json" =<< readFile Elm.docs
226
227 adjustHtmlFile :: FilePath -> IO ()
228 adjustHtmlFile file =
229   do src <- BSC.readFile file
230      let (before, after) = BSC.breakSubstring "<title>" src
231      BSC.writeFile (FP.replaceExtension file "elm") $
232         BSC.concat [before, style, after]
233      removeFile file
234
235 style :: BSC.ByteString
236 style =
237     "<style type=\"text/css\">\n\
238     \  a:link {text-decoration: none; color: rgb(15,102,230);}\n\
239     \  a:visited {text-decoration: none}\n\
240     \  a:active {text-decoration: none}\n\
241     \  a:hover {text-decoration: underline; color: rgb(234,21,122);}\n\
```

```
    \  body { font-family: \"Lucida Grande\",\"Trebuchet MS\",\
    \  \"Bitstream Vera Sans\",Verdana,Helvetica,sans-serif !important; }\n\
    \  p, li { font-size: 14px !important;\n\
    \          line-height: 1.5em !important; }\n\
    \</style>"
```

9.4.4 Editor.hs

```haskell
{-# LANGUAGE OverloadedStrings #-}
module Editor (editor,jsEditor,jsIde,ide,empty) where

import Data.Monoid (mempty)
import Text.Blaze.Html
import qualified Text.Blaze.Html5 as H
import qualified Text.Blaze.Html5.Attributes as A
import Network.HTTP.Base (urlEncode)
import qualified System.FilePath as FP

import qualified Elm.Internal.Utils as Elm
import Data.Maybe (fromMaybe)

import Generate (addSpaces)


-- | Display an editor and the compiled result side-by-side.
jsIde :: String -> String -> FilePath -> String -> Html
jsIde cols participant fileName code =
    jsIdeBuilder cols
                 participant
                 ("JS Editor: " ++ FP.takeBaseName fileName)
                 fileName
                 ("/_compile?input=" ++ urlEncode code)

-- | Display an editor and the compiled result side-by-side.
ide :: String -> String -> FilePath -> String -> Html
ide cols participant fileName code =
    ideBuilder cols
```

```
30                participant
31                ("Elm Editor: " ++ FP.takeBaseName fileName)
32                fileName
33                ("/compile?input=" ++ urlEncode code)
34
35  -- | Display an editor and the compiled result side-by-side.
36  empty :: Html
37  empty = ideBuilder "50%,50%" "1" "Try Elm" "Empty.elm" "/Try.elm"
38
39  jsIdeBuilder :: String -> String -> String -> String -> String -> Html
40  jsIdeBuilder cols participant title input output =
41      H.docTypeHtml $ do
42        H.head . H.title . toHtml $ title
43        preEscapedToMarkup $
44           concat [ "<frameset cols=\"" ++ cols ++ "\">\n"
45            , "  <frame name=\"input\" src=\"/_code/", input, "?p=",
46              participant, "\" />\n"
47            , "  <frame name=\"output\" src=\"", output, "\" />\n"
48            , "</frameset>" ]
49
50
51  ideBuilder :: String -> String -> String -> String -> String -> Html
52  ideBuilder cols participant title input output =
53      H.docTypeHtml $ do
54        H.head . H.title . toHtml $ title
55        preEscapedToMarkup $
56           concat [ "<frameset cols=\"" ++ cols ++ "\">\n"
57                  , "  <frame name=\"input\" src=\"/code/", input, "?p=",
58                    participant, "\" />\n"
59                  , "  <frame name=\"output\" src=\"", output, "\" />\n"
60                  , "</frameset>" ]
61
62  -- | list of themes to use with CodeMirror
63  themes :: [String]
64  themes = [ "ambiance", "blackboard", "cobalt", "eclipse"
65           , "elegant", "erlang-dark", "lesser-dark", "monokai", "neat", "night"
66           , "rubyblue", "solarized", "twilight", "vibrant-ink", "xq-dark" ]
67
```

```haskell
jsFiles :: [AttributeValue]
jsFiles = [ "/codemirror-3.x/lib/codemirror.js"
          , "/codemirror-3.x/mode/elm/elm.js"
          , "/misc/showdown.js"
          , "/misc/editor.js?0.11" ]

jsFiles2 :: [AttributeValue]
jsFiles2 = [ "/codemirror-3.x/lib/codemirror.js"
          , "/codemirror-3.x/mode/javascript/javascript.js"
          , "/misc/showdown.js"
          , "/misc/editor.js?0.11" ]

jsEditor :: FilePath -> String -> Html
jsEditor filePath code =
    H.html $ do
      H.head $ do
        H.title . toHtml $ "JS Editor: " ++ FP.takeBaseName filePath
        H.link ! A.rel "stylesheet"
               ! A.href "/codemirror-3.x/lib/codemirror.css"
        mapM_ themeAttr themes
        H.link ! A.rel "stylesheet" ! A.type_ "text/css"
                                    ! A.href "/misc/editor.css"
        mapM_ script jsFiles2
        script "/elm-runtime.js?0.11"
        script "http://cdn.firebase.com/v0/firebase.js"
      H.body $ do
        H.form ! A.id "inputForm"
               ! A.action "/_compile"
               ! A.method "post"
               ! A.target "output" $ do
           H.div ! A.id "editor_box" $
             H.textarea ! A.name "input" ! A.id "input" $ toHtml ('\n':code)
           H.div ! A.id "options" $ do
             bar "documentation" docs
             bar "editor_options" editorOptions
             bar "always_on" (buttons >> options)
        embed "initEditor();"
  where themeAttr theme = H.link ! A.rel "stylesheet"
```

```
                                    ! A.href (toValue ("/codemirror-3.x/theme/"
                                                       ++ theme
                                                       ++ ".css" :: String))
        jsAttr = H.script ! A.type_ "text/javascript"
        script jsFile = jsAttr ! A.src jsFile $ mempty
        embed jsCode = jsAttr $ jsCode

-- | Create an HTML document that allows you to edit and submit Elm code
--   for compilation.
editor :: FilePath -> String -> Html
editor filePath code =
    H.html $ do
      H.head $ do
        H.title . toHtml $ "Elm Editor: " ++ FP.takeBaseName filePath
        H.link ! A.rel "stylesheet"
               ! A.href "/codemirror-3.x/lib/codemirror.css"
        mapM_ themeAttr themes
        H.link ! A.rel "stylesheet" ! A.type_ "text/css"
                                    ! A.href "/misc/editor.css"
        mapM_ script jsFiles
        script "/elm-runtime.js?0.11"
        script "http://cdn.firebase.com/v0/firebase.js"
      H.body $ do
        H.form ! A.id "inputForm"
               ! A.action "/compile"
               ! A.method "post"
               ! A.target "output" $ do
          H.div ! A.id "editor_box" $
            H.textarea ! A.name "input" ! A.id "input" $ toHtml ('\n':code)
          H.div ! A.id "options" $ do
            bar "documentation" docs
            bar "editor_options" editorOptions
            bar "always_on" (buttons >> options)
        embed "initEditor();"
  where themeAttr theme = H.link ! A.rel "stylesheet"
                                 ! A.href (toValue ("/codemirror-3.x/theme/"
                                                    ++ theme
                                                    ++ ".css" :: String))
```

```
144         jsAttr = H.script ! A.type_ "text/javascript"
145         script jsFile = jsAttr ! A.src jsFile $ mempty
146         embed jsCode = jsAttr $ jsCode
147
148 bar :: AttributeValue -> Html -> Html
149 bar id' body = H.div ! A.id id' ! A.class_ "option" $ body
150
151 buttons :: Html
152 buttons = H.div ! A.class_ "valign_kids"
153                 ! A.style "float:right; padding-right: 6px;"
154                 $ compileButton
155       where
156         compileButton =
157             H.input
158                  ! A.type_ "button"
159                  ! A.id "compile_button"
160                  ! A.value "Compile"
161                  ! A.onclick "compile()"
162                  ! A.title "Ctrl-Enter: change program behavior \
163                            \but keep the state"
164
165 options :: Html
166 options = H.div ! A.class_ "valign_kids"
167                 ! A.style "float:left; padding-left:6px; padding-top:2px;"
168                 $ (docs' >> opts)
169     where
170       docs' =
171         H.span  ! A.title "Show documentation and types." $ "Hints:" >>
172             H.input ! A.type_ "checkbox"
173                     ! A.id "show_type_checkbox"
174                     ! A.onchange "showType(this.checked);"
175
176       opts =
177         H.span  ! A.title "Show editor options."
178                 ! A.style "padding-left: 12px;" $ "Options:" >>
179             H.input ! A.type_ "checkbox"
180                     ! A.id "options_checkbox"
181                     ! A.onchange "showOptions(this.checked);"
```

```
182
183 editorOptions :: Html
184 editorOptions = theme >> zoom >> lineNumbers
185     where
186       optionFor :: String -> Html
187       optionFor text =
188           H.option ! A.value (toValue text) $ toHtml text
189
190       theme =
191           H.select ! A.id "editor_theme"
192                    ! A.onchange "setTheme(this.value)"
193                    $ mapM_ optionFor themes
194
195       zoom =
196           H.select ! A.id "editor_zoom"
197                    ! A.onchange "setZoom(this.options[this.selectedIndex].\
198                                 \innerHTML)"
199                    $ mapM_ optionFor ["100%", "80%", "150%", "200%"]
200
201       lineNumbers = do
202         H.span ! A.style "padding-left: 16px;" $ "Line Numbers:"
203         H.input ! A.type_ "checkbox"
204                 ! A.id "editor_lines"
205                 ! A.onchange "showLines(this.checked);"
206
207 docs :: Html
208 docs = tipe >> desc
209     where
210       tipe = H.div ! A.class_ "type" $ message >> more
211
212       message = H.div !
213                   A.style "position:absolute; left:4px; right:36px;\
214                            \overflow:hidden; text-overflow:ellipsis;" $ ""
215
216       more = H.a ! A.id "toggle_link"
217                  ! A.style "display:none; float:right;"
218                  ! A.href "javascript:toggleVerbose();"
219                  ! A.title "Ctrl+H"
```

```
                $ ""

      desc = H.div ! A.class_ "doc"
                   ! A.style "display:none;"
                   $ ""

```

9.4.5 Generate.hs

```
{-# LANGUAGE OverloadedStrings #-}
module Generate (logAndJS, logAndHtml, html, js, addSpaces) where

import Data.Monoid (mempty)
import Data.Maybe (fromMaybe)
import Text.Blaze (preEscapedToMarkup)
import Text.Blaze.Html5 ((!))
import qualified Text.Blaze.Html5 as H
import qualified Text.Blaze.Html5.Attributes as A

import qualified Elm.Internal.Utils as Elm
import Utils

logAndJS :: String -> String -> H.Html
logAndJS name src = getJSPage name src

logAndHtml :: String -> String -> (H.Html, Maybe String)
logAndHtml name src =
    let elmname = "Elm." ++ fromMaybe "Main" (Elm.moduleName src)
    in
      case Elm.compile src of
          Right jsSrc -> do
              (getHtmlPage name elmname jsSrc, Nothing)
          Left err -> do
              (getErrPage name err, Just err)

getJSPage :: String -> String -> H.Html
getJSPage name jsSrc =
```

```
29    H.docTypeHtml $ do
30        H.head $ do
31          H.meta ! A.charset "UTF-8"
32          H.title . H.toHtml $ name
33          H.link ! A.rel "stylesheet" ! A.type_ "text/css"
34                                      ! A.href "/misc/js.css"
35          script "/pixi.js"
36        H.body $ do
37          H.div ! A.style "width: 400px; height: 400px; position:\
38                          \ absolute; top: 0; left: 0; opacity: 0;" $ mempty
39          jsAttr $ preEscapedToMarkup jsSrc
40   where jsAttr = H.script ! A.type_ "text/javascript"
41         script jsFile = jsAttr ! A.src jsFile $ mempty
42         embed jsCode = jsAttr $ jsCode
43
44 getHtmlPage :: String -> String -> String -> H.Html
45 getHtmlPage name elmname jsSrc =
46   H.docTypeHtml $ do
47       H.head $ do
48         H.meta ! A.charset "UTF-8"
49         H.title . H.toHtml $ name
50         H.style ! A.type_ "text/css" $ preEscapedToMarkup
51           ("a:link {text-decoration: none; color: rgb(15,102,230);}\n\
52            \a:visited {text-decoration: none}\n\
53            \a:active {text-decoration: none}\n\
54            \a:hover {text-decoration: underline; \
55            \color: rgb(234,21,122);}" :: String)
56       H.body $ do
57         let js = H.script ! A.type_ "text/javascript"
58             runFullscreen =
59                 "var runningElmModule = Elm.fullscreen(" ++ elmname
60                                                          ++ ")"
61         js ! A.src (H.toValue ("/elm-runtime.js?0.11" :: String)) $ ""
62         js $ preEscapedToMarkup jsSrc
63         js $ preEscapedToMarkup runFullscreen
64
65 getErrPage :: String -> String -> H.Html
66 getErrPage name err =
```

```
67    H.docTypeHtml $ do
68        H.head $ do
69          H.meta ! A.charset "UTF-8"
70          H.title . H.toHtml $ name
71        H.body $
72          H.span ! A.style "font-family: monospace;" $
73          mapM_ (\line -> preEscapedToMarkup (addSpaces line) >> H.br)
74                  (lines err)
75
76
77
78  -- | Using a page title and the full source of an Elm program, compile down to
79  --   a valid HTML document.
80  html :: String -> String -> H.Html
81  html name src =
82    H.docTypeHtml $ do
83        H.head $ do
84          H.meta ! A.charset "UTF-8"
85          H.title . H.toHtml $ name
86          H.style ! A.type_ "text/css" $ preEscapedToMarkup
87           ("a:link {text-decoration: none; color: rgb(15,102,230);}\n\
88            \a:visited {text-decoration: none}\n\
89            \a:active {text-decoration: none}\n\
90            \a:hover {text-decoration: underline;\
91            \ color: rgb(234,21,122);}" :: String)
92        H.body $ do
93          let js = H.script ! A.type_ "text/javascript"
94              elmname = "Elm." ++ fromMaybe "Main" (Elm.moduleName src)
95              runFullscreen =
96                    "var runningElmModule = Elm.fullscreen(" ++ elmname
97                                                             ++ ")"
98          js ! A.src (H.toValue ("/elm-runtime.js?0.11" :: String)) $ ""
99          case Elm.compile src of
100           Right jsSrc -> do
101               js $ preEscapedToMarkup jsSrc
102               js $ preEscapedToMarkup runFullscreen
103           Left err ->
104               H.span ! A.style "font-family: monospace;" $
```

```
105                 mapM_ (\line -> preEscapedToMarkup (addSpaces line) >> H.br)
106                       (lines err)
107
108 addSpaces :: String -> String
109 addSpaces str =
110   case str of
111     ' ' : ' ' : rest -> "  " ++ addSpaces rest
112     c : rest -> c : addSpaces rest
113     [] -> []
114
115 js :: String -> String
116 js src = case Elm.compile src of
117            Right js -> "{ \"success\" : " ++ show js ++ " }"
118            Left err -> "{ \"error\" : " ++ show err ++ " }"
```

9.4.6 EmbedMeElm.elm

```
1  module EmbedMe where
2
3  import Mouse
4  import Window
5  import Keyboard
6  import JavaScript as JS
7  import JavaScript.Experimental as JEXP
8  import Http
9  import Json
10
11 (~>) = flip lift
12 infixl 4 ~>
13
14 clicks : Signal (Time, (Int,Int))
15 clicks = timestamp (sampleOn Mouse.isDown Mouse.position)
16
17 user_id = "1"
18
19 firebaseRequest requestType requestData =
20   Http.request requestType
```

```
    ("https://sweltering-fire-9141.firebaseio.com/dissertation/elm/" ++ user_id
                                                                    ++ ".json")
    requestData []

serialize r = r |> JEXP.fromRecord
                |> Json.fromJSObject
                |> Json.toJSString " "
                |> JS.toString

toRequestData (t, (x,y)) = {t = t, x = x, y = y} |> serialize

toRequest event = case event of
  (t, (x,y)) -> firebaseRequest "post" (event |> toRequestData)

requests = clicks ~> toRequest

sendRequests = Http.send requests
```

9.4.7 EmbedMeJS.elm

```
module EmbedMe where

import Mouse
import Window
import Keyboard
import JavaScript as JS
import JavaScript.Experimental as JEXP
import Http
import Json

(~>) = flip lift
infixl 4 ~>

clicks : Signal (Time, (Int,Int))
clicks = timestamp (sampleOn Mouse.isDown Mouse.position)

user_id = "1"
```

```
18
19 firebaseRequest requestType requestData =
20   Http.request requestType
21     ("https://sweltering-fire-9141.firebaseio.com/dissertation/js/" ++ user_id
22                                                                     ++ ".json")
23     requestData []
24
25 serialize r = r |> JEXP.fromRecord
26                 |> Json.fromJSObject
27                 |> Json.toJSString " "
28                 |> JS.toString
29
30 toRequestData (t, (x,y)) = {t = t, x = x, y = y} |> serialize
31
32 toRequest event = case event of
33   (t, (x,y)) -> firebaseRequest "post" (event |> toRequestData)
34
35 requests = clicks ~> toRequest
36
37 sendRequests = Http.send requests
```

9.4.8 fullScreenEmbedMe.js

```
1 var firebaseData = new Firebase(
2                 'https://sweltering-fire-9141.firebaseio.com/dissertation');
3 var embedMe = Elm.fullscreen(Elm.EmbedMe, {});
```

9.4.9 editor.js.diff

```
1 diff --git a/resources/misc/editor.js b/resources/misc/editor.js
2 index d2bebc8..302663e 100644
3 --- a/resources/misc/editor.js
4 +++ b/resources/misc/editor.js
5 @@ -293,7 +293,7 @@ function showOptions(show) {
6  function showType(show) {
7      cookie('showtype', show);
8      document.getElementById('show_type_checkbox').checked = show;
```

```
-    var newMode = (show ? { mode: Mode.TYPES, verbose: false }
+    var newMode = (show ? { mode: Mode.TYPES, verbose: true}
                         : { mode: Mode.NONE });
     if (mode.mode === Mode.OPTIONS) {
         mode.hidden = newMode;
@@ -305,8 +305,8 @@ function showType(show) {

 function toggleVerbose() {
     if (!mode.verbose) showType(true);
-    document.getElementById('toggle_link').innerHTML = mode.verbose ?
-      'more' : 'less';
-    mode.verbose = !mode.verbose;
+    document.getElementById('toggle_link').innerHTML = mode.verbose ? '' : '';
+    mode.verbose = true;
     updateDocumentation();
 }

@@ -318,8 +318,8 @@ function showVerbose() {
 function hideStuff() {
     if (mode.hidden) mode = mode.hidden;
     document.getElementById('options_checkbox').checked = false;
-    mode.verbose = false;
-    document.getElementById('toggle_link').innerHTML = 'more';
+    mode.verbose = true;
+    document.getElementById('toggle_link').innerHTML = '';
     updateDocumentation();
 }
```

9.4.10 MovingBox.js

```
/*

Try moving the square around with your keyboard's arrow keys

Click your mouse over there =====>
Use arrows Up, Down, Left, Right

```

```
Whee!

Now modify the code to prevent the square from going outside
the edge of the grey window.

*/

var WIDTH = 400;
var HEIGHT = 400;
var SQUARE = 40;
var COLORS = [
    "0x000000",
    "0xCCCCCC",
];
var MOVEMENT_SPEED = 5;

var stage = new PIXI.Stage(COLORS[1]);
var renderer = PIXI.autoDetectRenderer(WIDTH, HEIGHT);
document.body.appendChild(renderer.view);

var box = new PIXI.Graphics();
box.lineStyle(1, COLORS[0], 1);
box.beginFill(COLORS[1], 0);
box.drawRect(0, 0, SQUARE, SQUARE);
box.endFill();
stage.addChild(box);

box.x = (WIDTH / 2) - (SQUARE / 2);
box.y = (HEIGHT / 2) - (SQUARE / 2);

var keyState = {};

window.addEventListener('keydown', function(e) {
    keyState[e.keyCode || e.which] = true;
}, true);

window.addEventListener('keyup', function(e) {
    keyState[e.keyCode || e.which] = false;
```

```
}, true);

requestAnimFrame(animate);

function animate() {
    if (keyState[37]) {
        box.x -= MOVEMENT_SPEED;
    }

    if (keyState[38]) {
        box.y -= MOVEMENT_SPEED;
    }

    if (keyState[39]) {
        box.x += MOVEMENT_SPEED;
    }

    if (keyState[40]) {
        box.y += MOVEMENT_SPEED;
    }

    renderer.render(stage);
    requestAnimFrame(animate);
}
```

9.4.11 MovingBox.elm

```
{-

Try moving the square around with your keyboard's arrow keys

Click your mouse over there =====>
Use arrows Up, Down, Left, Right

Whee!

Now modify the code to prevent the square from going outside
```

```
11  the edge of the grey window.
12
13  -}
14
15  import Keyboard
16
17  areaSize = 400
18  squareSize = 40
19
20  main : Signal Element
21  main = lift display position
22
23  delta : Signal Float
24  delta = fps 30
25
26  input : Signal (Float, (Float,Float))
27  input =
28      let vectors = lift toVector Keyboard.arrows
29      in  sampleOn delta (lift2 (,) delta vectors)
30
31  toVector : { x:Int, y:Int } -> (Float,Float)
32  toVector {x,y} =
33      if x /= 0 && y /= 0
34        then (x / sqrt 2, y / sqrt 2)
35        else (x,y)
36
37  position : Signal (Float,Float)
38  position = foldp update (0,0) input
39
40  update : (Float, (Float,Float)) -> (Float,Float) -> (Float,Float)
41  update (dt,(vx,vy)) (x,y) =
42      (x + dt * vx / 2, y + dt * vy / 2)
43
44  display : (Float,Float) -> Element
45  display xy =
46      collage (round areaSize) (round areaSize)
47        [ rect areaSize areaSize
48            |> filled grey
```

```
      , rect squareSize squareSize
          |> outlined (solid black)
          |> move xy
      ]
```

9.4.12 DecodeMouseData.py

```
#!/usr/bin/python2.7

import json
import sys

class DecodeMouseData(object):

    def decode(self, jsonString):
        return json.loads(jsonString)

    def getNumberOfClicks(self, jsonString):
        return len(self.decode(jsonString))

    def getSessionDuration(self, jsonString):
        decoded = self.decode(jsonString)

        finish = max([x["t"] for x in decoded.values()])
        start = min([x["t"] for x in decoded.values()])

        # compute number of minutes in this number of milliseconds
        return (finish - start) / 60000.0

    def getFilesTimeClickDict(self, files):
        timeAndClicks = dict()
        filename = str()
        for jsonFile in files:
            with open(jsonFile) as f:
                jsonString = f.read()
                filename = f.name
                print filename
```

```
31              timeAndClicks[filename] = (self.getSessionDuration(jsonString),
32                      self.getNumberOfClicks(jsonString))
33          return timeAndClicks
34
35      def getDictPrettyPrint(self, timeAndClicks):
36          output = ""
37          tablelines = "-"*11 + " " + "-"*10
38          output += tablelines + "\n"
39          output += "Time (min)  Clicks\n"
40          output += tablelines + "\n"
41          length = len(timeAndClicks)
42          for i,filename in enumerate(timeAndClicks):
43              output += ("%10f %10d" % (timeAndClicks[filename][0],
44                      timeAndClicks[filename][1])) + "\n"
45          output += tablelines
46          return output
47
48
49  if __name__ == "__main__":
50      dmd = DecodeMouseData()
51      print dmd.getDictPrettyPrint(dmd.getFilesTimeClickDict(sys.argv[1:]))
```

9.4.13 test_DecodeMouseData.py

```
1   import unittest
2   import DecodeMouseData as d
3
4
5   class FooTests(unittest.TestCase):
6
7       def setUp(self):
8           self.dmd = d.DecodeMouseData()
9
10      def testDecode(self):
11          expected = {'1':2, '3':4}
12          actual = self.dmd.decode('{"1":2, "3":4}')
13
```

```
14          self.assertEquals(actual, expected)
15
16      def testMouseDecode(self):
17          expected = {"-JKMBewWrFje3lHT8spD" :
18                  {"t" : 1397327310399, "y" : 646, "x" : 629}}
19          actual = self.dmd.decode(
20                  '{"-JKMBewWrFje3lHT8spD" : ' +
21                  '{"t" : 1397327310399, "y" : 646, "x" : 629}}')
22
23          self.assertEquals(actual, expected)
24
25      def testNumClicks(self):
26          expected = 1
27          actual = self.dmd.getNumberOfClicks(
28                  '{"-JKMBewWrFje3lHT8spD" : ' +
29                  '{"t" : 1397327310399, "y" : 646, "x" : 629}}')
30
31          self.assertEquals(actual, expected)
32
33      def testLotsClicks(self):
34          expected = 2
35
36          actual = self.dmd.getNumberOfClicks("""{
37    "-JKMBewWrFje3lHT8spD" : {
38      "t" : 1397327310399,
39      "y" : 646,
40      "x" : 629
41    },
42    "-JKMBewawNo6G_Zdfnkk" : {
43      "t" : 1397327310465,
44      "y" : 646,
45      "x" : 629
46    }
47  }""")
48
49          self.assertEquals(actual, expected)
50
51      def testComputeSessionTime(self):
```

```
52          expected = 0.0011
53
54          actual = self.dmd.getSessionDuration("""{
55    "-JKMBewWrFje3lHT8spD" : {
56      "t" : 1397327310399,
57      "y" : 646,
58      "x" : 629
59    },
60    "-JKMBewawNo6G_Zdfnkk" : {
61      "t" : 1397327310465,
62      "y" : 646,
63      "x" : 629
64    }
65  }""")
66
67          self.assertEquals(actual, expected)
68
69  def main():
70      unittest.main()
71
72  if __name__ == '__main__':
73      main()
74
```

Appendix D

9.5 Pilot Study 1 – Questionnaire

9.5.1 Consent Form

9.5.1.1 Study Overview This study aims to assess how Functional Reactive Programming Languages are used. To do this, we will be asking you to modify a Mario game to get him to fly. The session will take no more than 1 hour.

During the session, you will be introduced to Elm, a functional reactive programming language, as well as being shown what we want you to create.

We'll also present you with a questionnaire to see what experience you've had with Functional programming (or similar concepts) before. Finally we'll give you another questionnaire to ask how you think the session went, and the level of workload in the task.

The session will be recorded on video and then the audio from the session will be transcribed anonymously in order to find any problems that you had during the session. During this process, the data will be stored securely. Important Information

All data collected during this study will be recorded such that your individual results are anonymous and cannot be traced back to you. Your results will not be passed to any third party and are not being collected for commercial reasons. Participation in this study does not involve physical or mental risks outside of those encountered in everyday life. All procedures and information can be taken at face value and no deception is involved. You have the right to withdraw from the study at any time and to have any data about you destroyed. If you do decide to withdraw, please inform the experimenter.

By signing this form you acknowledge that you have read the information given above and understand the terms and conditions of this study.

Name	Age	Sex	Occupation
.....	...	...	.....

Signed

- ..

Date

- ..

Experimenter: Simon Buist, Dept. of Computer Science. EMAIL ADDRESS

9.5.2 Pre-questionnaire

9.5.2.1 Functional Programming languages

1. Have you ever used a Functional programming language before? Examples are: Scheme, Lisp, Haskell, ML, SPARK. Please circle one answer)
    - Yes
    - No
2. If so, please list the Functional programming languages you've used before:
    - .....

9.5.2.2 Design & Software For the purposes of this questionnaire, we consider a piece of software to be an application for which you have received/conceived of a specification, and coded a solution that meets this solution.

1. Have you designed software before?
    - Yes
    - No
2. On what platforms have you designed software?
    - Desktop
    - Mobile
    - Tablet
    - Web
3. For what purposes have you designed software?
    - Commercial
    - Academic (e.g. Coursework)
    - Personal project
    - Other:
4. Roughly how many pieces of software have you designed?

- ...

5. What programming languages do you know?

   - .....

9.5.2.3 General Demographics

1. How old are you?

   - ...

2. What is your sex? (Please circle one answer)

   - Male
   - Female

3. What is the highest degree or level of education you have completed? If currently enrolled please indicate the highest you have attained previously. (Please circle one answer)

   - None
   - GCSEs or equivalent
   - A/AS levels or equivalent
   - BSc/BA or equivalent
   - MSc/MA or equivalent
   - PhD or equivalent

4. In what field was your highest qualification?

   - .....

5. What is your current employment status? (Please circle one answer)

   - Unemployed
   - Self-employed
   - Employed
   - Student
   - Retired
   - Unable to work

9.5.3 Post-Questionnaire

1. Please detail any comments on the result that you achieved

. . . . . .
. . . . . .
. . . . . .

2. Please detail any comments on how you achieved it

. . . . . .
. . . . . .
. . . . . .

3. Please detail any other comments

. . . . . .
. . . . . .
. . . . . .

If you want to have the study as a whole explained to you, please do so now. However we ask that you refrain from discussing this with potential future participants.