Background Chapter

Lewis Raeburn

October 2021

June

I used the programming language learning platform "Codecademy" (Codecademy, 2011) to learn the basics of HTML, CSS, and JavaScript. With this knowledge alone I would have been capable of developing my own interactive webpage. Why did I learn this? JavaScript is the main tool used to interact with the browser and I will require it in order to implement webcam support for Links.

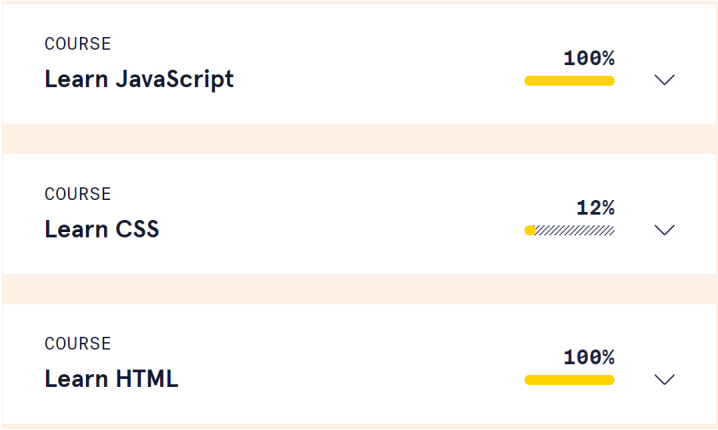

Figure 1: Learning progress in Codecademy. Regarding the low progress on "Learn CSS", I felt that I had learnt enough CSS from the 12% in order to style webpages adequately, and that I could use Google to look-up anything new I came across.

July

I used the tutorials on the website "reactjs" (Facebook, 2013) (the learn by doing tutorial and step-by-step guide, respectively) to develop a basic understanding of the JavaScript library, "React.js". Why did I learn this? I heard of this popular JavaScript library and how useful it was for developing user interfaces for webpages. Also, the React webpage was referenced by my supervisor, Sam Lindley, on the Degree Project Management Tool (DPMT) webpage for my project and so I felt it would be necessary to learn.

August

I used React.js and CSS to develop my own webpage. The webpage includes a stickman figure which can be moved around the webpage by holding down the arrow keys, respectively. I created a stickman walking animation (using CSS) which displays when the stickman moves. There are walls placed around the page with which the stickman collides. Why did I try this? I believed that, by using the skills I learnt (React, JavaScript, HTML, CSS) on my own ideas, I would become better and more familiar with them. Front-end web development seems to be a large part of my project, and so it would be wise to become proficient in this area.

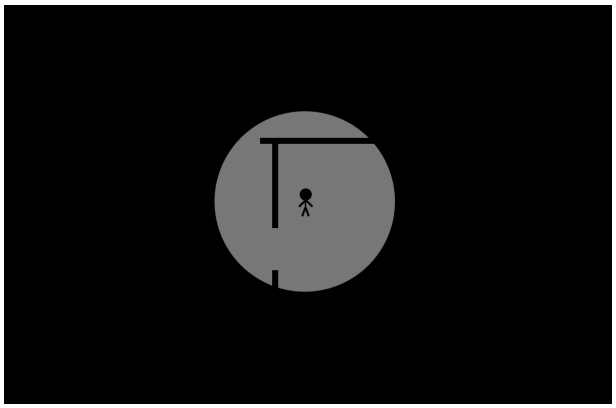

Figure 2: Web application implemented with React which represents the user as a stickman and allows the user to move around the page using the W, A, S, D keys on their keyboard. The user collides with objects, such as the black bars which represent walls, and their view of the page is limited to within a certain radius of their position.

I taught myself WebRTC and how it works regarding peer to peer connections, through various sources. These sources include the WebRTC website itself (Google, 2011), a very informal YouTube video (Nasser, 2020), and an ex-

ample my supervisor referred me to (Hladis, 2019). Then, I created a markdown document which contains a thorough explanation of how WebRTC works and an explanation of an example application using WebRTC, all in my own words. Why? It was vital I developed a deep understanding of WebRTC as I needed to implement support for it in a programming language other than JavaScript (Links).

I used various tutorials on the OCaml website (https://ocaml.org/, 1996) to teach myself OCaml. Specifically, I completed the "Up and Running", "A First Hour with OCaml", and "Data Types and Matching" tutorials. Using what I learnt, I completed the first 20 problems of the "99 Problems (solved) in OCaml" section. Why did I do this? OCaml is a functional programming language, similar to Links, so it will help me get used to programming using functional programming concepts (e.g. no side effects, recursion, higher-order functions, function composition).

September

I began to try to learn the functional programming language, Links, through the Links website (https://links-lang.org/, 2005-2009) and the Links GitHub repository (https://github.com/links-lang/links, 2014). Initially, I gave the Links documentation (found on the website) a read through to get a shallow grasp of the syntax, types, and web development features of the language. My main source of learning Links has been through example Links programs found on the GitHub. Why did I learn this? Links is the language I will be using to develop the goal web page so it is essential I get familiar with it.

My supervisor referenced me to a web page which consisted of a thread (eambutu, 2020) where many people mentioned various web applications which are similar to gather.town in the sense that users can video call and walk around in a virtual space (e.g. a room inside a building). Seeing this, I thought it would be a good idea to record these systems on a document (Raeburn, 2021) so that I could compare them and identify patterns, as well as point out potential features which would make the systems better and increase their usability.

October

I learnt how to have the client broadcast data to all connected clients using the distributed actor-style concurrency (https://en.wikipedia.org/wiki/Actor_model, 2021) in Links which was demonstrated to me by one of the developers of Links, Simon Fowler. In simple terms, this style of messaging works as follows: the server creates a process which waits for messages to arrive, a client connects to the server and creates their own process and sends their process ID to the server process along with a message (e.g. data of any type), the server receives this message and process ID and adds this process ID to it's collection of client's

process IDs, then the server sends the message to all of the process' in it's collection. This notion of broadcasting messages is used in the implementation of multi-user video chat with WebRTC (Hladis, 2019) in order to allow the user to broadcast data to all connected users. For this reason, it was important that I knew how to implement this in Links.

The next step was not to learn anything else, but try to implement correctly a WebRTC connection between users in Links, and subsequently a video call. This was arguably the most important part of my project as it forms the foundation of the goal web application. Throughout the two or three weeks I worked on this step, I came across countless errors which I struggled to fix, and also ended up with versions of the program which did not function correctly. Most of these bugs required a simple change to a few lines of code to fix, but it was the time it took to find these simple changes which prolonged the implementation. For example, the one bug which took me nearly two weeks to fix was due to a server's process ID (which was sent to the server by the client beforehand) not being taken as a process by the client after being sent back. I was surprised to see that having the server send "self()" to the clients instead worked. Once I fixed this bug, it took only a week to complete two different implementations of the program which worked just as well as the example program which was programmed purely in JavaScript.

References

Codecademy (Aug. 2011). *Programming language learning platform.* URL: `https://www.codecademy.com/`. (accessed: 10.06.2021).

eambutu (Nov. 2020). *gather.town thread.* URL: `https://news.ycombinator.com/item?id=25039370`. (accessed: 26.09.2021).

Facebook (May 2013). *JavaScript library for building user interfaces.* URL: `https://reactjs.org/`. (accessed: 05.07.2021).

Google (2011). *Website for WebRTC.* URL: `https://webrtc.org/`. (accessed: 20.08.2021).

Hladis, Jirka (Oct. 2019). *Multi-user video chat with WebRTC.* URL: `https://www.dmcinfo.com/latest-thinking/blog/id/9852/multi-user-video-chat-with-webrtc`. (accessed: 01.08.2021).

https://en.wikipedia.org/wiki/Actor_model (Oct. 2021). *Distributed actor-style concurrency for messaging.* URL: `https://en.wikipedia.org/wiki/Actor_model`. (accessed: 01.10.2021).

https://github.com/links-lang/links (Jan. 2014). *Links GitHub Repository.* URL: `https://github.com/links-lang/links`. (accessed: 05.09.2021).

https://links-lang.org/ (2005-2009). *Links Website.* URL: `https://links-lang.org/`. (accessed: 05.09.2021).

https://ocaml.org/ (1996). *OCaml Website*. URL: https://ocaml.org/. (accessed: 27.08.2021).

Nasser, Hussein (Nov. 2020). *WebRTC Crash Course*. URL: https://www.youtube.com/watch?v=FExZvpVvYxA. (accessed: 24.08.2021).

Raeburn, Everett (Sept. 2021). *Dynamic Video Chat Web Apps*. URL: https://docs.google.com/document/d/1CF4jH96AcmPn03hOdhhIdghhJ_vpk3vGzp2kuc1VXn4. (accessed: 27.09.2021).