Project Reflection

Jingyi Zhang

In this paper, there will be a brief introduction for the group project I did with my co-developer Merritt Ruthrauff in the past 2 weeks with some reflections I had for Elm language compared to the other project I did using Django and Phaser.

First of all, the project we did was an updated version of pong. The content of this game contains 2 teams with 4 players and 3 power ups for each team which will activate based on the points they score. Although the time period for me to work on this project was relatively short, during the making of this game, I still had better understandings for this language especially when I ran into problems. While trying to solve those issues, it really made me think of how to use the Elm architectures cleverly.

Last year, I did a project which was pretty similar to this one. For that project, I used Django for web frameworks and Phaser for the game frameworks. I would say building up that project was a rather interesting experience in the beginning, but painful when I was trying to finish it up. In the beginning, web developments and game developments were separated so things flowed fluently. However, Django takes Python and Phaser takes JS, so one of the problems I had was to waste a huge chunk of time on researching the methods and restructure the codes I wrote just to run the JS applications using Django. Other than this issue from Django’s end, another problem that I ran into was that with Phaser’s JS library, the base game was built easily. But it was difficult to make the game more entertaining. I felt my hands were tied while trying to build up the game since I could only use what Phaser’s library provided and most of my ideas couldn’t be achieved due to the lack of flexibility in Phaser. To sum this up, although both of Django and Phaser had great modules and functionalities for a web app such as Django’s modules for web securities with MySQL database for users and Phaser’s easily used physics engine, they just took too much time for me to learn and understand for some of the simple things I was trying to do and they were not flexible enough for some of the creative ideas I wanted to make. So compare to that project I did last year, I believe Elm gave me a better experience this time for developing a web game.

For me personally, Elm’s program architecture was pretty easy to understand. I would say it shares similarities with Django programs’ architecture. It has three separate parts: Model, Update and View. The state of application and types can be made in Model then updated based on different conditions in Update, finally, the codes in View can build up the user interface. The thing I like about Elm is that although it’s programs also has functionality to talk to JS, it is not required for web game developing because Elm itself provides modules which can be used for web games, thus it saved me a lot of time which I used on game’s content developments instead of doing enormous researching and debugging just to get the basic prototype of the game running. So one thing I really liked about Elm was its simplicity.

Since getting the base game working and running was by contrast an easy task thanks to Elm’s simplicity, I started to think about what I could add into this game to make it more interesting and intense for the players to play. After few discussions with my co-developer Merritt, few amusing ideas were bought up. We thought that regular pong games were just not interesting enough compared to other mini games nowadays. So we decided to have 2 additional players and power-ups for our pong game. When it comes to power ups, I thought about various ways to build those power ups into the codes. Unlike to Phaser, I think Elm is more flexible and it allows me to make my ideas possible by modifying the three parts in Elm program I introduced previously (Model, Update, View) and playing around with the arguments which some specific functions would take. After modifying the program based on what the game content requires, the rest of the work was just debugging to make sure each team gets correct power ups under certain conditions and every power up could behave correctly. I would say while developing those game contents, it was not a frustrating experience due to Elm’s simplicity and flexibility. Although I had troubles that I ran into but I eventually solved them by changing my perspectives on coding. A great example would be for some power ups, modifying their behaviors in Update was just a disaster and some of them just wasn’t functional. But playing around with different conditions in View and new functions with different arguments in Update helped me solving some of those issues quicker and easier than I imagined. So the second thing I liked about Elm is the combination of its simplicity and flexibility.

Overall, it was a great experience for me to develop a web game using Elm. I would say if I had more time with this project, I will be able to explore more functionalities and usages of this language and the library it currently has. Based on the experience I had with Elm and past experience with other methods on building websites and web apps, I think using Elm is an easier and faster way for those who wants build a simple website. However, I can’t give an accurate response for those who wants to build a fully developed website using Elm because I still haven’t experienced how Elm programs deal with interactive actions among different clients or how they deal with databases including how to secure user’s information. But I believe with this language’s growing community and some of its unique traits, modules which are able to help developing those functionalities will eventually be added into the library if they currently aren’t. So I’m optimistic about the future of Elm and I have a high expectation on using this language again for my upcoming projects.