Person-Hour Estimate Project 3

As a part of design we laid out a few components: a map, a pathfinder, a search, and runtime. The map is made up of several classes: nodes, floors, buildings, and a final map class to contain them all. These are primarily containers for data and shouldn't take much work, so we'll estimate 15 hours to put them together. The pathfinder is a large chunk of the meat of the program, so we'll estimate 15 hours for that component. The search function represents another large chunk of functionality so we'll estimate 10 hours to get it running. Finally, we'll estimate that all of the runtime code and documentation will take another 10 hours. This leaves us with a total of 50 hours for our time estimate. These estimates are based on how long we took to make similar-sized components in the previous projects.

Person-Hour Estimate Project 4

To complete project 4, all that needed to be done was expand the base established in project 3 to include all of Eaton. This would involve creating the background maps for the other floors as well as the node maps for those floors. Based on the time those tasks took for the ground floor, we estimate that completing the remaining floors will take 10 hours. Aside from that, the interface must be updated to display multiple floors as well as a path that traverses floors. The pathfinding algorithm is already well equipped to handle a path between floors. Modifying the rest of the code base to account for more floors shouldn't take too long, seeing as the systems we already have in place from project 3 are very modular. Because of this, we estimate that the other components of the program will take 15 hours to modify.

Design Paradigm

For this project we chose the object-oriented design paradigm. The script for our project is broken down into classes, each of which handle a specific portion of the functionality. The broad sections of functionality for the program are as follows: storage of map data, pathing through the map, and the search function.

We create the map using a set of Nodes which represent vertices on a path. These node objects are stored inside a container which we call a Floor. Floors represent the levels of a building, and a set of Floors are contained inside of a Building class. Finally, there is a Map class made up of one or more Buildings. This method of data storage was a natural consequence of object-oriented design, as we examined the real-world object we were trying to model, a set of buildings on campus, and broke it down into logical components.

The pathing is done by one primary class and two helper classes. The Pathfinder class uses Dijkstra's shortest path algorithm to generate the shortest path from a single starting location to any other location on the map. To do this it makes use of a min priority queue class along with a queue element class. The queue element class wraps the thing being stored in the queue with a numerical priority.

The Search class is more functional than object-oriented. It is responsible for validating the input of the search bar and for finding the node associated with a searched location. The search class works in tandem with runtime code to initiate the pathing based on user input.

We chose the object-oriented paradigm for a few reasons. One of the primary reasons was that all of us primarily have experience working with object-oriented languages like Java and C++, so it was only natural for us to gravitate towards an object-oriented implementation for this project. Also, it was very intuitive to design the map data storage in an object-oriented manner, as it is easy to visualize how a map fits together using Node objects; it's essentially a game of connect-the-dots.

Software Architecture

For our project we decided to go with a 2-tier architecture to go along with our object-oriented design paradigm. The two tiers are the client and the second is the data layer stored within the program. The client can interact with the website in a variety of ways which affect what occurs.

The program is displayed through a website which is put on the internet. The users can open this in whatever web browser they have access to, including using their phone as it is a responsive layout. This means that the user client can interact with the program in multiple ways from the web.

The data-layer contained within the program is what changes the website in response to what the client does. The intent of the program is to allow users to enter in the location of a classroom inside the engineering complex and to get detailed instructions on how to get there. Right now, for the prototype, we are only modeling one floor of Eaton Hall to demonstrate the technology and vision. For the final version we envision a complete, multi-floor Eaton Hall implemented. However, this could get applied to the entire engineering complex or campus in general.

The interactions that the client can currently do are to set the starting location, enter in a classroom, and to visit one of our githubs. The starting locations are preset from a drop-down menu in this prototype. It also includes things such as the elevator and staircase, which in the final version will probably not be an option as those two will be done from behind the scenes if trying to traversal multi-level.

For the classroom interaction, the user has to provide a valid classroom or else the program will return to the client that it was an invalid string. This is one example of the interaction between the client and the program in our 2-tier architecture. The 2-tier architecture allows us to have a flexible program which can deal with multiple client instances while having consistent intended results.

Design Patterns

For this project we used many design patterns in the creation of this project. They are Factory method, Singleton, Composite, Decorator, and Iterator.

We used the Creational Design Pattern Factory method by making a Building object. The Building object has multiple floors in it and we only instantiate a floor when we are going to use it for pathfinding.

We used the Creational Design Pattern Singleton by making a floor search object. We use that object to see if the user input is valid or not before it goes to the pathfinding object.

We used the Structural Design Pattern Composite when we made the floor object. Each floor has an array of nodes that contains an x and y location and if it is an end node a name of that location. The nodes are linked together by vertices so each node knows what other nodes are connected to it.

We used the Structural Design Pattern Decorator in the node object. Each node has an x and y location in it but there is a subclass of nodes called end nodes and they have an x and y location as well as a name and room number in it.

We used the Behavioral Design Pattern Iterator in multiple parts of our project. The first place we use it is in the pathfinding algorithm to see what nodes are connected to other nodes. The second and third time we use it is in the search class. The first iterator in the search class iterates over the node array to see if the user input is in the node array or not. The second iterator in the search class also iterates over the node array but it looks for the node number that matches the user input so that we can run the pathfinding algorithm with that node as the end point.

Integration Strategy

For project 3 & 4 we did the Bottom-Up integration strategy. The reason we did it this way was that each team member could make a part of the project and then at the end we could put it all together into one working program. This allowed each team member to contribute to the project and made sure that no one was taking advantage of the other team members by not working on the project.

The way we integrated each other's code was that we had someone do the most important base part of the project first and then we would add code around that until we got the whole project to work. For project 3 & 4 the base part of the project was the website and the node map that represents points on the floor of Eaton hall. After that we had two people working on a pathfinding program and a way to get user input from the website. After they were done with those two things we had someone make a path drawing program so that when you gave the program a location you wanted to go to it drew a line from the start location to that end location. While adding all the parts to the project we tested to make sure that our parts work with the previously added components. This integration strategy worked well for our team and it allowed each team member to do good work on their respective components. and allowed us to make a wonderful project.

Deployment plan

The first step in our deployment plan is to add the rest of the buildings in the Engineering complex to the website. Our initial idea for project 3 and 4 was to create an interactive map that

covered the whole of KU's engineering complex, however we concluded that it was more feasible to do just one building for projects 3 and 4. After implementing the rest of the engineering buildings, we would pitch the idea of this website to the School of Engineering and to the University. If it gets approved the next step is to get a domain name for the website and a server to host the website. We would most likely piggyback off of what KU's uses for hosting a website. Basically, we are trying to do the same thing that the students who made the EECS Undergraduate Graduation Planner website did. After that, we would develop an app version of the website so that it would be easily available on mobile devices.

When we first came up with this idea the potential target market was people new to the KU School of Engineering or new to campus in general. Specifically, we thought that the tool would be very useful to new engineering students who have never been inside the KU engineering complex. However, as we got deeper into the development of the project we realized that more groups of people could use this to help navigate the School of Engineering. The first group was new professors and staff who don't know their way around the buildings yet. The second group of people were students visiting the School of Engineering on a college visit or other tour. The final group of people are the people who come to the Engineering complex for events like HackKU or the Engineering Expo. This will help these groups find classrooms and meeting areas faster and help them avoid getting lost in the buildings.

The cost of deployment will vary based on whether we are able to use the same web hosting server as the Engineering school or not. If we are able to use the same servers employed by KU then we don't have to spend any money on our own server. However, if we can't do that then web hosting will cost around $8 a month. The domain name would cost $21 a year for a .org name. If we gave or sold the project to the University there is probably some way to get the .edu domain through the US government; that would cost around $40 a year. If we make the app version of the website it would cost $99 a year to have it on the Apple App Store and a one time payment of $25 to have it on the Google Play store. This adds up to an annual running cost of around $29 if we hosted it ourselves or around $40 and up if we host it on the University's servers, depending on whether or not KU needs us to pay for the use of their servers. Add $99 annually to these totals if we decide to maintain a mobile version of the project.

Maintenance plan

Maintenance of this project will vary greatly depending on whether or not the School of Engineering takes over this project. If the Engineering school does take over the project then maintenance of the project becomes much easier. The reason for this is that I assume that the Engineering school has its own dedicated servers for their websites along with some developers whose job is to maintain those websites. If this is true then the only maintenance that would be required for our project is maintaining the website's domain name. That would cost us $21 a year for a .org name or $40 a year for a .edu name. So if this is the case then there would be very little maintenance that we would have to do.

However, if we needed to host the servers ourselves then it would cost us a lot more. The first thing we would have to do is keep the website running. To do that we would have to have a web server hosting the website and a domain name. The servers would cost around $8 a month and the domain name would cost around $21 a year for a .org name. That comes out

to around $29 a year just to keep the website running, costs of proper maintenance aside. To do actual maintenance we would need to hire a web developer that knows both HTML and Javascript. At the current scope of the project I believe that one web developer would be able to maintain the website. There is not a lot of adaptive, corrective, or preventive maintenance to do for the website. The only major maintenance would be perfective maintenance. An average web developer's salary is around $69,430 a year or around $34 an hour. So if we add a web developer to our maintenance then it would be around $69,459 a year, which is a lot of money. Luckily we don't have to worry about monthly or annual fees for a distribution platform, but if we do add an app version like we mentioned in our deployment plans then it will cost more money for maintenance. It would cost around $99 a year to keep our app on the Apple app store and if we want to add an app developer also it would cost around $70,000 a year. So if you add the maintenance of a mobile app to our original maintenance plan the yearly cost would be around $139,558 a year to maintain a website and mobile app with developers. However, if we didn't have any developers the yearly cost would be around $128 a year to maintain the website and mobile app. Going the way of having no developers has its drawbacks however, because then there is a chance that the website or mobile app might break when the software updates or something else could happen that breaks it and nobody is readily available to fix it.

One other possible area of maintenance for the program is to make sure that it is current. KU's main campus periodically gains a new building, or old buildings undergo renovations. In order to keep the program functioning properly, maps must be updated and added to as the campus changes.