Ahmed Khan
Adam Freed
Gavin Silva

Professor Chenhansa
CS-124
20th April 2023

Preliminary Group Lab Report

Purpose:

The purpose of this lab is to be a capstone in our Ohlone College Computer Science journey, and a true stepping stone from introductory concepts to the upper division and professional world. This world is built upon a few core tenets, including best practices and industry standard tools, and we are attempting to incorporate these into our project as well. These include the use of git version control, modular integration of external APIs, concise and granular class and function implementation of powerful and versatile data structures to manipulate our application data, all centering around a clearly defined goal.

Our program, called Eagle Eye, aids rangers in their never ending stewardship of the natural world. Our forests and mountains, the last bastions of Earth untainted by the spread of civilization, are under threat from manmade and natural forces. Though we have the tools to transport ourselves across the earth in little time, something just a stones throw away can go unnoticed and be unreachable until it is too late. If our ample resources were only directed to the right place and the right time, we could ensure our nature would both survive and thrive into the future.

Wildfires are Eagle Eye's most pressing concern. They are a destructive phenomena but paradoxically integral to the natural order. Therefore must care and attention must be placed for what they are, where they are occurring, and how they are being dealt with. According to the National Park Service, "nearly 85 percent of wild-land fires in the United States are caused by humans...from campfires left unattended, the burning of debris, equipment use and malfunctions, discarded cigarettes, and intentional acts of arson" (1).

Much work must be done, more committed individuals must be recruited, and more powerful tools must be made to meet UN SDG 15's "aims to protect, restore, and promote sustainable use of terrestrial ecosystems, sustainably manage forests, combat desertification, and halt and reverse land degradation and halt biodiversity loss" (2). Eagle Eye will make a contribution towards this goal by providing the innovative means of having a greater perspective, and a faster more efficient allocation of present and future resources towards the cause.

We hope that a vast variety of individuals will benefit from our program, from park rangers in American national parks, or the inhabitants of the last great wildernesses such as the Amazon or the African Savanna. Perhaps even the average individual can come to benefit, both by contributing their own individual sightings, or using such information to inform their activities. Lastly we hope it can integrate with emerging drone and satellite technology to greatly expand its reach.

References:

1. https://www.nps.gov/articles/wildfire-causes-and-evaluation.htm
2. https://sdgs.un.org/topics/forests

Plan:

Eagle Eye will take a high altitude overview of terrain and highlight points of interest with waypoints. The waypoint data structure will be a core aspect of the program. This will be an object that represents a section of whatever park our program is being used at. It will store its location, name, and will contain an unordered_map<string, int> or other data, of all the different things being measured at that location, and what the measurements are. Most of the program will revolve around the manipulation of these waypoints, adding various types carrying various functionalities. An example is highlighting a potential fire and a priority level to addressing it, or instead a chain of waypoints to produce a path where a graph search algorithm will find the shortest traversal for purposes of extinguishing, creating fire lines, rescue operations, etc.

The waypoints won't necessarily have a value in every metric. For example, one waypoint might have a high fire risk, but won't have a value for flood risk, while another waypoint in the same location will have a flood risk but not fire risk. There will be a set that contains every single metric in use by any waypoint.

We will have multiple data structures that each store pointers to all of the Waypoints. This allows for different user needs to each have high time-efficiency. First, we will have a hashmap that stores all existing waypoints. The hashmap will use separate chaining. The hash function will operate on the name of the Waypoint. This way, if the user needs to search for a specific waypoint, they will be able to find it in O(1) time.

Next, we will have a vector of all the existing waypoints. This allows for the user to execute a general search for all waypoints, ordered by priority in any given measurement. We will be using heapsort for this. For example, if a user wants to search for all of the waypoints in order of highest fire risk, it would reorder this vector and then display in order. We considered using priority queues but realized that the user might care about which waypoint was second or third highest risk as well.

We will also store a set of all of the measurements the user is storing, so that the user can add or remove measurements as needed. For example, some parks might care more about flood risk, while other parks will care about the risk of drought. Different parks will have different priorities in what they measure, so they will be able to choose for themselves what they keep track of.

The last core part of our code will be the "undo button". This will be implemented by keeping a stack of deleted waypoints. Any time the user chooses to delete a waypoint, it will be removed from the hashtable and vector, and added to the stack. If the user hits the undo button, it will be popped from the stack and added back to the hashmap and vector. The stack will be implemented using a linked list.

Giving the user the option to manipulate waypoints and navigate the map at will will require a fundamental change in program structure from previous endeavors. Event driven programming, where the program waits to see what the user will do rather than prompting them to do a specific thing, will be a core focus. There will be function calls based on conditions of various keystrokes and options such as adding and removing a waypoint, undoing previous changes, etc. The program will listen to the input and then go down that particular path branching to the desired conclusion before returning to listen for another event. There is debate between creating an ASCII or wxWidget GUI, since the project requirements demand complete functionality of the core program before non-essential endeavors are undertaken, however the visual interface is a core aspect to the program so perhaps the functionality will be kept open ended to easily integrate an external API.

Files will be organized based on classes, with distinct and concise functions that do only one thing, but organized into highly versatile classes. The inheritance structure will boil down to a "runtime" class which will actually handle data and events. Main will only exist to call the runtime class. The runtime class itself will store the state booleans and "global" data members that are merely members of the overarching class.

Timeline:

4/23/23:

- Create Waypoint base class and decide on what data they will contain
- Research on Forest conservation topics and important data points
- Find professionals to contact for more information about the subject matter

4/24/23:

- Find a day where we could interview a ranger/conservationist/etc
- Implement quicksort algorithm to arrange waypoints by various resource values
- Create a vector of waypoints to be the baseline storage method for printing to the map

4/25/23-4/26/23:

- Use hashtable to store information about each waypoint such that data can be efficiently retrieved
- Make user interface to search for waypoint name in the hashtable
- Allow user to insert and delete waypoints
- Create a stacks to collect the deleted waypoints for an undo button

4/27/23-4/28/23:

- Use sets to display several options or even sets of options
- Make an event handler that uses a tree or BST for complex decision pathways

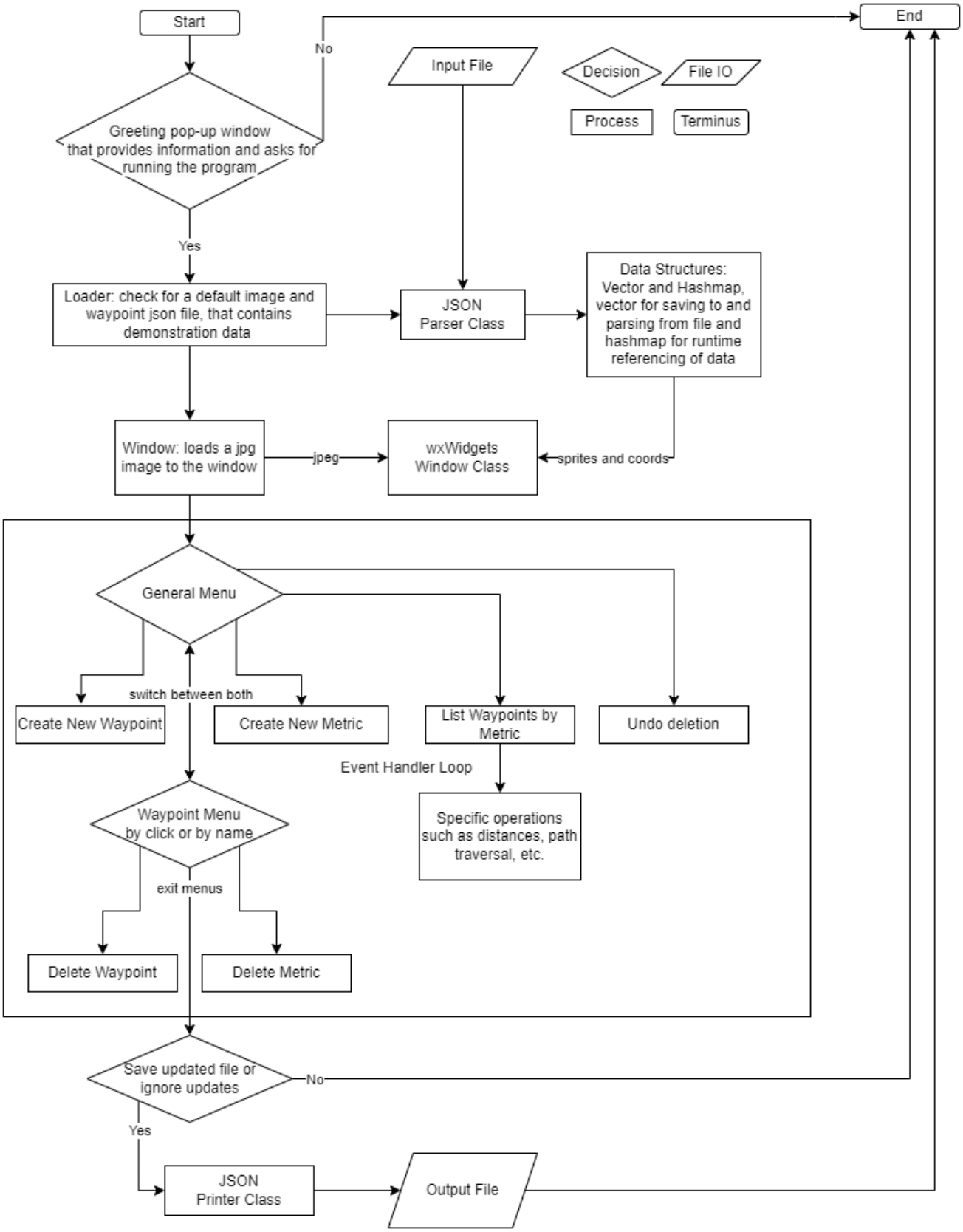
4/29/23-4/31/23:

- Create a working GUI for our waypoints to be mapped on either in wxWidgets or in Bash ASCII
- Proofread code for potential optimizations/corrections

5/1/23:

- Time allocated to finish the necessary data structures and features for the project

Flowchart


Start
End
No
Input File
Decision
File IO
Process
Terminus
Greeting pop-up window that provides information and asks for running the program
Yes
Loader: check for a default image and waypoint json file, that contains demonstration data
JSON Parser Class
Data Structures: Vector and Hashmap, vector for saving to and parsing from file and hashmap for runtime referencing of data
Window: loads a jpg image to the window
jpeg
wxWidgets Window Class
sprites and coords
General Menu
switch between both
Create New Waypoint
Create New Metric
List Waypoints by Metric
Undo deletion
Event Handler Loop
Specific operations such as distances, path traversal, etc.
Waypoint Menu by click or by name
exit menus
Delete Waypoint
Delete Metric
Save updated file or ignore updates
No
Yes
JSON Printer Class
Output File