Status Report 1

Group 5: Taylor Hopkin, Davin Lim, Oliver Tipton, Kiko Lancastre

March 24th 2024

1. Introduction

1.1 Highlights

• What was the plan for this iteration?

- Wireframe for UI.
- Collect Data.
- Implementation of Dijkstra's Algorithm for Small Set of Data.
- Meet with various Customers/StakeHolders to fine tune expectations and goals of the project.

• Highlight what the team accomplished.

- We were able to accomplish all of our plans. Each goal was achieved in a way that we hoped it would be. The only issues we ran into was how difficult data collection was. We were able to complete it, but it took far longer than expected.

1.2 Changes

Summarize any major changes since the proposal.

- We want to be more organized without eventual layout of our system. The goal is to make the UI far clearer in terms of what each location is. For example, we want to separate all of the academic buildings, dining options, etc.
- Similarly, we want to add a couple more services that would be of particular use to the admissions office (one of our primary target customers). We hope to highlight locations that are of particular interest to see. For example, the sculptures, the front of chambers, etc. Similarly, we want to specify accessible paths.
- No need to include the gaming aspect where people can compete with one another to achieve faster times.

Include each change’s date, motivation, description, and implications.

- March 18th: Met with Maggie Woodward who helped us determine the change to make the system organized by building type. The implications will be as we start developing the UI, we will have to cater to her requests/suggestions.
- March 19th: Met with Cate Goodin who helped us determine that it would be useful to have a scenic and accessible option. This will require a slight modification to the algorithm but should not be too difficult.
- March 21st: Met with Matthew Bernard who said that the gaming aspect was not particularly necessary. No implications besides there not being a need to code an additional gaming system.

2. Customer-Iteration Goals

Describe the product backlog of this iteration.

- Wireframe for UI.
- Collect Data.
- Implementation of Dijkstra's Algorithm for Small Set of Data.
- Meet with various Customers/StakeHolders to fine tune expectations and goals of the project.

Describe the customer’s desired overall experience.

- After meeting with customers during this iteration, we feel we have a far more complete understanding of how we can cater our product to meet our customers desired experience. It seems that we are going to have two primary customers: students/faculty and the admissions office.
- Students and Faculty: The desired overall experience for these customers is a simple, functional app that quickly allows them to find the fastest route to their desired location(s) and provides them an accurate estimate for how long the specified trip is going to take.
- Admissions Office: The desired overall experience for these customers is a simple, functional system that allows them to hit a number of locations in an efficient manner to give prospective students and families a comprehensive tour of the campus. These customers want to have a clear presentation that separates different buildings by categories and specifies which paths may be most scenic as well as paths that are going to be accessible.

Describe the sprint backlog of this iteration.

- In this sprint, we split the work up and gave everyone individual tasks to complete. Everyone took responsibility in data collection, by all of us (and a number of volunteers) recording their walking times, we are able to get a good estimate for the population average for each walking edge. Similarly Taylor took the lead in implementing Dijkstra's Algorithm for a Small Set of Data. It will allow us to integrate it into the GoogleMaps API in our next sprint. Finally, Davin and Oliver took the lead in starting to create the wireframe for our UI

Explain why you have selected the work items in the sprint backlog for this sprint (or iteration).

- These were all items crucial to future sprints. Without having completed these items, making progress would be very difficult. Now that they are done, we can start to move along.

2.1 Use Cases

Write a use case for each main user goal for a primary or secondary customer.

Show each use case’s title, user goal, and full basic flow. Choose meaningful titles.

Each use case should have at least one specific alternative flow and one bounded alternative flow.

Name: Find the shortest path from point A to point B on campus	
Goal: A user is able to search the fastest way to get somewhere on campus	
Actors: primary user, Wildcat Ways system	
Basic Flow: 1. The user accesses the system. **{ Load System }** 2. The user inputs the starting point **{ Process User Input for Staring Point }** 3. The user inputs the destination **{ Process User Input for Destination }** 4. The system runs the shortest-pathfinding algorithm **{ Run Dijkstra's Algorithm}** 5. The system outputs the shortest path from the starting point and the destination on a map **{ Display Path on Map}** **{ Terminate System }**	Alternative Flow(s): Variations on the basic flow Specific Alternative Flow: At **{ Process User Input for Destination }**, if the selected destination is not a valid destination, the system displays “Your selected destination is not valid. Please check your destination.” Resume the basic flow at **{ Process User Input for Staring Point }** Bounded Alternative Flow: At any point between **{ Process User Input for Staring Point }** and **{ Display Results }**, if the network connection is lost, Display “Connection lost. Please check your network connection.” Resume the basic flow at **{ Load System }**

Name: Find the shortest path from point A to point B to point C on campus
Goal: A user is able to search the fastest way to get somewhere on campus while stopping by somewhere else in the middle
Actors: primary user, Wildcat Ways system

Basic Flow: 1. The user accesses the system. **{ Load System }** 2. The user inputs the starting point **{ Process User Input for Staring Point }** 3. The user inputs the stop point **{ Process User Input for Stop Point }** 4. The user inputs the destination **{ Process User Input for Destination }** 5. The system runs the shortest-pathfinding algorithm **{ Run Dijkstra's Algorithm}** 6. The system outputs the shortest path from the starting point to the stop point to the final destination on a map **{ Display Path on Map}** **{ Terminate System }**	Alternative Flow(s): Variations on the basic flow Specific Alternative Flow: At **{ Process User Input for Stop Point }**, if the selected stop point is not a valid stop point, the system displays “Your selected stop point is not valid. Please check your stop point.” Resume the basic flow at **{ Process User Input for Staring Point }** Bounded Alternative Flow: At any point between **{ Process User Input for Staring Point }** and **{ Display Path on Map }**, if the network connection is lost, Display “Connection lost. Please check your network connection.” Resume the basic flow at **{ Load System }**

Name: Find an accessible path from point A to point B on campus
Goal: A user is able to find an accessible path to get somewhere on campus
Actors: primary user, Wildcat Ways system

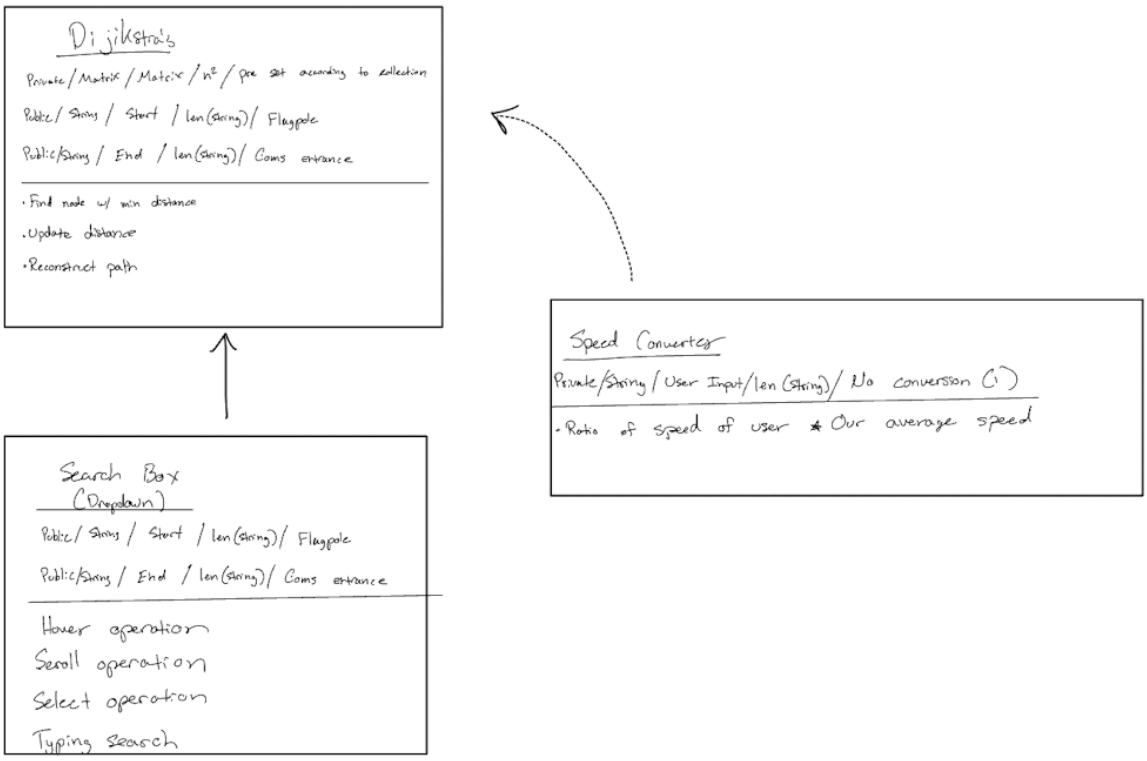
Basic Flow:	Alternative Flow(s): Variations on the basic flow
1. The user accesses the system. **{ Load System }** 2. The user inputs the starting point **{ Process User Input for Staring Point }** 3. The user inputs the destination **{ Process User Input for Destination }** 4. The system finds the most accessible path from the starting point to the destination that avoids stairs **{ Run Accessible Algorithm }** 5. The system outputs the shortest accessible path from the starting point and the destination on a map **{ Display Results }** **{ Terminate System }**	Specific Alternative Flow: At **{ Run Accessible Algorithm }**, if there exists no such accessible path from the user-selected starting point and destination, the system displays "Sorry, there is no accessible path from [starting point] to [destination]. Please choose a different starting point or destination." Resume the basic flow at **{ Process User Input for Starting Point }** Bounded Alternative Flow: At any point between **{ Process User Input for Staring Point }** and **{ Display Results }**, if the network connection is lost, Display "Connection lost. Please check your network connection." Resume the basic flow at **{ Load System }**

3 System Description

The diagram and descriptions should be more precise and detailed.

Draw a block diagram to show how the proposed system will interact with external services, databases, etc. Clearly mark the boundaries of the system.

Use the above diagram to introduce the system. What are the main elements of the proposed system?



In our past sprint, we focussed on these parts of the system. The main elements we discussed were the Dijkstra's algorithm, the search box to determine start and end points, and a speed converter for accurate time estimations. The diagram illustrates how they all interact with each other.

4 Current Status

Summarize the current implementation status of your system.

Our current implementation status revolves around our code that implements dijkstra’s algorithm.

dijkstra(matrix, start, end): This function implements Dijkstra's algorithm to find the shortest path between the **start** node and the **end** node in a graph represented by the adjacency matrix **matrix**. The algorithm maintains a list of visited nodes, a list of distances from the start node to every other node (initialized to infinity), and a list to keep track of the previous node in the shortest path. The function iterates over all nodes, updating the distances and previous nodes based on the current minimum distance node. After traversing all nodes, the function reconstructs the shortest path from the end node to the start node using the previous nodes list and returns the shortest distance and the path.

4.1 Screenshots

Add the screenshot(s) of the system's working part(s). Explain the screenshot, i.e., explain (1) what you are trying to show in the screenshot, (2) which part in the system description diagram the part belongs to, (3) what is the behavior of the system, and (4) why this system important.

```python
import numpy as np

def create_random_matrix(num_nodes):
    # Create a matrix with random distances, using -1 to represent no direct connection
    matrix = np.triu(np.random.randint(0, 10, size=(num_nodes, num_nodes)).astype(float))
    matrix += matrix.T  # Mirror the upper triangle to the lower triangle
    matrix[matrix == 0] = float('inf')  # Replace 0 with infinity to indicate no connection
    np.fill_diagonal(matrix, 0)  # Set diagonal to 0 (distance to self is 0)
    return matrix

def dijkstra(matrix, start, end):
    num_nodes = len(matrix)
    visited = [False] * num_nodes
    distance = [float('inf')] * num_nodes
    distance[start] = 0
    previous = [None] * num_nodes

    for _ in range(num_nodes):
        # Find the node with the minimum distance that hasn't been visited
        min_dist = float('inf')
        for node in range(num_nodes):
            if not visited[node] and distance[node] < min_dist:
                min_dist = distance[node]
                current_node = node

        visited[current_node] = True

        # Update distances for neighbors of the current node
        for neighbor in range(num_nodes):
            if matrix[current_node][neighbor] != float('inf') and not visited[neighbor]:
                new_dist = distance[current_node] + matrix[current_node][neighbor]
                if new_dist < distance[neighbor]:
                    distance[neighbor] = new_dist
                    previous[neighbor] = current_node

    # Reconstruct the path
    path = []
    current = end
    while current is not None:
        path.append(chr(65 + current))
        current = previous[current]
    path.reverse()

    return distance[end], path

# Example usage
num_nodes = 5
matrix = create_random_matrix(num_nodes)
print("Matrix:")
print(matrix)

start, end = 3, 4
shortest_distance, path = dijkstra(matrix, start, end)
print(f"Shortest distance from {chr(65 + start)} to {chr(65 + end)}: {shortest_distance}")
print(f"Path: {' -> '.join(path)}")
```

What is being shown: The screenshot shows the complete code for generating a random graph and finding the shortest path between two nodes using Dijkstra's algorithm. Two functions are visible: **create_random_matrix**, which generates a matrix to represent the graph, and **dijkstra**, which calculates the shortest path.

Part in the system description diagram: This code represents the core logic of the system—the algorithm implementation component. In the system diagram this is the core logic for the section called **dijkstra.**

Behavior of the system: Upon execution, the system will first create a random graph with specified node counts. Then, the **dijkstra** function will find the shortest path between two specified nodes. The system maintains a list of visited nodes, a list of distances, and a list of previous nodes to reconstruct the path. In the future, this will instead run the data for all of campus instead of the random matrix

Importance of this system: This system is critical because Dijkstra's algorithm is a well-established solution for pathfinding problems. By using it we know we are efficiently finding the most efficient path for the customer.

```
Matrix:
[[ 0.  7. inf  1.  4.]
 [ 7.  0. inf  6.  5.]
 [inf inf  0.  5.  6.]
 [ 1.  6.  5.  0.  9.]
 [ 4.  5.  6.  9.  0.]]
Shortest distance from D to E: 5.0
Path: D -> A -> E
```

What is being shown: The output consists of the randomly generated matrix, which represents the adjacency matrix of the graph, and the results of running Dijkstra's algorithm on this graph, showing the shortest distance and the path from node D to node E.

Part in the system description diagram: This relates to the **dijkstra** part of our system. It is the output of that part of the system. Similarly, this output relates to the system's user interface or the results display component. It is what the user would see after the system has processed the input (the graph) through the algorithm (Dijkstra's algorithm).

Behavior of the system: When the system runs, it creates a matrix representing a graph with weights on the edges between nodes. Some edges are marked as 'inf' (infinity), indicating no direct path between those nodes. After processing this graph through the **dijkstra** function, the system outputs the shortest distance and path from node D to node E. In this case, the shortest path is D -> A -> E with a total distance of 5.0.

Importance of this system: The output is crucial as it visually confirms the correctness and effectiveness of the algorithm. It allows users to verify the graph's structure and the algorithm's performance in real-time. This kind of output is essential for debugging, verifying, and understanding how Dijkstra's algorithm navigates the graph to find the most efficient path. It's a tangible result that can be applied to various practical scenarios, such as logistics optimization, network design, and urban planning.

4.2 Tests

List the tests you performed for this iteration's implemented parts.

- The only tests performed in this iteration were on Dijkstra's algorithm. We utilized a series of different matrix representations to ensure that the algorithm itself was working.
- Since this iteration was so heavily focused on data collection and the one algorithm, we didn't have as many tests as we will in the next iteration.

From the acceptance tests in the proposal, explain your plan to test them technically. In other words, how would you automatically test your accep- tance tests? What are the inputs to the tests and outputs from the tests? How would you determine whether the test was passed or failed?

1. Given a list of class locations for the day, when the student inputs these into the app, then the app should display the most efficient route covering all locations.
    a. If the app does not display the most optimized route according to our test cases based on the given input, then we know that the given test case has failed.
2. Given a set time frame for a campus tour, when the visiting family inputs their start and end points, then the app should provide a route that can be completed within the allotted time.
    a. The input will be a specified time for a campus tour. The output should be a tour route which can be completed in the allotted time. If the output can not be completed in the time given (according to our data collection and written test cases), then we know that this test has failed.
3. Given a faculty member's class schedule, when they integrate this schedule into the app, then the app should suggest optimal routes based on the timing and locations of their classes and meetings.
    a. This is something that will be simple to test along with test #1. The input is different locations on campus and the output should be the optimized path for that specific order/route. If it is not the optimized path which we will have hand calculated, then we know it is not functioning correctly and has failed the test.
4. Given a prospective student's interest in specific campus landmarks, when they select these landmarks in the app, then the app should provide a route that includes these points and offers information about each landmark.
    a. This will be tested through inputting specific Davidson landmarks and ensuring that the recommended tour route accounts for these.
    b. If the landmarks are not a part of the recommended tour route, then we know that the test has failed and our implementation for this specific functionality is not working correctly.
5. Given an unexpected event or high pedestrian traffic on a usual route, when the staff member checks the app, then the app should suggest an alternative route to avoid delays.
    a. This pedestrian traffic/event functionality is something which we have not focused on. It will be difficult to keep up and running, particularly after we leave school (unless T&I or another student org took over). Therefore, during this process, this is a test that we do not

5. Project Management

Continue to maintain the Change Log. Add any new changes to the project, tracking the date and description of each change. Use the table below:

Date Description
Date the change was made A summary about the change made to the system

Date	Description
March 18th	Met with Maggie Woodward who helped us determine the change to make the system organized by building type. The implications will be as we start developing the UI, we will have to cater to her requests/suggestions.

March 19th	Met with Cate Goodin who helped us determine that it would be useful to have a scenic and accessible option. This will require a slight modification to the algorithm but should not be too difficult.
March 20th	Met with Matthew Bernard who said that the gaming aspect was not particularly necessary. No implications besides there not being a need to code an additional gaming system.

6. Review and Retrospective

- What went well?
    - Customer meetings were really successful. They gave us helpful information about what our users might be looking for in terms of categorization and potentially how inclusivity and accessibility could be a more integral part of WildCatWays
    - Full Data collection is something important that we needed for this iteration in order to move into our software.
    - Similarly, Dijkstra's algorithm was good to get done, as we need to focus on the front-end at this point in the semester.
- What didn't go well?
    - We were able to meet, however not as frequently as necessary at this point in the semester.
    - Our focus on the back-end was intentional, however our lack of focus on the UI is something which needs to be adjusted..
- For the goals that were not met, what were the issues?
    - Schedules are busy in college so setting up a time where all 4 of us can meet is challenging.
    - We focused heavily on back end and data collection so have no front-end to really show at this point.
- How do you plan to overcome the issues?
    - We are going to sit down with our schedules so we can better finalize 1 solid time a week we can meet. Otherwise, we are making sure that we split up the work so that it doesn't require all 4 of us to be there for a meeting.
    - We are going to focus more heavily on the front-end, while still making sure the back-end is accomplishing our goals and has the data integrated.
- What do you plan to do differently in the next iteration?
    - Ramp up the amount of meetings, focus more on front-end, split up work a bit more so that all bases are being covered.

7. Team Management

What were the team roles for this iteration? What did each team member contribute?

- Each team member manually collected data on the nodes, the edges connecting each node, and the weights of each edge (the time it takes to walk the path from node A to node B). We divided the campus into four regions and each collected data for our assigned region.

What were the challenges regarding team management, e.g., regular meeting, etc.?

- We all have very different class schedules and conflicting extracurricular activities after class, so it was hard to find a set-time that works for everyone to meet weekly.

What are the plans to overcome the challenges?

If you were the third party who knows very well about your team, what suggestions would you give to your team?

8. Goals for the Next Iteration

Next iteration product log:

- As a user, I want a UI map presentation, so that I can see my optimized path.
- As a user, I want the demo to have data for all buildings on campus, so I can have an optimized path no matter where I'm going on campus.
- As a potential new student, I want WildCatWays to have different categories, so I can pick destinations based on categories since I don't know the campus well.
- As a customer, I want a working demo, so that I can offer feedback for the final product rollout.

Next iteration Sprint log:

- Data integration
- UI (Front-Back link)
- Functionalities
    - Categorizing (food, school buildings, etc.)
- Bug assessment of working demo

Other than the issues discussed in Section 6, i.e., Review and Retrospec- tive, what potential challenges do you see in the next iteration?

- Google Maps API implementation/linkage to back-end development

I see some potential challenges with usage of the google API. It is going to require a pretty concrete understanding of how the technology works and how to make adjustments to it, which without having spent a lot of time using it, we could foresee being challenging.

Briefly explain how your team would overcome each of the mentioned challenges.

Time at task is going to be incredibly important for this next iteration. As a team, we are going to dive into understanding the fundamentals of the google maps API and its linkage to the back-end timing which Google Maps offers. Once we have that concrete understanding, we feel confident that we will be able to accomplish our goals of making our own Google Maps style API for Davidson's campus using our algorithm and personalized data.