Jumana Schmidt (jumanas2), David Medina (davidrm3), Roy Lu (roylu2), Will Marshall (wjm5)

Project Proposal: Optimizing the Climb

1. **Leading Question:** *How and Why Find the Path of Least Resistance Up Any Terrain?*

For this project, we intend to find the shortest and most energy efficient path from one point to a specified destination. There are countless reasons as to why this project is not only useful but also extremely practical. Our program will virtually be able to take a black and white image of almost any land-based terrain, a destination, and a starting point, and be able to find the path of shortest distance and least resistance. This program is applicable for hikers, rescuers, GPS, real estate developers, and more. To be able to demonstrate the full ability of our project, we'll be using a picture of a mountainous terrain as our input, as there is more variation in height. Additionally, we'll be referring to the person/thing trying to find the best path as the hiker, in our code and our project's description, but in reality, anyone with any purpose could find use for this program.

The program should take in a black and white image of a heightmap of some terrain, and this image will essentially be our dataset. Using the PNG class, HSLAPixel class, and our own class, it will save each pixel and its luminance as a graphical node. The luminance will essentially be the height of the terrain at that pixel/node. Additionally, our class will also take in starting and ending pixels which will be saved as specific nodes into the class' private members section. Lastly, we will also have a private variable that increases every time the hiker expends energy, so, more specifically, the variable will increase at a greater value if the hiker traverses to another node of a greater height and lesser if the next node is of a lower or even with the current height. Thus the purpose is to end up with the smallest energy consumption value.

First, we will implement Dijkstra's Algorithm to find the shortest path while expending the least amount of energy. This algorithm will end up giving us a definite and most optimal path. Second, we plan on using A* to approximate the best path in order to compare the first algorithm's output. We also want to test which has a faster runtime, since A* is approximating this calculation. Finally, we want to output a 3-D rendering of the terrain along with the best path the hiker ended up taking. Likely, we will use OpenGL to accomplish this render. Overall, this project gives a solution to a very practical problem and a visual representation of the best path to take, which could definitely help many of the people mentioned above and more.

2. **Dataset Acquisition and Processing**:

    a. *Data Format:*

The input will be a PNG image of a heightmap, so part of our algorithm will need to be converting that image into our dataset. The pixel's luminance and location values will be stored as graphical nodes. Likely, the image will be of lower quality, so this storage option will be viable and easier to traverse. For this part, we will use the entire dataset, but depending on how far and where the destination and start nodes are set.

    b. *Data Correction:*

For the algorithm, its method of parsing the data is described more thoroughly above, including its edge cases. For actually implementing the erosion, we will definitely have an edge case of not allowing the algorithm to darken below the viable range of luminance. We also cannot traverse or edit pixels outside of the image/array, so we need a bounds check throughout.

    c. *Data Storage:*

The sediment, height, and luminance values will be stored in a *2-D array format*. Likely, the image will be of lower quality, so this storage option will be viable and easier to traverse. For this part, we will use the entire dataset, but depending on how many pixels water is placed on, the algorithm could only access a part of it (ie. the part of the image the water traverses).

3. **Graph Algorithms**:

a. *Function Inputs:*
    i. The input for the hiker algorithm will be a black and white image height map. This image will have a max size of 600 x 600 pixels. This will allow us to easily create a 2D array that will hold the pixel data for each pixel. The pixel data will be read using the HSLA pixel class that is within the PNG class. We decided to go with array formatting since it allows us to work with images that are visually easy to test and prototype code. Within this image, we would have to create a system of energy for the hiker. In our case, it will be tracked by jumps per pixel. Say the hiker moves to a pixel ahead with a luminance difference of 3 to its current pixel. The system would take this difference and from it calculate the energy cost that it would take for the hiker to make this climb. It is worth noting that the energy expenditure for a 2 luminance difference climb must be exponentially larger than that of a one luminance difference climb and not linearly related, otherwise every path may come out to the same energy expenditure.
    ii. For this project, we will be using two graphing algorithms to traverse the inserted image. The use of the shortest distance algorithm is needed to keep track of the energy used by the hiker as they look for the shortest(least energy-intensive) path. For this, we'll be using the shortest path BFS algorithm. Specifically, Dijkstra's Algorithm since we feel that we can map nodes on every 10 pixels as markers for the algorithm since having nodes equal pixels may have heavy overhead. We could have our image act as an Adjacency Matrix of size vxv where v is the number of vertices in the graph. This will be our very large weighted graph, the shortest path will be used.
    iii. At the moment, the second algorithm will be implementing the same thing as the first algorithm but in a different way. For this, we will use the A* algorithm since we feel this pathfinding algorithm already comes with a heuristic function that can estimate the cost of the shortest path. We could even expand and feed it a general guess of where the hiker wants to go and leave the A* algorithm to calculate the shortest path with that goal node in mind.
    iv. Extras:
        1. For the heightmap for the OpenGL 3D plane image, we'd need to take the original greyscale image and run it through another algorithm that will define the vertices of the 3D image. We are basically creating triangles by plotting this mesh and then adding the height depending on the greyscale factor. This while requires a 2D array input of the pixel and then another function to place into a new 3D array for plotting in OpenGL
        2. We could use Minecraft to plot the height map and maybe the path the hiker would take with certain blocks.
b. *Function Outputs:*
    i. The expected output of the hike algorithm is an image with the path of least energy spent drawn for the hiker. It is similar to a GPS, except it will tell the hiker what path is the easiest for their body.
    ii. For fun, a 3D rendering program called OpenGL could be used to render the image in 3D. We'd read the file into an array and use the hike algorithm to decide what to color in. From here apply the z-axis conversion depending on the greyscale value. This would then create a height map of the image that we can render in 3D. This will then allow us to create a 3D plane map using the "OpenGL terrain rendering demo". Or alternatively, Minecraft was the team's fun option since it's easy to see visually where the path is.

c. *Function Efficiency:*
    i. The time complexity to traverse a 2-D array is found to be O(n*m), where n is the number of arrays, which correlates to the first dimension of our 2-D array, and m is the max size of the arrays, our second dimension. Knowing this the actual time complexity can be described as O(N), meaning it is linearly related to the size of the input.
    ii. For Dijkstra's Algorithm, the time complexity would be o(n^2) however if we get the image pixels imputed as Adjancey list, then the complexity would drop too(E log V). V being vertices and E being edges. Unfortunately, this would require us to heavily limit the size of the imputed data image. Since having too many vertices and edges would lead to a large overhead.
    iii. For the A* algorithm, the worst case for us would be that it searches for unbounded and takes o(b^n). Where b is the branching factor, the number of children at each node. If we bound it and give A* a general direction then it improves to o(E), where E is edges. Once again, limiting the size of the number of leafs will be needed so that it can even run.
    iv. The time complexity for the OpenGL heightmap output is heavy. Essentially we are generating two triangles per mesh map tile and uploading them to get processed by the OpenGL software. Since we are also working with arrays here and a new 3d array most likely, the time complexity will jump to O(n^2) worst since we'd have to iterate through three times.
    v. Goals of time complexity:
        1. As mentioned earlier, the time complexity for the hiker algorithm has the possibility to have high overhead due to the ever-expanding search for the closest nodes. We want the input to be an image that we iterate through. The main build for this project will most likely be mp_ traversals. Combining the main algorithms would lead to a worst-case of o(E*n^2). We were warned that this could lead to long compile times as the image gets bigger. If it reaches a size that's bigger than then the available heap memory of a member's computer, it may crash. To counter this, we hope to start with small images and slowly upscale them. We hope that the A* algorithm will allow us to expand the images since it runs on an o(E) time complexity.

4. **Timeline**:

Week 1 (March 28th - April1st): **Rewrite project proposal**: Meet with TA, figure out our project, and find which graph algorithms we're using.

Week 2 (April 4th - April 8th): **Prepare Input / Learn About OpenGL**: Find a black and white height map for input and write code to process it with PNG class and make each pixel a graph node with a luminance value that acts as the height. Write the class structure with private variables for the hiker's location and energy expenditure as well as for the destination node. Learn the basics of OpenGL to keep in mind for final terrain output generation.

Week 3 (April 11th - April 15th): **Class Structure / Start Presentation**: Write pseudocode for basic Dijkstra's Algorithm to find the shortest path between hiker and destination nodes as well as optimize energy consumption for the hiker. Account for edge cases in the graph.

Week 4 (April 18th - April 22nd): **Finish & Test Dijkstra's Code / Start Writing A* Code**: Finish writing functional code for Dijkstra's Algorithm. Compare predicted Big O with reality. Start writing code for the same process/goal but with A* to approximate the best path for the hiker to take. Maybe come back to OpenGL & start planning how to generate output now that we have basic algorithms.

Week 5 (April 25th - April 29th): **Finish & Test A*'s Code / Start Visualization of Output**: Finish writing functional code for Dijkstra's Algorithm. Compare predicted Big O with reality. Start writing code for OpenGL final output.

Week 6 (May 2nd - May 6th): **Prepare Final Output / Finish Presentation**: Finish complete functional code for each algorithm. Finish and test OpenGL 3D model and find Big O. Finish presentation