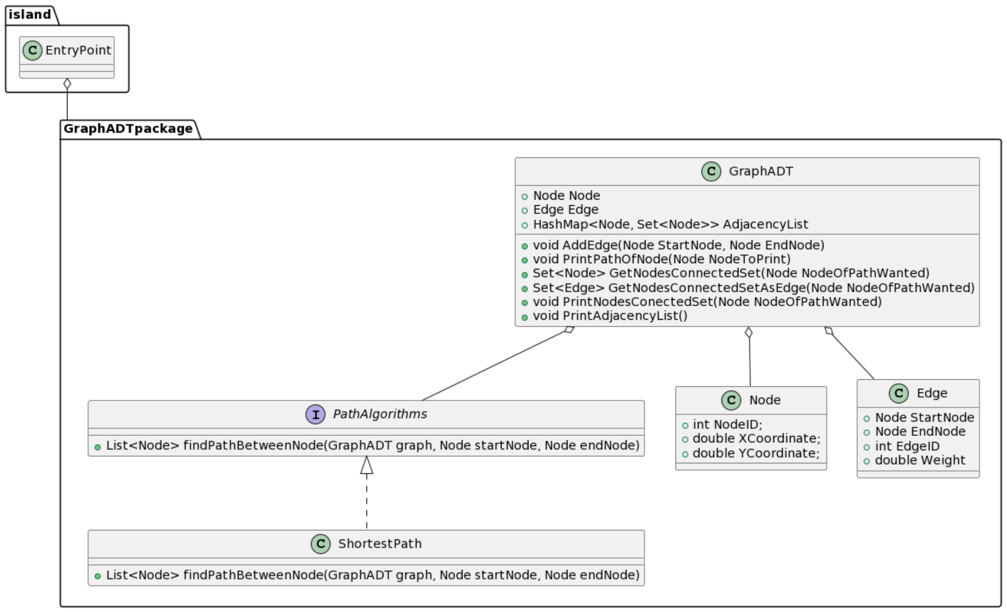
Questions

1) Provide a class diagram of your pathfinder library.


island
EntryPoint
GraphADTpackage
GraphADT
Node Node
Edge Edge
HashMap<Node, Set<Node>> AdjacencyList
void AddEdge(Node StartNode, Node EndNode)
void PrintPathOfNode(Node NodeToPrint)
Set<Node> GetNodesConnectedSet(Node NodeOfPathWanted)
Set<Edge> GetNodesConnectedSetAsEdge(Node NodeOfPathWanted)
void PrintNodesConectedSet(Node NodeOfPathWanted)
void PrintAdjacencyList()
PathAlgorithms
List<Node> findPathBetweenNode(GraphADT graph, Node startNode, Node endNode)
Node
int NodeID;
double XCoordinate;
double YCoordinate;
Edge
Node StartNode
Node EndNode
int EdgeID
double Weight
ShortestPath
List<Node> findPathBetweenNode(GraphADT graph, Node startNode, Node endNode)

2) Explain why your code is SOLID and locate your technical debt. If you have used any pattern, describe them and justify your decision.

For my code, I followed SOLID by first using single responsibility. Each class in my Graph Package only has one responsibility. This makes it so that I can debug easier, and the code is a lot more readable. Second, I have the open closed principle. For this I just did as it stated. Once I finished with a class in my ADT, I left it untouched. It can only be extended, and I add all the methods that will be needed by the class before closing. Third I have the Liskov Substitution Principle. The only case here that was used would be my Path finder algorithm. Since the PathAlgorithms interface is abstract, it can be switched out with my ShortestPath class. It makes it so each are substitutable. Fourth is the Interface Segregation. There was not much here I could do in my project. Everything I created for GraphADT needed to be used, so there was no need to hide anything. Finally, I have the Dependency Principle which can be seen in my abstraction for the PathAlgorithms Interface. We depend on the interface instead of the actual class. We don't care how we get the path; we just want it. For my pattern, I used the façade pattern for my entry point. Although this pattern can be messy, I had a lot of small distinct factors that could be accessed. This meant the façade pattern could access any things I had to bundle together.

3) Explain why your tests are "reasonable" (refer to CORRECT/BICEP if needed

First, I used the Ordering part of CORRECT to make sure my testing is reasonable. I ordered it by doing nodes, edges, Graph, Path in this order. This means I work my way up for the test from the ground up making sure that the graph I created was right. Next, I used Oracles to make sure that was I tested was right. These oracles did not change and in doing so I made sure that what I was creating matched up with the oracles. Finally, in BICEP, I checked for the trivial functionality of the code. I made sure that when I tested each standard function like creating the adjacency list, creating a path, and making our edges all worked. I also tried to check boundary for cases that were close to failure with nodes and edges.

4) Which design patterns have you used to support integrating your pathfinding library into your generator?

For my path finder class, I used the abstract factory pattern. We have an interface this means that we can create the shortest path or any path algorithm that we want in the future without caring about how it is done. In the future, we will add methods and algorithms that we want into the interface what the interface will use that can be grouped together. We have multiple ways of find the path, but we do not care how or which one to choose from.

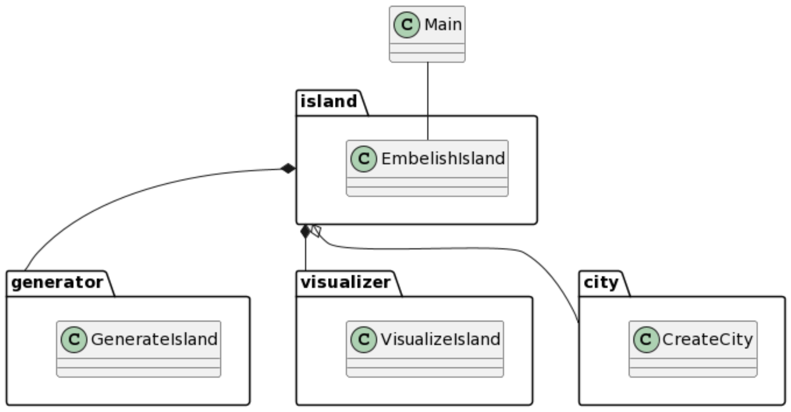
For my GraphADT class, I implemented the singleton pattern. This means that we have one Graph for our whole Mesh. The GraphADT will never create a second graph that it can use unless it needs to do unit tests. I tried to implement it by only ever creating one instance of the graph in my mesh. This make it so that we only create the Graph when needed and our graph and adjacency list is encapsulated into our graph.

5) What about performances? Do you envision any scalability issues in your road network generation?

For performance of my GraphADT, I went with an adjacency list to represent my graph. I first went with a matrix representation, but as soon as I added my nodes; it created a huge matrix that took up way too much memory. This meant I failed and decided to go with an adjacency list which would provide much better performance on my graph. Likewise, this means that as we scale up the product in terms of graph size, we can take on any size graph and path needed. However, if we were to add the bonus or extra features, we would need quite a bit of refactoring. But overall, this scalability really depends on what we are adding. I believe that reasonably sized scalable tasks would be easy to do, and I am prepared for that in my work.

6) Draw a high-level sequence diagram explaining how your generation process works from a coarse-grained point of view. The point is to document the whole process, not only the urbanism part.

Extremely High level. Imagine that the classes in each package are multiple classes that draw what we need.



7) Locate your technical debt in the generator. How can one extend it by adding a new generation mechanism in the generator? For example, if one plans to introduce resource production mechanisms (e.g., forests produce wood, lakes produce fish, …), how would it be supported by your project?

My technical debt I believe lies in the shortest path. I could not get the path to be completely straight, and I have ideas why, but am unsure on how to fix them. The paths are pretty straight, but sometimes cannot be too accurate. To extend, we know the generation just creates our polygons and adds no embellishments to it. For this, we know that if we wanted to add wood to a forest for example, we are unable to add a specific request to change the polygon's property or change maybe the parameters of a polygon that has wood in it. To do this, we could create a library that would take in our polygons at the generation stage and add our requirements to the polygon. This would be like how the GraphADT changes the visualizer, but instead we are changing some of the properties with the polygons and adding wood or fish. Additionally, we can think of patterns that could work, or if patterns don't work, we could use more basic principles.

Self-Reflection

1) Backward: What process did you go through to produce this result?

To produce the result that I wanted I had to fail a lot. Failing meant that I wrote a lot of iterations of the same code, tested it, and tried again as it fundamentally didn't work. I started with my Graph ADT being completely wrong and way too convoluted. So, from the code I wrote I scrapped everything I did and started again. The main process I learned to produce my result was "fail early, fail often". The process also included planning. There were a lot of things I didn't expect and through this failed a lot.

2) Inward: What were your standards for this piece of work? Did you meet your standards?

I think what I produced met my standards. My main standard for the work is that it worked to create a mesh with the embellishments and our code had good OOP structure. Our first section was not really that good, but my Assignment 3 really improved. I can always go back and redo a lot of the code that seems obvious to fix now that there is no grade attached. This means although some parts don't meet my standard now, I am able to go back and fix it. Likewise, we reached a lot of the OOP standards we set. We tried to implement what we were learning in class to our code, and we did a pretty good job especially with the city implementation.

3) Outward: What is the one thing you particularly want people to notice when they look at your work?

I think the different features we add will be the main mesh is what I want people to notice. In A3, we were able to get the bonus done. This means that there are multiple ways someone can view our graph. This aspect is cool because it takes the same mesh and creates something that is completely different. Adding all the arguments will also be something that stands out. Customizing the mesh will be something that I want the user to notice, and I am proud of.

4) Forward: What lessons will you keep from this assignment in your professional practice?

From this assignment I learned a lot. I really understand how OOP works and what it means to have good practice when creating code. Also, I learned a lot about design software and how we can implement these ideas into our own work. Likewise, I really learned the importance of data structure and how you should always think before deciding what structure you want to use.