Group 26: Optimal Shipping Routes

Anton Gudonis, Dallin Whitford, Aidan Wolf, Liam Seagram

Summary:
We have a map of set size with set landmass locations in the form of a large grid. On the map, we will have 9 ports generated with varying distances from each other. The ship will be set to start at one of these ports. Each port generates with a type of cargo that it has, and a type of cargo that it needs, out of the three choices of cargo.
We need to generate a ship route that will deliver the type of cargo our starting port has to a port that needs it, then pick up the type of cargo that port has, and deliver it to a port that needs it, until we have visited all 9 ports and satisfied all their needs, in the least amount of tiles travelled.

Propositions:

- Land(x,y)
  - True if the xy location is land
- Water(x,y)
  - True if the xy location is water
- Ship_n(x,y)
  - True if the xy location has a ship on it
  - n is the copy of the ship at time n (first is 0, +1 for subsequent steps in time)
- Previous(x,y)
  - True if the xy location was previously visited by the ship
  - Maybe use it as a counter?
- Cargo_n(type)
  - Type can be Produce, Appliances, Cars
  - True if the ship is currently carrying a certain kind of cargo
  - n is the copy of the cargo on the ship at time n (first is 0, +1 for subsequent steps in time)
- Port_n(x, y, has_cargo_type, wants_cargo_type)
  - The x,y locations will be randomly generated on our map
  - has_cargo_type, and wants_cargo_type: 3 Possible types.
  - True if the xy location has a port and the ports wants and has are specified by cargo types
  - An empty cargo type will be assigned to both a port's wants and has once they are both satisfied.
  - n is the copy of the port at time n (first is 0, +1 for subsequent steps in time)
- Wildcard(x, y, type)
  - True if location is "Special" and in the water
  - Types:
    - Whirlpool, Moves the ship 2 nodes forward instead of one
    - Kraken, throws the ship 2 nodes left and 3 nodes forward. The throw must land in water, so it is cut short if it crosses land

        - Tidal wave, moves you one node to the right, or to the left if the node to the right is land

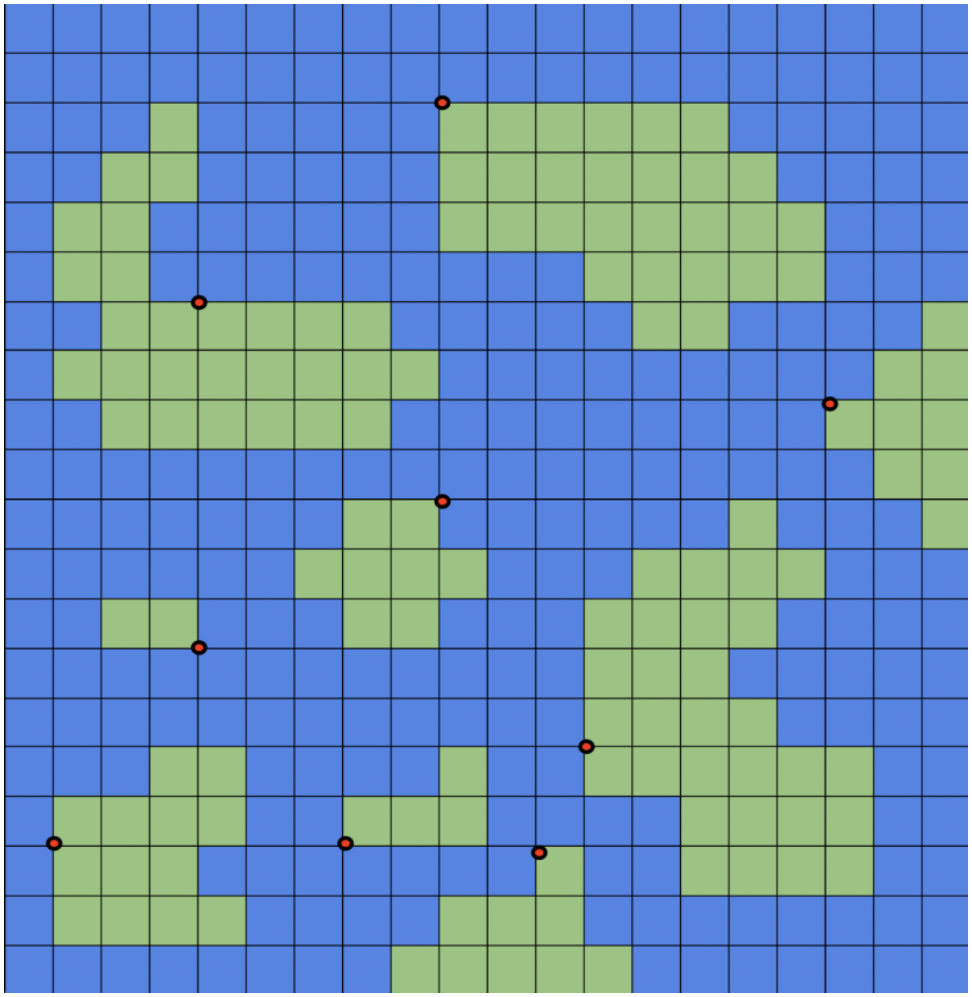
(The wildcard proposition is not at all implemented yet, and we were thinking that would come along once we have our base problem laid out and solvable)

Constraints:

- Port is on edge of land
    - Port_n(x,y,has_cargo_type, wants_cargo_type) ∧ (Water(x-,y) ∨ Water(x+1,y) ∨ Water(x,y-1) ∨ Water(x,y+1))) ∧ Land(x,y)
- Ship can't touch land unless ship is on port
    - ((Land(x,y) ∧¬Ship_n(x,y)) ∨ (Land(x,y) ∧ Ship_n(x,y) ∧ Port_n(x,y))) ∧¬((Land(x,y) ∧¬Ship_n(x,y))∧ (Land(x,y) ∧ Ship_n(x,y) ∧ Port_n(x,y)))
- Water and land are mutually exclusive
    - (Land(x,y) ∨ Water(x,y))∧¬(Water(x,y) ∧Land(x,y))
- A port cannot both have and need the same cargo
    - ¬Port_n(x,y, (produce), (produce)) ∧¬Port_n(x, y, (appliances), (appliances))∧¬Port_n(x, y, (cars), (cars))
- The ship must pick up the cargo that is at the port it has travelled to, and drop the cargo it has. The port's wants and has values will then be set to 0. Since the ship is specified a cargo type at the start this incentivises the ship to move.
    - Ship_n(x, y) ∧ Port_n(x, y, (has_cargo_type), (wants_cargo_type))∧ Cargo_n(has_cargo_type) → Port_n+1(x, y, (0), (0))∧ Cargo_n+1(wants_cargo_type)
- If the ship is on tile (x,y) then it must have been on (x+1, y) or (x, y+1) or (x-1, y) or (x, y-1) or if the ship is on the starting tile at time 0 then the ship must move to (x+1, y) or (x, y+1) or (x-1, y) or (x, y-1) (these 4 tiles are mutually exclusive pretend the disjunction is xor)
    - (Ship_n(x,y)→((Ship_n-1(x+1, y) ∨ Ship_n-1(x, y+1) ∨ Ship_n-1(x-1, y) ∨ Ship_n-1(x, y-1))) ∨ (Ship_0(start_tile_x, start_tile_y)→(Ship_1(x+1, y) ∨ Ship_1(x, y+1) ∨ Ship_1(x-1, y) ∨ Ship_1(x, y-1)
- If a ship was on a tile then the previous proposition is true
    - Ship_n-1(x,y)→Previous(x,y)
- (Wildcard constraints later)
- enforce with constraints that all the ports start out lacking the cargo they need at time 0, and must have the cargo they need by the final time step
- Need to capture goal: if the list of previous tiles equals the list of port tiles then the goal has been reached,
    - If the ship has visited all ports and ports wants and has are both set to zero then win
        - Previous(x,y) ∧ Port(x,y,0,0) then win (not sure of modelling)

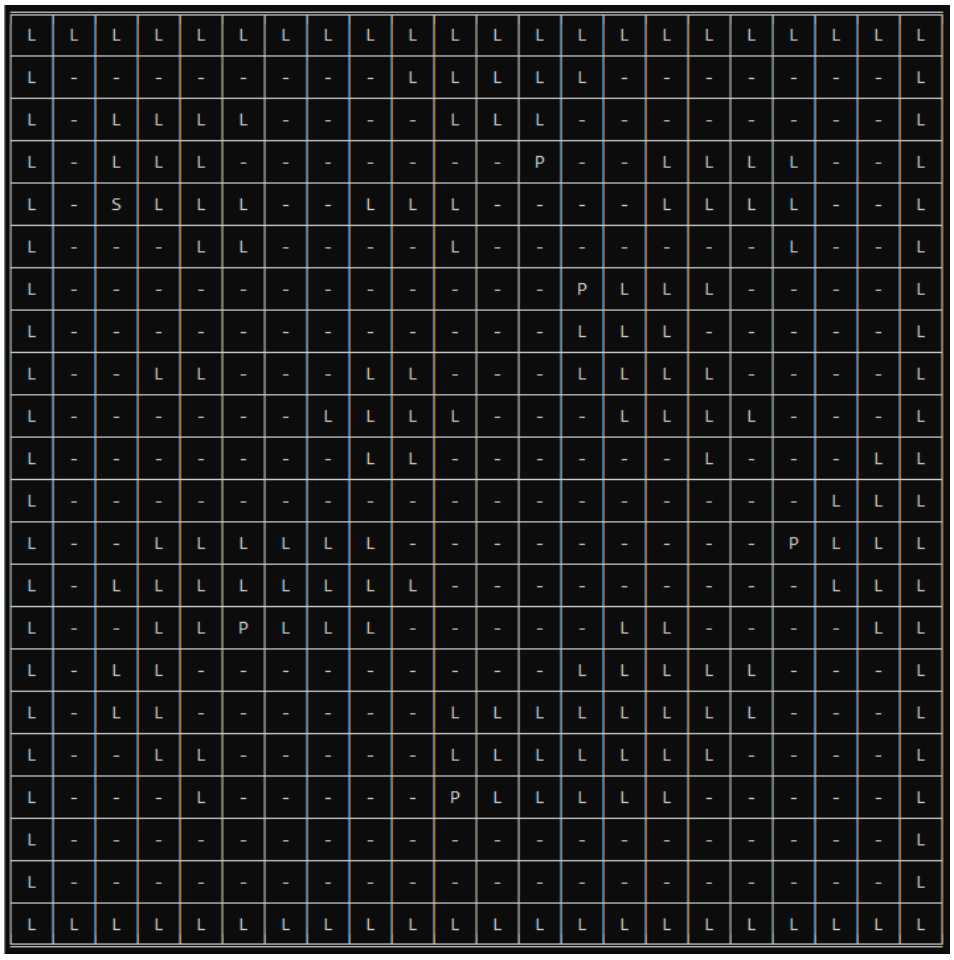
Model Exploration

Crucial to our model is the map that the program will be interacting with, this will mostly stay static, but hopefully we can implement a feature where the locations of our ports will be somewhat random. Deciding how to do the map was tough, but we eventually decided that having set landmasses that the program interacts with would be best- randomly generating them may add too much complexity and constraints to our already complicated model. The original idea was also to have nine ports, as that would make the prettiest solution to our problem. Here is an early draft of what the map could look like:



This version had the ports on “nodes” instead of tiles, with the idea that the ship would travel via the nodes, but we eventually decided that having tiles would work best for our implementation of the code.

Here is an example of what the map currently looks like, as generated by our python file, illustrated using tabulate:



L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
L	-	-	-	-	-	-	-	-	L	L	L	L	L	-	-	-	-	-	-	-	L
L	-	L	L	L	L	-	-	-	-	L	L	L	-	-	-	-	-	-	-	-	L
L	-	L	L	L	-	-	-	-	-	-	-	P	-	-	L	L	L	L	-	-	L
L	-	S	L	L	L	-	-	L	L	L	-	-	-	-	L	L	L	L	-	-	L
L	-	-	-	L	L	-	-	-	-	L	-	-	-	-	-	-	-	L	-	-	L
L	-	-	-	-	-	-	-	-	-	-	-	-	P	L	L	L	-	-	-	-	L
L	-	-	-	-	-	-	-	-	-	-	-	-	L	L	L	-	-	-	-	-	L
L	-	-	L	L	-	-	-	L	L	-	-	-	L	L	L	L	-	-	-	-	L
L	-	-	-	-	-	-	L	L	L	L	-	-	-	L	L	L	L	-	-	-	L
L	-	-	-	-	-	-	-	L	L	-	-	-	-	-	-	L	-	-	-	L	L
L	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	L	L	L
L	-	-	L	L	L	L	L	L	-	-	-	-	-	-	-	-	-	P	L	L	L
L	-	L	L	L	L	L	L	L	L	-	-	-	-	-	-	-	-	-	L	L	L
L	-	-	L	L	P	L	L	L	-	-	-	-	-	L	L	-	-	-	-	L	L
L	-	L	L	-	-	-	-	-	-	-	-	-	L	L	L	L	L	-	-	-	L
L	-	L	L	-	-	-	-	-	-	L	L	L	L	L	L	L	L	-	-	-	L
L	-	-	L	L	-	-	-	-	-	L	L	L	L	L	L	L	-	-	-	-	L
L	-	-	-	L	-	-	-	-	-	P	L	L	L	L	L	-	-	-	-	-	L
L	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	L
L	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	L
L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L

In this map, “L” represents a land tile, “P” represents a port tile, “S” represents the current tile location of the ship, and “-” represents a water tile.

In the current version, our map only has six ports- this is because we’re trying to figure out how to make all the parts of the model work together to create a solution, and having fewer ports does make this easier.

One problem that we ran into was realizing that we had no propositions or constraints for dealing with the map ending, or the ship reaching a water tile that was at the edge of the map, and trying to move beyond it. The current fix for that is just having the entire border of the map be land tiles, as our constraints are already in place saying the ship can’t touch the land.

Current state of our code:

All of our base propositions are implemented in the code, and don't produce any errors.

Ran into big issues implementing constraints. Not completely sure how to make use of the bauhaus library, so implementation here is not finished. The constraints that we've written are currently all commented out, as a lot of them don't work properly. The ones that do work are at line 403, "A port must be on the edge of land", and at line 428, "port cannot have and want the same cargo type". The other constraints do return results, but we're just not sure if they are sound logically. If uncommented they produce no errors, so feel free to test them out.

We currently don't have any way for our ship to move in the model, so we're in the process of coming up with a way to do that.

Jape Proof Ideas.

$\exists x.P(x) \rightarrow \forall y.S(y) \vdash \forall z.(P(z) \rightarrow S(y))$

$\forall x.((E(x) \vee Q(x)) \wedge \neg(Q(x) \vee E(x)))$

$\forall x.(S(x) \wedge P(x)) \vdash \exists x.\neg E(x)$

E -> Land

Z -> cargo variable

Q -> water

Requested Feedback

- Any feedback on how to use bauhaus more effectively would be appreciated.
- Guidance on understanding bauhaus output to determine what is logically sound and what is not in order to help us more thoroughly test our own code
- Do you have any ideas on implementing the movement of our ship into our code?