**Group 26:**

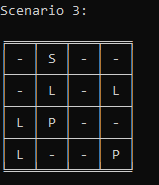
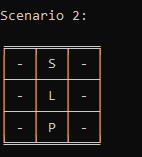
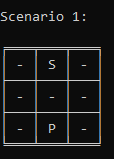
**Shipping Routes**

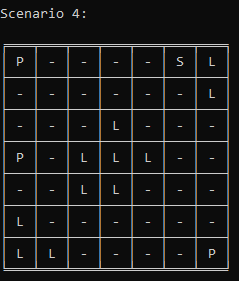
****

**Anton Gudonis, Dallin Whitford, Aidan Wolf, Liam Seagram**

**Summary:** In our model, we have a map, in the form of a grid, that contains three kinds of blocks, and one “ship” that moves on the map. The three kinds of blocks in the map are Land, Water, and Port. The aim of the model is to test whether or not the ship can travel from its starting point in the water on a given map, and go from port to port, dropping off the cargo it has at the port, and picking up the cargo that port has (satisfying the port) in a given amount of time (where each move to a square represents one time step). The model is satisfied when the ship has visited every port on the map, and swapped cargo at each one.

Here are the five scenarios that we have created to test our model:

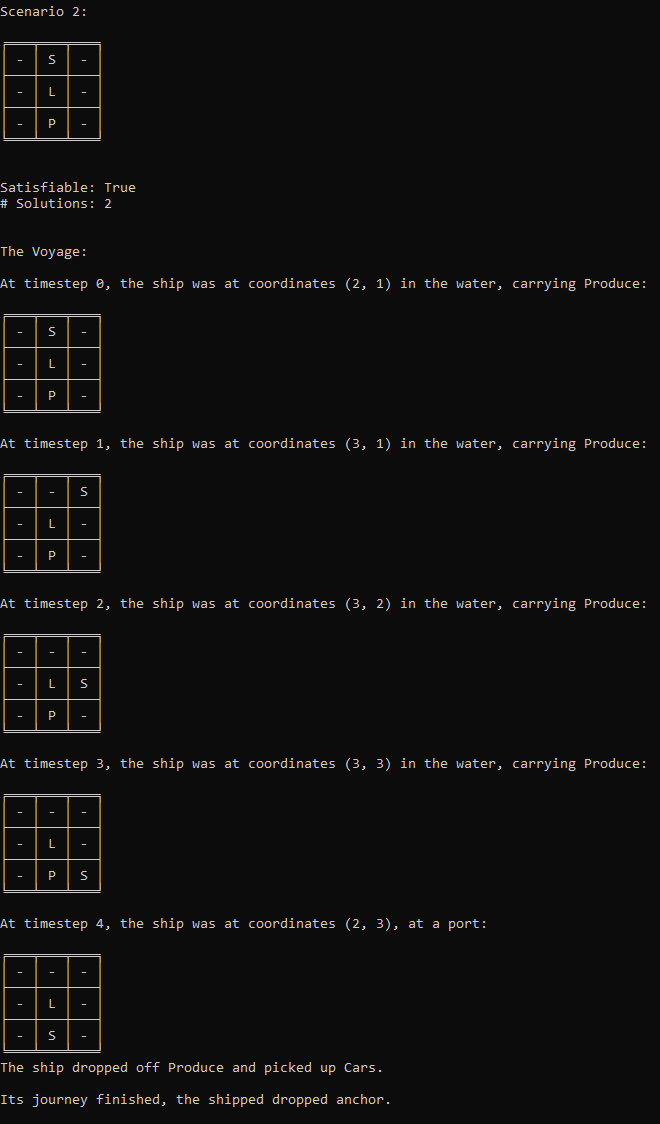




In these five scenarios, we can see all the different types of tiles- Land, represented by ‘L’, Water, represented by ‘W’, and Port, represented by P. The ‘S’ tile is also shown, as that is the starting point for the ship in each of the five scenarios, there is a water tile on the same tile as the ship. The ship will move around on the map as the model is solved, covering each tile on its path until it eventually has visited every port, and swapped cargo at each, satisfying the model and completing its voyage.

Once a timestep is inputted, the model will find the most optimal path from port to port- the ship moving one tile in any direction constitutes one “timestep”. This is achieved using propositions and constraints to create our model within bauhaus. If the ship cannot reach every port within the given timestep, the model will not be solvable.

Here is an example of the model being solved on scenario 2:

****

**Propositions:**

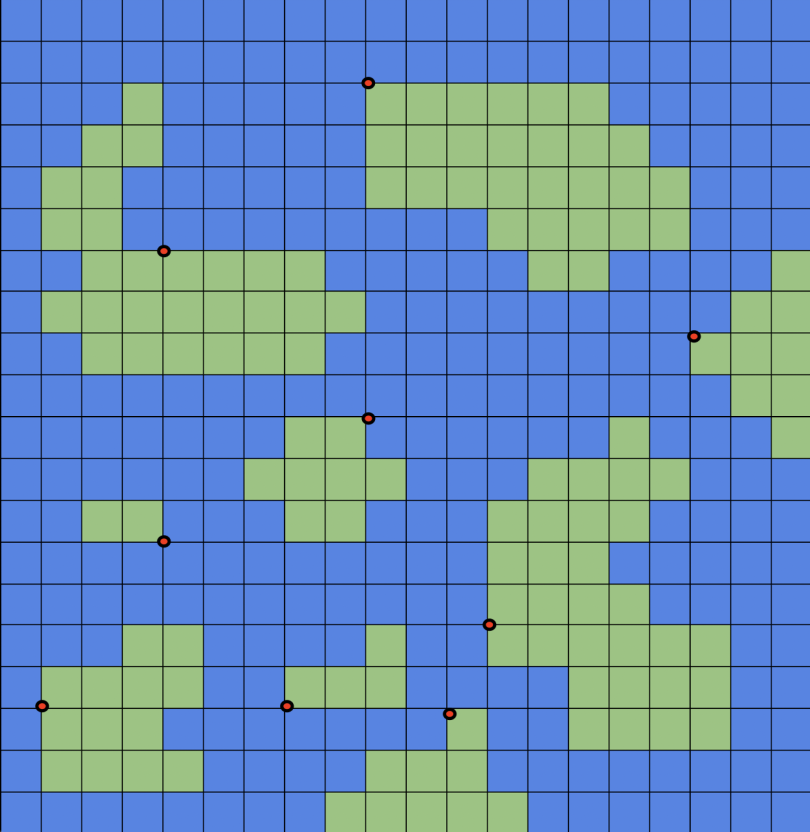
* Land(x,y)
  + True if the xy location is land
* Water(x,y)
  + True if the xy location is water
* Ship(time, x, y)
  + True if the xy location has a ship on it at given time with a specified cargo type
  + time can be any value between 0 and the end time
* Cargo(time, type)
  + Type can be Produce, Appliances, Cars
  + Time can be any value between 0 and end time
  + 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(time, x, y, has\_cargo\_type, wants\_cargo\_type)
  + True if the xy location has a port and the ports wants and has are specified by specific cargo types at a given time
  + time can be any value between 0 and the end time
  + has\_cargo\_type, and wants\_cargo\_type: 3 Possible types.
  + An empty cargo type will be assigned to both a port’s “want” and “has” values once they are both satisfied.

**Constraints:**

* If the ship is at a time less than the end time then there must be a ship at the next time step at an adjacent tile (pretend the disjunction is xor)
  + (Ship(n,x,y) ⋀ ¬Ship(end,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))
* Any port at any time step can either have it’s goods or be finished
  + (Port(time,x,y, has\_cargo\_type, wants\_cargo\_type) ⋁ Port(time,x,y, 0, 0)) ⋀¬(Port(time,x,y, has\_cargo\_type, wants\_cargo\_type) ⋀ Port(time,x,y, 0, 0))
* Port at last time step must be finished
  + Port(end,x,y, 0, 0)
* Port at time 0 must have it’s goods
  + Port(0,x,y, has\_cargo\_type, wants\_cargo\_type)
* If the ships location is the same as a ports location then there must be a ship on the generated path at that location
  + (Port(end,x,y) → Ship(end,x,y) ⋁ Ship(end - 1,x,y) ⋁ … ⋁ Ship(start + 2, x, y) ⋁ Ship(start + 1, x, y) ⋁ Ship(start, x, y))
* If a finished port is true then a finished port at the next time step must also be true
  + Port(time,x,y, 0, 0)→Port(time+1,x,y, 0, 0)
* A finished port implies that the port is also finished at the previous time step or there is a ship currently at the port
  + Port(time,x,y, 0, 0) → (Port(time-1,x,y,0,0) ⋁ Ship(time,x,y))
* An unfinished port implies the next port is also unfinished or there is a ship at the next port
  + Port(time,x,y, has\_cargo\_type, wants\_cargo\_type) → (Port(time+1,x,y,has\_cargo\_type,wants\_cargo\_type) ⋁ Ship(time,x,y))
* 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 “want” and “has” values will then be set to 0.
  + 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)
  + (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)
* There can only be one ship at any given timestep
  + (Ship(time,x,y) ⋁(Ship(time,p,q))⋀ ¬(Ship(time,x,y) ⋀(Ship(time,p,q))
* There must be exactly one of the three cargo types carried by the ship at any given timestep
  + (Cargo(time, produce)⋁(Cargo(time, cars) ⋁Cargo(time, appliances))) ⋀ ¬ (Cargo(time, produce)⋀(Cargo(time, cars) ⋀Cargo(time, appliances)))

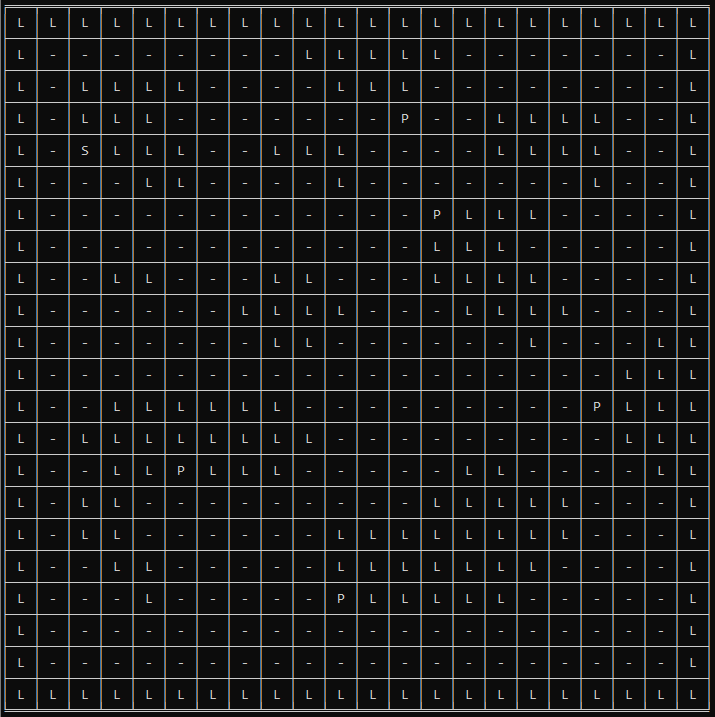
**Model Exploration**

Crucial to our model is the map that the program will be interacting with. Deciding how to do the map was tough, but we eventually decided that having hardcoded landmasses that the program interacts with would be best. Randomly generating a map, we decided, added too much complexity and constraints to our already complicated model. The original idea was also to have nine ports. Here is a very early mockup of what we originally thought the map might 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.

Originally, we thought that the model would work best on a very large scale map, but as we implemented our code, we quickly realized that the larger the map, the more exponentially complex our code was going to get, as the more tiles we have, the more ship propositions the code is going to create to solve the model, to the point that the time it would take for the model to solve would be extremely long. Here’s an example of what one of our maps looked like originally:



Of course, it quickly became apparent that use of a map this large was simply unfeasible for the code to find a good path through in a decent amount of time, so we had to downsize to much smaller variants of the same format of map, as shown in our final scenarios introduced at the beginning of this document.

Our ideas with the actual format for how the model would be solved changed a lot throughout the process of creating the project: Originally, we wanted the model to find the optimal route, and the optimal time it would take to complete that route, but that proved extremely difficult to try to model, so we moved on to the idea of providing the model with a time to solve within, and seeing if a proper solution could be found within that time.

**First Order Extension:**

Propositions:

* S(x) : x is a ship
* P(x) : x is a port
* W(x) : x is water
* L(x): x is the land
* T(x): x is the current time
* H(x): x is a cargo type
* C(x): x is a completed scenario

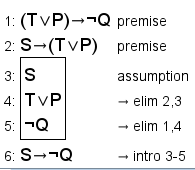
Constraints:

* If the ship is in the water or at a port it is not on land
  + ∃x∀y(((S(x)∧W(y))∨(S(x)∧P(y))∨(¬(S(x)∧W(y))∨¬(S(x)∧P(y))))→(¬L(y))
* When all ports are fulfilled the shipping route has been completed
  + ∀x∃y∃z(((P(x)∧T(y))→C(z))
* There must be exactly one of the three cargo types carried by the ship at any given timestep
  + ∀x∃y∃z∃i((S(x)∧H(y))→¬(H(z)∧H(i)))
* At any given timestep, there can only be one ship.
  + ∀x∀y∀z((S(x)⋁S(y))∧T(z))∧¬((S(x)∧S(y))∧T(z))

**Jape Proofs:**

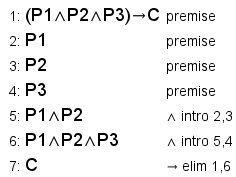
1. This proof is based on our ship only being able to be in the water- basically it’s saying that since we know that water or port implies that there’s no land, and we also know that ship implies only water or port, we can conclude that ship implies no land.

S = Ship, T = Water, P = Port, Q = Land



1. This proof is based on the final solution, in which if all ports are fulfilled then the shipping route has been completed. This is assuming a scenario in which there are three ports on the map.

P1 = Port1, P2 = Port2, P3 = Port3, C = Completed route



1. This proof is based on cargo types- it’s saying that if a port has one cargo type, then it cannot have either of the other types of cargo.

C1 = Cars, C2 = Produce, C3 = Appliances, P = Port

