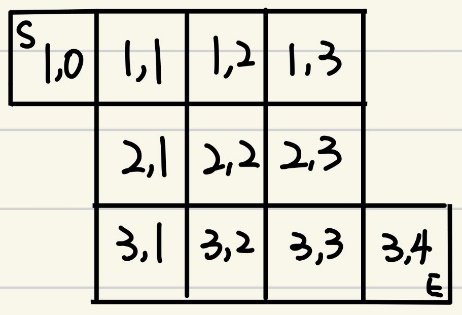
# Project Summary

*This is a game about connecting pipes in a 3x3 grid, where the start is in the top left corner and the end is in the bottom right corner. The goal of this game is to connect the starting pipe, the ending pipe, and every pipe in between them (there are three types) by rotating the pipes on the grid. We made a simple diagram to get a better idea of ​​what the entire grid looks like. We will simulate how pipes are placed, and how they are connected in the end.*

# *A screenshot of rico's chiliwork(pipe puzzle)*Propositions

*TODO*

1. Check if we need more propositions
2. Refinement of Variable

Draft:

* pipe\_type (): In addition to the start and end points, which have only one opening, there are three other styles, as shown above, Straight line, corner type, and T shape. If we recognize that it's a two-opening type with WE or NS in one location, we randomly choose from option 1. If we recognize that it's a three-opening type, we randomly choose from option 2. Otherwise, we choose from option 3.
* Location (pipe\_type, loc): pipe is at location. The location is where the board that things exist.
* Orientation (): We have four directions on each of the grids, which are east, west north and south, represented by “NSEW”.
* Neighbor (loc): Indicates neighboring grids, e.g. [11, 12]
* Neighbor\_updown(Neighbor, pipe\_type): For up and down adjacent grids, there exist pipes that can be connected (with corresponding opening directions)
* Neighbor\_leftright(Neighbor, pipe\_type): For left and right adjacent grids, there exist pipes that can be connected (with corresponding opening directions)
* Grid\_position\_connected(Neighbor, PIPE\_TYPE): one position has south and the other has north (two cells connected)
* Solver runs one configuration

Solver (loc, PIPE\_TYPE, Orientation): A particular pipe is oriented in a particular way. When you run the solver, it’s going to say your configuration and the directions.

# Constraints

*TODO*

1. Check if we need more constraints.
2. Complete the corresponding jape proof.

Draft:

* In a grid, only one pipe with exactly one config can exist.

¬ location(pipe\_type1, loc) \/ ¬ location(pipe\_type2, loc)

* Two faces are connected if they are each adjacent and have lines facing each other.

Neighbor ()Connect ()

* We need at least one solution to exist, and we can stop when checking for a feasible route.
* Win condition: A transitive connection from the starting pipe to the ending pipe

Connected(start)Connected1Connected2 … Connected(end)

* There is only one possible pipe type for the start(E) and one for the end point(W).
* The pipe type of the start and end points can only be on grids “01” and “34” respectively.
* Pipe\_connect: after we randomize the setup, check to see if each grid is connected.
* There is a limit to the pipe next to the starting and ending pipe. They can't be in a situation where can't connect.

# Model Exploration

For one of our constraint pipe\_type, we write a nested for loop to find every possible figure a pipe can have (like [‘N’,’ W’], but the elements in there should not be the same). Nested for loop for i, for j, for k when running giving back something like [‘N’,’ W’,’ W’] which is not the expectation. Then we realized something was wrong in the j and k loops since they are repeated ones. After correcting the staring value in loops from (0,i+1,i+2) to (0,i+1,j+1). The previous nested loop goes over the second part of the array twice which causes j and k loops to form the same element.

We have this constraint connected which generates with for loop what kind of a pair of pipe\_type can be connected like (pipe with opening to E and opening EW). We found it returned [[‘E’] [‘W’]] which means that the start and end are connected which is not possible. Since that is the only case that is possible in the whole generated array, we just delete that from the array

# Jape Proof Ideas

1. When there are left and right connected grids, it may not be necessary to look at the up and down of the left grid to see whether they are connected or not.
2. No need to consider optimal solutions.

# Requested Feedback

1. We have no idea how to start on Jape, can you tell us how it should look on Jape code?
2. Do the propositions and constraints make sense? (like can you picture how the game goes and win)

# First-Order Extension

*Describe how you might extend your model to a predicate logic setting, including how both the propositions and constraints would be updated.* ***There is no need to implement this extension!***

# Useful Notation

*Feel free to copy/paste the symbols here and remove this section before submitting.*