# Group 6: Miuki, Ming, and Joshua

# Project Summary

A grid of numbers on a piece of paper

Description automatically generatedThis game is 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. This game aims to connect the starting pipe and the ending pipe together by rotating the pipes on the grid (there are 5 types of pipes). After connecting, the water will flow through them and the water can only flow forward and downward. We made a simple diagram to get a better idea of ​​what the entire grid looks like. We will constraint the pipe orientation for each type of pipe on the gird based on their type and location. And the solution will arise from this process. Though it is possible that some grid won’t have a solution depending on the pipes’ location and type.

# Propositions

* Location (p, loc): There is a pipe with a certain pipe orientation at location loc. There are 11 locations available for a 3x3 grid and 3 pipes types with a total of 12 pipe orientations.
* NeighborLR (l1, l2): location l1 and l2 are neighbours and they are beside each other
* NeighborUD (l1, l2): location l1 and l2 are neighbours and one is on top of the other
* Connected (l1, l2): location l1 and l2 is connected
* Have\_to\_east (loc): the pipe on loc has an opening facing east
* Have\_to\_south (loc): the pipe on loc has an opening facing south
* Have\_from\_west (loc): the pipe on loc has an opening facing west
* Have\_from\_north (loc): the pipe on loc has an opening facing north
* Flow (l1, l2): water can flow from l1 to l2
* Straight\_Pipe (p, loc): the pipe is straight pipe at lozcation loc
* Angled\_Pipe (p, loc): the pipe is angled pipe at location loc
* Three\_Opening\_Pipe (p, loc): the pipe is 3-opening pipe at location loc
* Solution (l1, l2): location l1 to l2 have a solution

# Constraints

One location:

* There is only one pipe orientation of one pipe at location 10 and location 34(the start and end piece)

the opening of the start piece at location 10 can only face east

the opening of the end piece at location 34 can only face west

* All other locations (locations that are not 10 and 34) will not have opening only facing east or only facing west
* If there is a certain pipe at a location, then that location can have a different pipe orientation at the same location but not a different pipe type

Straight\_Pipe (pipe orientation, 11)>>add\_exactly\_one (Location (EW,11), Location (NS,11))

(If there is a straight pipe (orientated in EW or NS) at location 11, then location 11 can have exactly one pipe orientation from (EW, NS). But the other 2 pipes in a total of 8 pipe orientations can not be at location 11)

* For all locations, there is one pipe orientation from one pipe type depending on their location and all other orientation will not be allowed in this grid cell.

Ex. Three\_Opening\_Pipe (p, 11)>> Location(['S','E','W'], 11) or

Three\_Opening\_Pipe (p, 11)>> (~Location (['N', 'S', 'E'], 11) & ~Location (['N', 'S', 'W'], 11) & ~Location (['N', 'E', 'W'],11))

* If a pipe is orientated in a way at a location, then that location will not have the openings that the pipe orientation does not have

(ex. (Location (p, 11) & Straight\_Pipe (NS, 11))>> (~Have\_to\_east (11) **&**~Have\_from\_west (11)))

* If location loc has no opening toward east, then location loc can’t be connected to the grid cell on its right

~ Have\_to\_east (loc)>>~Connected (loc, loc+1)

* If location loc has no opening toward south, then location loc can’t be connected to the grid cell below it

~ Have\_to\_south(loc)>> ~Connected (loc, loc+10)

(Same constraint for opening west and north ^)

Two locations:

* If locations differ by exactly 1 or 10, they will be neighbors

(ex. 10 and 11 will be NeighborLR, and 11 and 12 will be NeighborUD)

* Every location that does not differ in 1 or 10 will not be neighbors

(ex. 10 and 34 is not NeighborLR or NeighborUD)

* If two locations are not neighbors, they are not connected and will not be able to flow from one another (ex. 10 and 34 is not connected)

! neighbor>> ! connected

* No connection upward or leftward.

(Connected (l1, l2) **~** Connected (l2, l1) where l2>l1)

Whole grid

-there is a solution if there is a connection from start to end

(Connected (10, 11) & Connected (11, 12) & … & Connected (33, 34)) | … | (Connected (10, 11) & Connected (11, 21) & … & Connected (33, 34))>>Solution (10, 34)

# Model Exploration

* For the model exploration, we first tried to set first row all to straight pipe with the function no\_sol\_with\_row\_strai (). According to our model, this should have no solution

A screenshot of a computer

Description automatically generated

A picture of the no\_sol\_with\_row\_strai in code

A black background with white text

Description automatically generated

Output when execute no\_sol\_with\_row\_strai ()

This setup does not have a solution because even though each element in each row have connection between them, there is no connection between rows which is needed for the constraint for the whole grid.

* Remove the pipe on location 21

A screen shot of a computer

Description automatically generated A screenshot of a computer screen

Description automatically generated

On the left is a test case we used for our model and on the right, it shows that this test case has solutions and display all the connections this test case has. The one solution route for this one is: location 10 to 11 to 21 to 31 to 32 to 33 to 34.

For the model exploration, we removed a pipe from solution route. Since it only has one solution, and it must go through 21, then, according to our model, this should not have a solution.

A screen shot of a computer program

Description automatically generated A screen shot of a computer

Description automatically generated

Code for the function and output of executing the function

But instead of saying no solution for this grid, solver did give a solution that is not right (it has connection for grid cell 21). Since the condition for two grid cells to be connected is if they are neighbours and the pipes on them have opening towards each other, and 21 has no pipe on it means that there should be no connection to it and from it. We soon realized that we are missing a constraint and a proposition that checks if there are empty grid cells. Before this function, we assumed that all gird cells have a pipe on it.

A black screen with white text

Description automatically generated

Add constraint to check if any grid cell is empty

A screen shot of a computer code

Description automatically generated

Add constraint: check all location that is neighbour, if any location in a neighbour pair empty, there should be no connection to that cell

After adding the codes in the pictures, now executing the function empty\_grid\_cell () give us an output saying there are no solutions.

* disconnect\_at\_beginning()

We wanted to make sure that if the beginning pipe was not connected that there would be no solution because it violates the constraint we have put in for this game. This is a relatively simple model exploration and only needed to make sure that if the starting pipe and the pipe at position 1,1 does not connect, then there is no solution to be found.

A constraint is added to check if this is the case.

A black background with white text

Description automatically generated

The result is that there is no solution after testing it with a test case.

* all\_angled\_pipe()

This is a edge case where we check whether there is a solution if the entire grid is filled with angled pipes other than the starting and ending pipe at 10 and 34. There should be no solution because due to our constraint that pipes can only be connected from left to right and from up to down, there is no path in which all angled pipe can connect from start to finish. In order for the pipes to be connected, they have to move backwards and upwards, which is not possible due to our constraints.A screen shot of a computer program

Description automatically generated

This is the code in which the entire grid is filled with only angled pipes. After running this code, the result is that there is no solution, as expected.

# Jape Proof

1. A screenshot of a computer

   Description automatically generatedIf there is a straight pipe on grid ‘13’, then the route containing this grid is not feasible.

The following assumptions are placed into Jape to set things up.

P: Straight Pipe

P1: Vertical Pipe facing the north-south direction.

P2: Horizontal Pipe facing the east-west direction.

A3: It means location ‘13’

1. P→(P1∨P2): A straight pipe implies that it can have either an NS direction or an EW direction pipe.
2. P1∧A3: vertical pipe in grid ‘13’.

P2∧A3: horizontal pipe in grid ‘13’

1. ¬C: the grid ‘12’, ‘13’, and ‘23’ are not connected.
2. R: The result does not contain a route for ‘13’.
3. There is a "chain of connections" from i1 to i3, which implies that any two grid spaces along the way are connected.

The following assumptions are placed into Jape to set things up.

1. ∀x.∀y.(PF(x,y)→PC(x,y)) : If the water can flow from grid x to grid y, it means they are connected.
2. ∀x.∃y.∀z.(PF(x,z)→(PF(x,y)∧PF(y,z))) : If water can flow from grid x to grid z, it means x can flow from grid x to y, and then flow from grid y to z.
3. Set up the initial configuration: actual i1, actual i2, actual i3

A screenshot of a computer

Description automatically generatedWe want to prove: ∃x.∃y.PC(x,y) two grids on the route that are connected.

1. If the first grid is connected to the second grid. However, the second grid is not connected to the third grid. This route is not connected.

The following assumptions are placed into Jape to set things up.

1. ∀x.∀y. (PC(x,y)→PF(x,y)) : If grids x and y (neighbor up-down or neighbor left-right) are connected, water can flow from grid x to grid y.
2. ∀x.∀y.∀z.((PF(x,y)∧PF(y,z))→PF(x,z): If water can go through from grid x grid to y and from grid y to grid z, then water can flow from grid x to grid z.
3. ∀x.∀y.(PC(x,y)→PF(x,y)): if grids x and y are not connected, it means water can not flow from x to y.
4. Set up the initial configuration: actual i1, actual i2, actual i3, PC (i1, i2), PC (i2, i3)
5. ∀x.∀y.∀z.((PF(x,y)∧PF(y,z))→PF(x,z): If water cannot flow from y to z, even if it can flow from x to y, it cannot eventually flow from x to z.

A screenshot of a computer

Description automatically generatedwe want to prove: PF (i1, i3) Water can not go through in this route.

# First-Order Extension

To start with, we’ll define some of the predicates that we might have, following the propositions above:

* **Location (p, loc):** There is a pipe type with a certain pipe orientation at location loc.
* **Have\_to\_east (loc)**: the pipe on loc has an opening facing east
* **Have\_from\_west (loc)**: the pipe on loc has an opening facing west
* **Connected (l1, l2):** location l1 and l2 is connected.
* **Solution(r):** Route r (l1 to l2) have a solution.

Here are the new predicates that we want to use:

* **Route(r):** r is a route.
* **Start (r, s):** s is the starting location of route r
* **End (r, e):** e is the ending location of route r
* **Orientation (p, o):** Pipe p has orientation o.

- Every Pipe is Placed in Exactly One location:

One location can only have one pipe.

No two different pipes can be placed at the same location.

∀p1​. ∀p2​. ∀loc.((Location(p1​,loc)Location(p2​,loc))→(p1​=p2)​)

* Pipe rules after the setup

Any pipe at the location can have different pipe orientations.

∀p.∀loc. o1. o2.(location(p,loc)(Orientation(p,o1)Orientation(p,o2))

* Start and End Pipe Placement:

The starting pipe must be at the top-left corner facing East, and the ending pipe at the bottom-right corner facing West.

loc.(Start(r, s), Have\_to\_east (‘10’))Location(End(r, e), Have\_from\_west (‘34’))

* Route connected

For both grid and pipe, it cannot connect to the left and up grid, it can only go right or down. (It can avoid self-loop and will be easier to find the solution.)

Connected (l1, l2) **~** Connected (l2, l1)

* Condition of solution

For one of the routes, if there is a connection from start to end, there will be a solution.

∀r. (Route(r)∃s.∃e. (Start(r, s)End(r, e)Connected(s, e))→Solution(r)).