Group 10: Overflow

Stuart Fong (20sjkf)

Nathan Fong (20njhf)

Vincent Proulx (20vp24)

Wasiq Wadud (17ww41) (No longer in group)

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

December 3, 2022

Contents

1 Abstract

We model Overflow, a mini-game inside of the popular massively multiplayer online game Animal Jam. The goal is to create a continuous path from the ocean to the moat of a sandcastle. This is accomplished by rotating square tiles containing either a straight path, a curved path, or a bridge. We create a solver for the game where given the layout of a level, the solver finds a solution if one exists. In addition, it finds the longest possible path in order to maximize the number of points gained. We detail the propositions and constraints provided to an SAT solver, the intermediate steps to build the model, and how the solver can be further extended in a predicate logic setting.

2 Introduction

2.1 How to Play

When the user starts a level, they are greeted with a randomized layout of tiles. These tiles can contain paths used to transport water:

Straight tile: A straight path that goes up and down or left and right.

Curved tile: Bends the path of the water by 90 degrees in any direction.

Bridge tile: Allows water to flow straight in either or both directions.

(a) New level

(b) Completed level

Figure 1: A level in Overflow.

All other tiles are blank and are unable to transport water. In the first phase of the level, the player is able to freely rotate the path tiles. In preparation for the next phase, the user's goal is to create a continuous path from the ocean, a red gate with water on one side, to the moat, a dark-brown ditch that surrounds a sandcastle. Once the user clicks on the blue "Start Water" button at the bottom of the screen, the gate separating the ocean and the rest of the level opens, allowing water to travel along the path created by the player. During this phase, the user is unable to rotate any tiles. If at any point the water is unable to travel further due to the path ending, the level is failed and the user is forced to restart. Otherwise, if the water is successfully able to flow from the ocean to the moat, the level is completed and the player is rewarded with the in-game currency "gems" based on the length of their solution path. The number of gems rewarded is directly proportional to the length of the solution path.

2.2 Solver Objective

The goal of the solver is to find the longest solution path in a given level layout from the ocean to a moat tile. This can be broken down into two parts: finding all solution paths and calculating the length of each. When finding all solution paths, we restrict how each tile can link to the tiles around it. We then model the flow of water from one tile to the next through linking. If water is able to travel to the moat tile, the path is deemed a solution to the level. After all solutions are found, we count the number

of water tiles in each solved level and store the counts. Finally, we calculate the maximum of the counts and the solution with this count is returned as the solved level.

3 Propositions

For our initial model, each type of path is listed as a separate proposition. Let $r, c \in \mathbb{Z}$ and $r, c \geq 0$, where r is the r-th row and c is the c-th column of the level. Note that the origin is situated at the top left of the screen with coordinate $(0, 0)$.

$S(r, c)$: If there is a straight tile at coordinate (r, c).

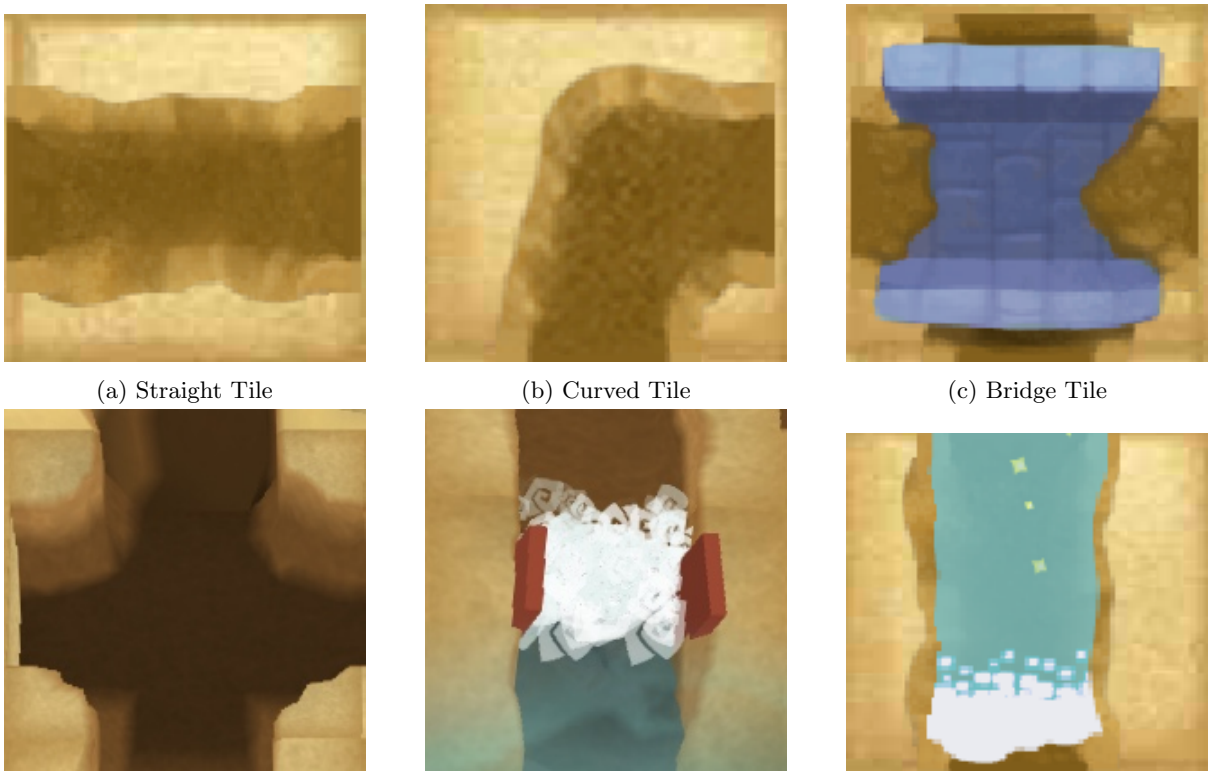
$C(r, c)$: If there is a curved tile at coordinate (r, c).

$B(r, c)$: If there is a bridge tile at coordinate (r, c).

In order to handle the position of the ocean and moat, we treat them as path tiles.

$M(r, c)$: If there is a moat tile at coordinate (r, c).

$O(r, c)$: If there is an ocean tile at coordinate (r, c).



(a) Straight Tile (b) Curved Tile (c) Bridge Tile

(d) Moat Tile (e) Ocean Tile (f) Water

Figure 2: Types of tiles present in a level.

Now that we have all the positions of the tiles, we need propositions representing the orientation of each tile. Initially we created four propositions to indicate if a tile links to another tile in a certain direction.

$LU(r, c)$: If the tile at coordinate (r, c) is linked to the tile at $(r - 1, c)$.

$LD(r, c)$: If the tile at coordinate (r, c) is linked to the tile at $(r + 1, c)$.

$LL(r, c)$: If the tile at coordinate (r, c) is linked to the tile at $(r, c - 1)$.

$LR(r, c)$: If the tile at coordinate (r, c) is linked to the tile at $(r, c + 1)$.

Note that linking is undefined at the borders of the level, so tiles at the edges and corners of the level do not have all four linking propositions.
To model the water in the second phase of the level, we create a proposition to indicate if a tile would contain water if the water is started.

$W(r, c)$: If the tile at coordinate (r, c) contains water.

Note the difference between the linking and water propositions: the water proposition indicates that the chosen tile is linked to other tiles, but does not specify the direction in which the tile is linked.
When all possible solution paths are found by the SAT solver, we want to find the longest path. To do this, we create a proposition which indicates the length of a path.

$N(r_j, c_k, i)$: If the length of the solution path counting in row-major order up to the tile at (r_j, c_k) has a 1 in its bit representation at the i-th position from the right.

For example in Figure 1b, we count $14_{10} = 001110_2$ water tiles up to and including coordinate (2, 2). This means that $N(2, 2, 1)$, $N(2, 2, 2)$, and $N(2, 2, 3)$ hold, while $N(2, 2, 0)$, $N(2, 2, 4)$, and $N(2, 2, 5)$ do not. Note that we only count one moat tile and that bridges are only counted once.

4 Constraints

Each of the tile types have their own rules on how they can be linked with other tiles. When creating the constraints, we restrict linking only when the tile contains water. This ensures that the path from the ocean to the moat is continuous as water travels from a tile to another tile:

- Straight tiles link tiles that are opposite from each other:

$$\begin{aligned} S(r,c) \wedge W(r,c) \rightarrow & (LU(r,c) \wedge LD(r,c) \wedge \neg LL(r,c) \wedge \neg LR(r,c)) \\ & \vee (\neg LU(r,c) \wedge \neg LD(r,c) \wedge LL(r,c) \wedge LR(r,c)) \end{aligned} \tag{4.1}$$

- Curved tiles link tiles at a 90 degree angle:

$$\begin{aligned} C(r,c) \wedge W(r,c) \rightarrow & (LU(r,c) \wedge \neg LD(r,c) \wedge LL(r,c) \wedge \neg LR(r,c)) \\ & \vee (LU(r,c) \wedge \neg LD(r,c) \wedge \neg LL(r,c) \wedge LR(r,c)) \\ & \vee (\neg LU(r,c) \wedge LD(r,c) \wedge LL(r,c) \wedge \neg LR(r,c)) \\ & \vee (\neg LU(r,c) \wedge LD(r,c) \wedge \neg LL(r,c) \wedge LR(r,c)) \end{aligned} \tag{4.2}$$

- Bridge tiles link opposite tiles or tiles from all four directions:

$$\begin{aligned} B(r,c) \wedge W(r,c) \rightarrow & (LU(r,c) \wedge LD(r,c) \wedge \neg LL(r,c) \wedge \neg LR(r,c)) \\ & \vee (\neg LU(r,c) \wedge \neg LD(r,c) \wedge LL(r,c) \wedge LR(r,c)) \\ & \vee (LU(r,c) \wedge LD(r,c) \wedge LL(r,c) \wedge LR(r,c)) \end{aligned} \tag{4.3}$$

For moat and ocean tiles, we restrict their linking so that they can only link to a single tile. This is to ensure that solutions start at the ocean tile and end at a moat tile:

$$M(r,c) \wedge W(r,c) \rightarrow \text{ exactly-one}(LU(r,c), LD(r,c), LL(r,c), LR(r,c)) \tag{4.4}$$

$$O(r,c) \rightarrow \text{ exactly-one}(LU(r,c), LD(r,c), LL(r,c), LR(r,c)) \tag{4.5}$$

In a level, at most one type of tile can be located at a given row and column. If none of the tile propositions are true at a certain position, then the tile is blank:

$$\text{at-most-one}(S(r,c), C(r,c), B(r,c), M(r,c), O(r,c)) \tag{4.6}$$

If a tile is blank, it cannot transport water:

$$\neg S(r,c) \wedge \neg C(r,c) \wedge \neg B(r,c) \wedge \neg M(r,c) \wedge \neg O(r,c) \rightarrow \neg W(r,c) \tag{4.7}$$

Each linking proposition in one direction has an equivalent linking proposition in the opposite direction:

$$LU(r,c) \leftrightarrow LD(r-1,c) \tag{4.8}$$
$$LL(r,c) \leftrightarrow LR(r,c-1) \tag{4.9}$$

The level is solved if a moat tile contains water (for a level with m rows and n columns):

$$(M(r_1,c_1) \wedge W(r_1,c_1)) \vee (M(r_2,c_2) \wedge W(r_2,c_2)) \vee \cdots \vee (M(r_m,c_n) \wedge W(r_m,c_n)) \tag{4.10}$$

where each pair (r_j, c_k) is the coordinate of a moat tile.
To compute the length of a solution path, we count the number of tiles containing water. Let (r_{j-1}, c_{k-1}) denote the coordinate of the tile that comes before the tile (r_j, c_k) when counting in row-major order. We create the following constraints:

$$W(r_j,c_k) \rightarrow (\neg P(r_{j-1},c_{k-1},i-1) \wedge \neg N(r_{j-1},c_{k-1},i) \rightarrow \neg N(r_j,c_k,i)) \tag{4.11}$$
$$W(r_j,c_k) \rightarrow (\neg P(r_{j-1},c_{k-1},i-1) \wedge N(r_{j-1},c_{k-1},i) \rightarrow N(r_j,c_k,i)) \tag{4.12}$$
$$W(r_j,c_k) \rightarrow (P(r_{j-1},c_{k-1},i-1) \wedge \neg N(r_{j-1},c_{k-1},i) \rightarrow N(r_j,c_k,i)) \tag{4.13}$$
$$W(r_j,c_k) \rightarrow (P(r_{j-1},c_{k-1},i-1) \wedge N(r_{j-1},c_{k-1},i) \rightarrow \neg N(r_j,c_k,i)) \tag{4.14}$$

where $P(r,c,i) \leftrightarrow N(r_j,c_k,0) + N(r_j,c_k,1) + \cdots + N(r_j,c_k,i)$. The reasoning behind these constraints will be explained in detail later.

5 Visualization

To interpret the outputs of our solver, we created a visualization which can be viewed on our GitHub repository.

5.1 Input

The layout of the level can be inputed in the level.py file. The level layout is stored in the level_layout variable, which is a list of strings, where each string is a row in the level. Each tile in the level can be translated to a character as follows:

Tile	Symbol
Straight	-
Curved	L
Bridge	+
Moat	#
Ocean	U
Blank	(space)

Table 1: Symbols corresponding to each tile.

5.2 Output

Once the overflow.py file is run, the solution path is printed if it exists. Tiles are rotated to signify their linking direction, and a blue background behind a character means that the tile contains water. Optionally, the –verbose argument can be specified to print detailed processing information onto the screen.

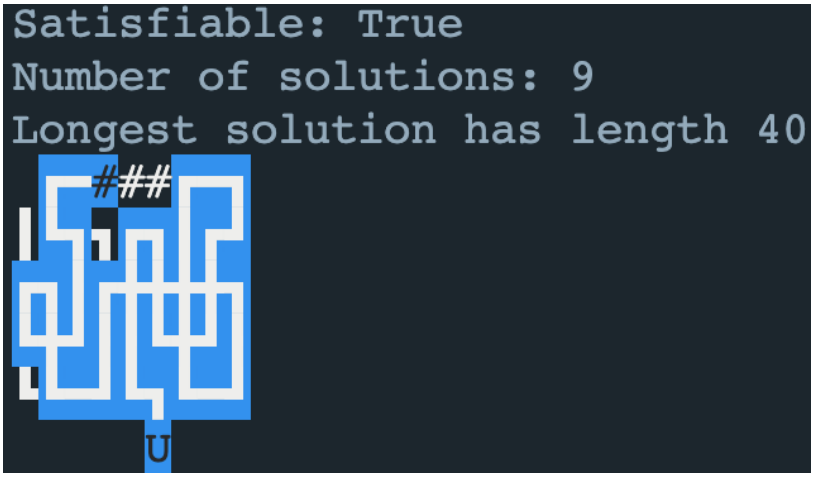


Figure 3: Example output from the visualization.

6 Model Exploration

6.1 Broken Bridges

Early on in our implementation, we constrained bridges to connect up and down, or left and right:

$$B(r,c) \land W(r,c) \to (LU(r,c) \land LD(r,c)) \lor (LL(r,c) \land LR(r,c)) \tag{6.1}$$

Here we intended to use the inclusive property of the disjunction to connect up and down, left and right, or in all four directions, but we did not restrict the path from going in three directions. This resulted in the bridges being used as a forked path, where a path would split into two paths, leading to two distinct moat tiles. To fix this, we replaced the previous constraint with a stricter truth-table-like constraint:

$$\begin{aligned} B(r,c) \land W(r,c) \to\ & (LU(r,c) \land LD(r,c) \land \neg LL(r,c) \land \neg LR(r,c)) \\ & \lor (\neg LU(r,c) \land \neg LD(r,c) \land LL(r,c) \land LR(r,c)) \\ & \lor (LU(r,c) \land LD(r,c) \land LL(r,c) \land LR(r,c)) \end{aligned} \tag{6.2}$$

6.2 The Self-Loop Problem

We noticed though a few test cases that self-loops could appear in our final result. This is because with finding the longest solution, self-loops filled with water incorrectly increase the length of the proposed path, making the solution more favourable. For example, consider a 2x2 level layout consisting only of curved tiles:

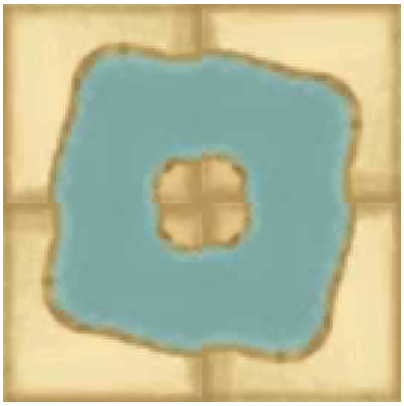


Figure 4: A theoretical self-loop.

We will show that using the current constraints, the tiles are able to arrange themselves to form a self-loop.

Theorem 1. $PC00, PC01, PC10, PC11, PW00, PW01, PW10, PW11,$
$PC00 \wedge PW00 \rightarrow PLD00 \wedge PLR00, PC01 \wedge PW01 \rightarrow PLD01 \wedge PLL01, PC10 \wedge PW10 \rightarrow PLU10 \wedge PLR10, PC11 \wedge PW11 \rightarrow PLU11 \wedge PLL11 \vdash PLD00 \wedge PLR00 \wedge PLD01 \wedge PLL01 \wedge PLU10 \wedge PLR10 \wedge PLU11 \wedge PLL11$

Proof. For all $r, c \in \{0, 1\}$, let $PC\langle r\rangle\langle c\rangle = C(r, c)$, $PW\langle r\rangle\langle c\rangle = W(r, c)$, $PLU\langle r\rangle\langle c\rangle = LU(r, c)$, $PLD\langle r\rangle\langle c\rangle = LD(r, c)$, $PLL\langle r\rangle\langle c\rangle = LL(r, c)$, $PLR\langle r\rangle\langle c\rangle = LR(r, c)$.

1:	PC00, PC01, PC10, PC11, PW00, PW01, PW10, PW11	premises
2:	PC00∧PW00→PLD00∧PLR00, PC01∧PW01→PLD01∧PLL01	premises
3:	PC10∧PW10→PLU10∧PLR10, PC11∧PW11→PLU11∧PLL11	premises
4:	PC11∧PW11	∧ intro 1.4,1.8
5:	PLU11∧PLL11	→ elim 3.2,4
6:	PLL11	∧ elim 5
7:	PLU11	∧ elim 5
8:	PC10∧PW10	∧ intro 1.3,1.7
9:	PLU10∧PLR10	→ elim 3.1,8
10:	PLR10	∧ elim 9
11:	PLU10	∧ elim 9
12:	PC01∧PW01	∧ intro 1.2,1.6
13:	PLD01∧PLL01	→ elim 2.2,12
14:	PLL01	∧ elim 13
15:	PLD01	∧ elim 13
16:	PC00∧PW00	∧ intro 1.1,1.5
17:	PLD00∧PLR00	→ elim 2.1,16
18:	PLD00∧PLR00∧PLD01	∧ intro 17,15
19:	PLD00∧PLR00∧PLD01∧PLL01	∧ intro 18,14
20:	PLD00∧PLR00∧PLD01∧PLL01∧PLU10	∧ intro 19,11
21:	PLD00∧PLR00∧PLD01∧PLL01∧PLU10∧PLR10	∧ intro 20,10
22:	PLD00∧PLR00∧PLD01∧PLL01∧PLU10∧PLR10∧PLU11	∧ intro 21,7
23:	PLD00∧PLR00∧PLD01∧PLL01∧PLU10∧PLR10∧PLU11∧PLL11	∧ intro 22,6

□

Self-loops like the one above make some shorter solutions longer, leading to a sub-optimal solution. Initially, self-loops were only seen with two moat tiles linking to each other. Naively diagnosing the problem, we specified that a moat tile cannot link with another moat tile.

$$\begin{aligned} M(r,c) \wedge W(r,c) \rightarrow & (LU(r,c) \wedge \neg LD(r,c) \wedge \neg LL(r,c) \wedge \neg LR(r,c) \wedge \neg M(r-1,c)) \\ & \vee (\neg LU(r,c) \wedge LD(r,c) \wedge \neg LL(r,c) \wedge \neg LR(r,c) \wedge \neg M(r+1,c)) \\ & \vee (\neg LU(r,c) \wedge \neg LD(r,c) \wedge LL(r,c) \wedge \neg LR(r,c) \wedge \neg M(r,c-1)) \\ & \vee (\neg LU(r,c) \wedge \neg LD(r,c) \wedge \neg LL(r,c) \wedge LR(r,c) \wedge \neg M(r,c+1)) \end{aligned} \tag{6.3}$$

After "fixing" this and running through a few more test cases, we were able to find a level layout where the "longest path" contained a path as well as a loop.

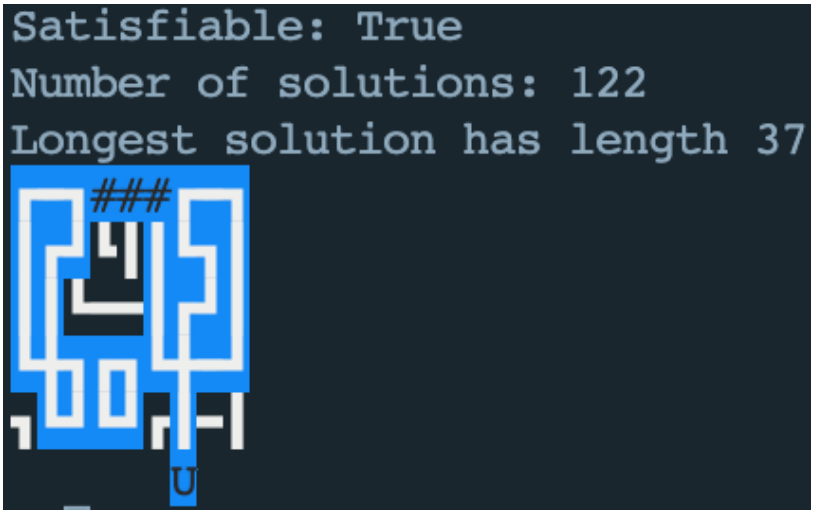


Figure 5: A solution containing self-loops.

We fixed this by instead finding an algorithm for detecting self-loops, where we run the SAT solver twice. During the first run, we remove all ocean tiles by treating them as blank tiles and temporarily remove the constraint that there must be a moat tile filled with water. Now when we find all possible models, each unique model consists of some combination of self-loops. On the second time we run the SAT solver, we add constraints based on the previous models that restrict the linking of tiles. In disallowing any combination of linking seen in the previous models, the tiles can no longer be linked in an orientation that produces a self-loop.

6.3 Too Many Solutions

When we were calculating the length of each solution path, we noticed that there were more solutions to the levels than we thought could occur. We found that this was due to the linking of the tiles that did not contain water, where we had no constraints for how they are linked. This caused there to be $2^4 = 16$ extra solutions for each tile that did not contain water. While this did not cause problems for small level layouts, the number of solutions quickly multiplied as the dimensions of the level increased. As a result, the time it took to find the length of all possible solution paths took much longer than necessary. We solved this by adding a constraint where a tile that does not contain water does not link in any direction.

$$\neg W(r,c) \rightarrow \neg LU(r,c) \wedge \neg LD(r,c) \wedge \neg LL(r,c) \wedge \neg LR(r,c) \tag{6.4}$$

This optimization places a minor assumption on the solution where we assume that the linking of the tiles that do not contain water does not have to be correct. That is, if a tile links to another tile but does not contain water, the program outputs that the tiles do not link at all. We remark that the linking of the tiles can be added afterwards with the tiles not containing water facing in any orientation.

6.4 Calculating Path Length

When calculating the length of a path, we initially thought to handle every possible subset of tiles containing water as individual cases. Specifically, we created mn length propositions where m is the number of rows and n is the number of columns in a level. We defined the i-th proposition to hold if the solution path has length i. For each subset of tiles in a level, we take the conjunction of all the tiles in the subset and the negation of all the tiles that are not in the subset, and set it to imply the j-th length proposition where j is the cardinality of the subset.

```
# Iterate over every possible subset of the tiles.
# If a bit is on, there is water in tile j. Else, it does not contain water.
for i in range(1, 2 ** (n_row * n_col)):
    # Give temp an initial value based on the top left tile
    if i & 1 == 1:
        temp = waterf[0][0]
        n_ones = 1
    else:
        temp = ~waterf[0][0]
        n_ones = 0

    for j in range(1, n_row * n_col):
        if i & (1 << j) > 0:
            # Tile must contain water
            temp = temp & waterf[j // n_col][j % n_col]
            n_ones += 1
        else:
            # Tile must not contain water
            temp = temp & ~waterf[j // n_col][j % n_col]
    # If a solution has this layout, the path is length equal to the number of 1 bits in i
    F.add_constraint(temp >> length[n_ones])
```

Figure 6: Initial code used to find the length of a solution path.

While this worked for small level layouts, the process of finding a solution using the SAT solver was extremely inefficient as the SAT solver would have more than 2^{mn} constraints as input. To put this into perspective, we estimated that the level in Figure 1 when used as input into the program would take more than one year to return a solution.
After some time, we found another way to approach the problem, where we "count" the length from left to right and top to bottom. We note that the length of the solution path up to a certain tile is the length at the previous tile, plus one if the current tile contains water. While this process would be straight forward using bit-blasting, we realized that we would have mn propositions for each tile, leading to a total of $(mn)^2$ propositions. To make this more efficient, we represented the length in binary, where for each tile in the level, we have a proposition for every bit $N(r_j, c_k, i)$. In order to add one in this scenario, we borrow the concept of "propagating a carry" from computer architecture.

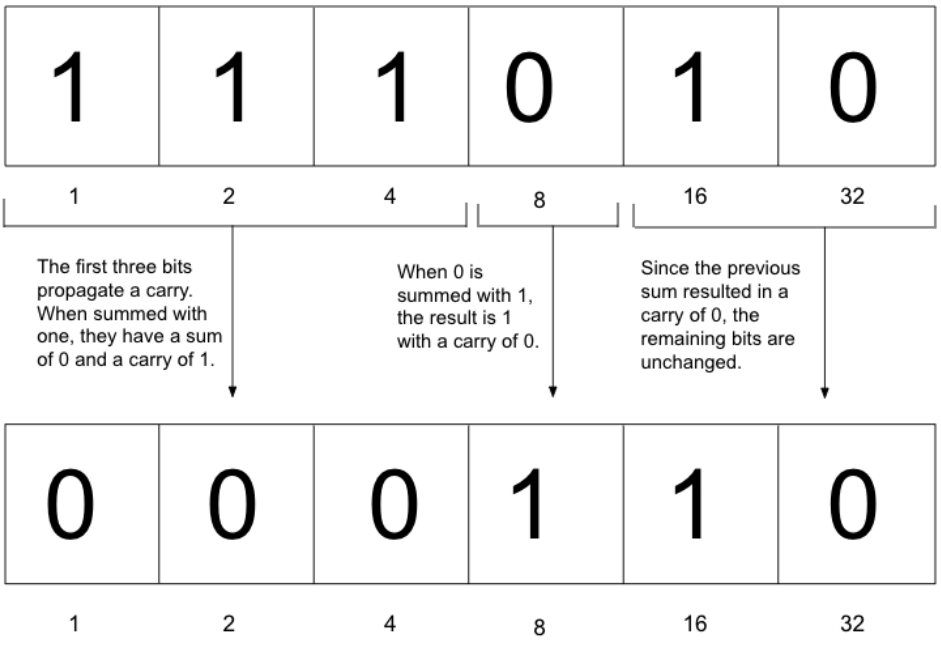


Figure 7: Diagram showing the operation of adding one to a binary string. (Binary written backwards)

For a given tile, if all previous length propositions hold and the current length proposition does not

hold, then in the next tile the previous length propositions would all not hold and the current length proposition would hold. The length propositions after that would not change. With this in mind, we add the following constraints:

$$W(r_j, c_k) \rightarrow (\neg P(r_{j-1}, c_{k-1}, i-1) \wedge \neg N(r_{j-1}, c_{k-1}, i) \rightarrow \neg N(r_j, c_k, i)) \tag{6.5}$$
$$W(r_j, c_k) \rightarrow (\neg P(r_{j-1}, c_{k-1}, i-1) \wedge N(r_{j-1}, c_{k-1}, i) \rightarrow N(r_j, c_k, i)) \tag{6.6}$$
$$W(r_j, c_k) \rightarrow (P(r_{j-1}, c_{k-1}, i-1) \wedge \neg N(r_{j-1}, c_{k-1}, i) \rightarrow N(r_j, c_k, i)) \tag{6.7}$$
$$W(r_j, c_k) \rightarrow (P(r_{j-1}, c_{k-1}, i-1) \wedge N(r_{j-1}, c_{k-1}, i) \rightarrow \neg N(r_j, c_k, i)) \tag{6.8}$$

where $P(r_j, c_k, i) \leftrightarrow N(r_j, c_k, 0) + N(r_j, c_k, 1) + \cdots + N(r_j, c_k, i)$, representing if the 0 to i-th bits at row r_j and column c_k can propagate a carry. Note that the tile at $(0, 0)$ is an edge case and is handled separately. This results in $4(mn)$ constraints to compute the length of a path as an alternative to the previous 2^{mn} constraints, greatly decreasing the time needed to find a solution.

6.5 Multiple Oceans

So far in the model, we were only using one ocean tile as per the rules of Overflow. Despite this, we wanted to explore how our model would generalize in the case that we had multiple oceans in a single level. Through some testing, we found that in a solution, all oceans transport water to distinct moats. That is, no path ends abruptly, and no two oceans have paths that link to the same moat. This is because for every path tile, if it contains water then it must transport the water to another tile. We also have that moats are only able to link in one direction. The only problem we encountered was that ocean tiles could link to other ocean tiles. This was similar to what we faced before, where moat tiles could link to other moat tiles. The difference between ocean tiles and moat tiles is that ocean tiles always contain water, so we were not able to isolate subsets containing only a single self-loop. To solve this, we replaced the ocean constraint with an equivalent constraint:

$$W(r, c) \wedge (W(r, c) \wedge O(r, c) \rightarrow \text{ exactly-one}(LU(r, c), LD(r, c), LL(r, c), LR(r, c))) \tag{6.9}$$

We show their equivalence in Section 7.2.
With this in mind, we replaced our ocean constraint and were able to remove the requirement that ocean tiles always contain water. This allowed us to use our self-loop algorithm from before as we were able to make ocean tiles temporarily have the same properties as moat tiles. With our solver being able to generalize to levels with multiple ocean tiles, we have shown how our methods are able to be extended to other games such as Numberlink.

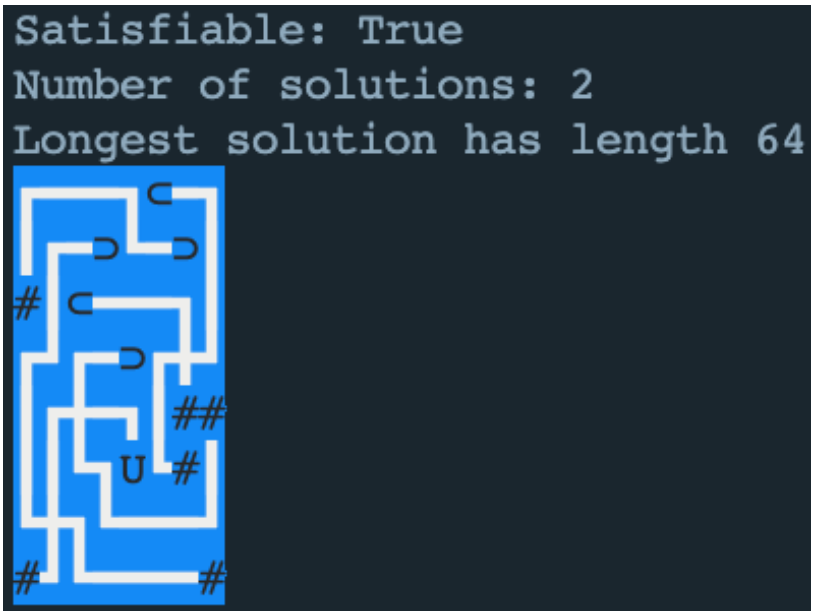


Figure 8: A solved level with multiple oceans.

6.6 Duplicate Linking

If we recall the constraints for equivalent linking propositions in Equation (4.8), we notice that $LL(r, c)$ can be thought of as an auxiliary variable for $LD(r-1, c)$, $LL(r, c)$ can be thought of as an auxiliary variable for $LR(r, c-1)$, and vice versa. When we discovered this, we wanted to try to see if we could eliminate two of the linking propositions. In the end we created two propositions $LH(r, c)$ and $LV(r, c)$ to link tiles horizontally and vertically respectively.

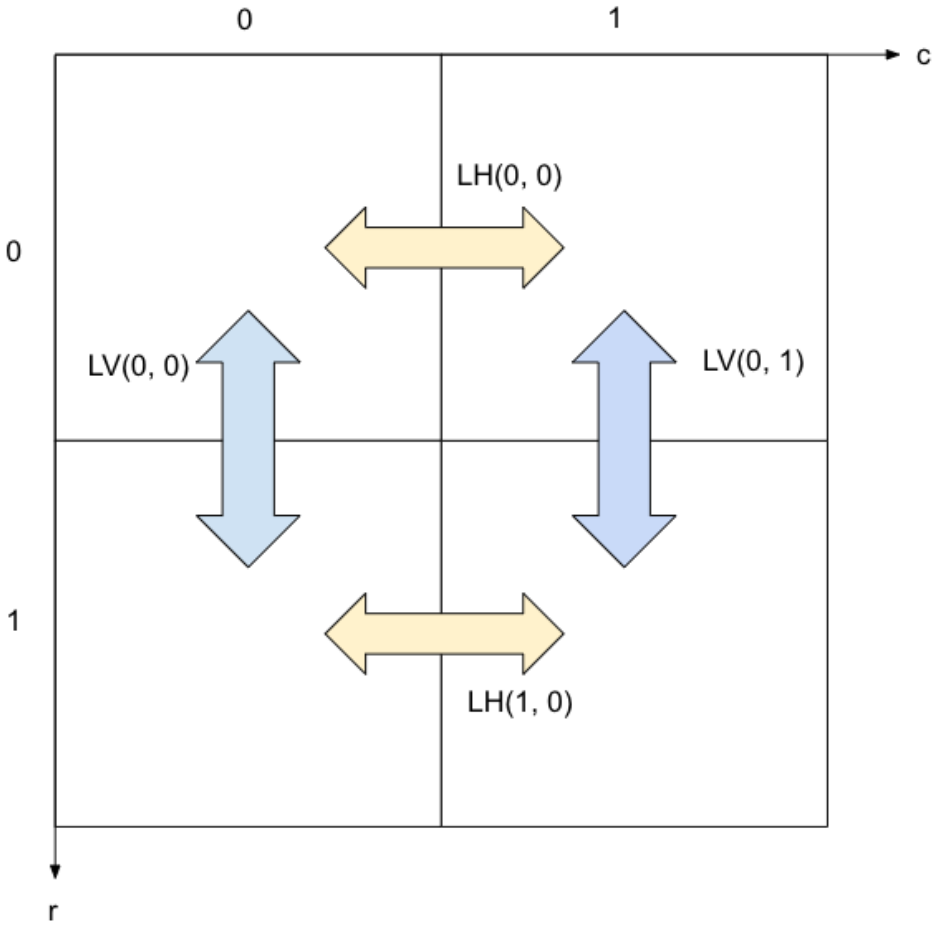


Figure 9: Diagram showing the linking directions of the new propositions.

These propositions replaced the existing four linking propositions as follows:

$$LV(r, c) \leftrightarrow LD(r, c) \leftrightarrow LU(r+1, c) \tag{6.10}$$
$$LH(r, c) \leftrightarrow LR(r, c) \leftrightarrow LL(r, c+1) \tag{6.11}$$

With this change, we halved the number of propositions needed to represent the linking of tiles.

6.7 Removing Water

As stated previously, the difference between the water and linking propositions is that linking specifies the direction in which paths connect, while water does not. To explore their properties further, we wanted to see if we could remove the water proposition entirely when finding a solution path. To do this, we further investigated the relationship between water and linking.
The proof can be found in Section 7.3.
So $W(r, c) \leftrightarrow LU(r, c) \vee LD(r, c) \vee LL(r, c) \vee LR(r, c)$, meaning that $W(r, c)$ is an auxiliary variable for whether a tile links to another tile or not. With this in mind, we were able to replace all instances of water propositions with linking propositions. This allowed us to remove the need for water and instead rely on the linking of tiles to determine a solution path. This relates to the first phase of the level when

the user is still constructing their path. Although the water is not yet started, the user is still able to make out which tiles will contain water based on the linking of the path.

7 Jape Proofs

7.1 Special Constraints on Edge Tiles

For levels with m rows and n columns, the "Edge Tiles" are the tiles that have one of the following r or c value: $r = 0$ or $r = m - 1$ or $c = 0$ or $c = n - 1$ ("Corner Tiles" are a type of edge tile that have both one of these r and one of these c values). Thus, edge tiles are tiles that are on the top, bottom, leftmost, or rightmost sides of the level map. Below is a theorem and a Jape proof that straight tiles that are situated on the top, bottom, leftmost, or rightmost sides of the level map have extra conditions that prevent the tile from linking in the direction of the side they are situated against. While coding the implementation of the project, we had to keep track of these extra constraints to make sure the tiles did not link out of bounds. To get started on the proof, first we must recall the original constraints on straight tiles and we must notice that tiles placed against the top wall of the level map cannot link upwards, since the index of the tile it would link to would be out of bounds. Similar logic applies to other types of tiles placed against these walls, with the only difference being that they each have their own distinct extra constraints (Except for bridge tiles; since they behave similarly to straight tiles, they thus have the same extra constraints as straight tiles).

Here we will use $P\langle r\rangle\langle c\rangle$ to represent that "the tile has a row index of $\langle r\rangle$ and a column index of $\langle c\rangle$", $Q\langle d\rangle$ to represent "linking in the $\langle d\rangle$ direction", and $R\langle t\rangle$ to represent "the tile is a $\langle t\rangle$ type of tile".

Theorem 2. $(RS \rightarrow ((QU \wedge QD \wedge \neg QL \wedge \neg QR) \vee (\neg QU \wedge \neg QD \wedge QL \wedge QR) \vee (\neg QU \wedge \neg QD \wedge \neg QL \wedge \neg QR))), (P0C \rightarrow (\neg QU)), (PMC \rightarrow (\neg QD)), (PR0 \rightarrow (\neg QL)), (PRN \rightarrow (\neg QR)) \vdash ((P0C \wedge RS) \rightarrow ((\neg QD \wedge \neg QL \wedge \neg QR) \vee (\neg QD \wedge QL \wedge QR))) \wedge ((PMC \wedge RS) \rightarrow ((\neg QU \wedge \neg QL \wedge \neg QR) \vee (\neg QU \wedge QL \wedge QR))) \wedge ((PR0 \wedge RS) \rightarrow ((\neg QU \wedge \neg QD \wedge \neg QR) \vee (QU \wedge QD \wedge \neg QR))) \wedge ((PRN \wedge RS) \rightarrow ((\neg QU \wedge \neg QD \wedge \neg QL) \vee (QU \wedge QD \wedge \neg QL)))$

Proof. Let $P0C = \text{Coordinates}(0, c)$, $PMC = \text{Coordinates}(m-1, c)$, $PR0 = \text{Coordinates}(r, 0)$, $PRN = \text{Coordinates}(r, n-1)$, $QU = LU(r, c)$, $QD = LD(r, c)$, $QL = LL(r, c)$, $QR = LR(r, c)$, $RS = \text{Straight-Tile}$.

1:	(RS→((QU∧QD∧¬QL∧¬QR)∨(¬QU∧¬QD∧QL∧QR)∨(¬QU∧¬QD∧¬QL∧¬QR))) , (P0C→(¬QU))	premises
2:	(PMC→(¬QD)), (PR0→(¬QL)), (PRN→(¬QR))	premises
3:	P0C∧RS	assumption
4:	RS	∧ elim 3
5:	(QU∧QD∧¬QL∧¬QR)∨(¬QU∧¬QD∧QL∧QR)∨(¬QU∧¬QD∧¬QL∧¬QR)	→ elim 1.1,4
6:	P0C	∧ elim 3
7:	¬QU	→ elim 1.2,6
8:	(QU∧QD∧¬QL∧¬QR)∨(¬QU∧¬QD∧QL∧QR)	assumption
9:	QU∧QD∧¬QL∧¬QR	assumption
10:	QU∧QD∧¬QL	∧ elim 9
11:	QU∧QD	∧ elim 10
12:	QU	∧ elim 11
13:	⊥	¬ elim 12,7
14:	(¬QD∧¬QL∧¬QR)∨(¬QD∧QL∧QR)	contra (constructive) 13
15:	¬QU∧¬QD∧QL∧QR	assumption
16:	¬QU∧¬QD∧QL	∧ elim 15
17:	¬QU∧¬QD	∧ elim 16
18:	¬QD	∧ elim 17
19:	QL	∧ elim 16
20:	¬QD∧QL	∧ intro 18,19
21:	QR	∧ elim 15
22:	¬QD∧QL∧QR	∧ intro 20,21
23:	(¬QD∧¬QL∧¬QR)∨(¬QD∧QL∧QR)	∨ intro 22
24:	(¬QD∧¬QL∧¬QR)∨(¬QD∧QL∧QR)	∨ elim 8,9-14,15-23
25:	¬QU∧¬QD∧¬QL∧¬QR	assumption
26:	¬QU∧¬QD∧¬QL	∧ elim 25
27:	¬QU∧¬QD	∧ elim 26
28:	¬QD	∧ elim 27
29:	¬QL	∧ elim 26
30:	¬QD∧¬QL	∧ intro 28,29
31:	¬QR	∧ elim 25
32:	¬QD∧¬QL∧¬QR	∧ intro 30,31
33:	(¬QD∧¬QL∧¬QR)∨(¬QD∧QL∧QR)	∨ intro 32
34:	(¬QD∧¬QL∧¬QR)∨(¬QD∧QL∧QR)	∨ elim 5,8-24,25-33
35:	(P0C∧RS)→((¬QD∧¬QL∧¬QR)∨(¬QD∧QL∧QR))	→ intro 3-34
36:	PMC∧RS	assumption
37:	RS	∧ elim 36
38:	(QU∧QD∧¬QL∧¬QR)∨(¬QU∧¬QD∧QL∧QR)∨(¬QU∧¬QD∧¬QL∧¬QR)	→ elim 1.1,37
39:	PMC	∧ elim 36
40:	¬QD	→ elim 2.1,39
41:	(QU∧QD∧¬QL∧¬QR)∨(¬QU∧¬QD∧QL∧QR)	assumption
42:	QU∧QD∧¬QL∧¬QR	assumption
43:	QU∧QD∧¬QL	∧ elim 42
44:	QU∧QD	∧ elim 43
45:	QD	∧ elim 44
46:	⊥	¬ elim 45,40
47:	(¬QU∧¬QL∧¬QR)∨(¬QU∧QL∧QR)	contra (constructive) 46
48:	¬QU∧¬QD∧QL∧QR	assumption
49:	¬QU∧¬QD∧QL	∧ elim 48
50:	¬QU∧¬QD	∧ elim 49
51:	¬QU	∧ elim 50
52:	QL	∧ elim 49
53:	¬QU∧QL	∧ intro 51,52
54:	QR	∧ elim 48
55:	¬QU∧QL∧QR	∧ intro 53,54
56:	(¬QU∧¬QL∧¬QR)∨(¬QU∧QL∧QR)	∨ intro 55
57:	(¬QU∧¬QL∧¬QR)∨(¬QU∧QL∧QR)	∨ elim 41,42-47,48-56
58:	¬QU∧¬QD∧¬QL∧¬QR	assumption
59:	¬QR	∧ elim 58
60:	¬QU∧¬QD∧¬QL	∧ elim 58

61:	¬QL	∧ elim 60
62:	¬QU∧¬QD	∧ elim 60
63:	¬QU	∧ elim 62
64:	¬QU∧¬QL	∧ intro 63,61
65:	¬QU∧¬QL∧¬QR	∧ intro 64,59
66:	(¬QU∧¬QL∧¬QR)∨(¬QU∧QL∧QR)	∨ intro 65
67:	(¬QU∧¬QL∧¬QR)∨(¬QU∧QL∧QR)	∨ elim 38,41-57,58-66
68:	(PMC∧RS)→((¬QU∧¬QL∧¬QR)∨(¬QU∧QL∧QR))	→ intro 36-67
69:	((P0C∧RS)→((¬QD∧¬QL∧¬QR)∨(¬QD∧QL∧QR))) ∧((PMC∧RS)→((¬QU∧¬QL∧¬QR)∨(¬QU∧QL∧QR)))	∧ intro 35,68
70:	PR0∧RS	assumption
71:	RS	∧ elim 70
72:	(QU∧QD∧¬QL∧¬QR)∨(¬QU∧¬QD∧QL∧QR) ∨(¬QU∧¬QD∧¬QL∧¬QR)	→ elim 1.1,71
73:	PR0	∧ elim 70
74:	¬QL	→ elim 2.2,73
75:	(QU∧QD∧¬QL∧¬QR)∨(¬QU∧¬QD∧QL∧QR)	assumption
76:	QU∧QD∧¬QL∧¬QR	assumption
77:	¬QR	∧ elim 76
78:	QU∧QD∧¬QL	∧ elim 76
79:	QU∧QD	∧ elim 78
80:	QU∧QD∧¬QR	∧ intro 79,77
81:	(¬QU∧¬QD∧¬QR)∨(QU∧QD∧¬QR)	∨ intro 80
82:	¬QU∧¬QD∧QL∧QR	assumption
83:	¬QU∧¬QD∧QL	∧ elim 82
84:	QL	∧ elim 83
85:	⊥	¬ elim 84,74
86:	(¬QU∧¬QD∧¬QR)∨(QU∧QD∧¬QR)	contra (constructive) 85
87:	(¬QU∧¬QD∧¬QR)∨(QU∧QD∧¬QR)	∨ elim 75,76-81,82-86
88:	¬QU∧¬QD∧¬QL∧¬QR	assumption
89:	¬QR	∧ elim 88
90:	¬QU∧¬QD∧¬QL	∧ elim 88
91:	¬QU∧¬QD	∧ elim 90
92:	¬QU∧¬QD∧¬QR	∧ intro 91,89
93:	(¬QU∧¬QD∧¬QR)∨(QU∧QD∧¬QR)	∨ intro 92
94:	(¬QU∧¬QD∧¬QR)∨(QU∧QD∧¬QR)	∨ elim 72,75-87,88-93
95:	(PR0∧RS)→((¬QU∧¬QD∧¬QR)∨(QU∧QD∧¬QR))	→ intro 70-94
96:	((P0C∧RS)→((¬QD∧¬QL∧¬QR)∨(¬QD∧QL∧QR))) ∧((PMC∧RS)→((¬QU∧¬QL∧¬QR)∨(¬QU∧QL∧QR))) ∧((PR0∧RS)→((¬QU∧¬QD∧¬QR)∨(QU∧QD∧¬QR)))	∧ intro 69,95
97:	PRN∧RS	assumption
98:	RS	∧ elim 97
99:	(QU∧QD∧¬QL∧¬QR)∨(¬QU∧¬QD∧QL∧QR) ∨(¬QU∧¬QD∧¬QL∧¬QR)	→ elim 1.1,98
100:	PRN	∧ elim 97
101:	¬QR	→ elim 2.3,100
102:	(QU∧QD∧¬QL∧¬QR)∨(¬QU∧¬QD∧QL∧QR)	assumption
103:	QU∧QD∧¬QL∧¬QR	assumption
104:	QU∧QD∧¬QL	∧ elim 103
105:	(¬QU∧¬QD∧¬QL)∨(QU∧QD∧¬QL)	∨ intro 104
106:	¬QU∧¬QD∧QL∧QR	assumption
107:	QR	∧ elim 106
108:	⊥	¬ elim 107,101
109:	(¬QU∧¬QD∧¬QL)∨(QU∧QD∧¬QL)	contra (constructive) 108
110:	(¬QU∧¬QD∧¬QL)∨(QU∧QD∧¬QL)	∨ elim 102,103-105,106-109
111:	¬QU∧¬QD∧¬QL∧¬QR	assumption
112:	¬QU∧¬QD∧¬QL	∧ elim 111
113:	(¬QU∧¬QD∧¬QL)∨(QU∧QD∧¬QL)	∨ intro 112
114:	(¬QU∧¬QD∧¬QL)∨(QU∧QD∧¬QL)	∨ elim 99,102-110,111-113
115:	(PRN∧RS)→((¬QU∧¬QD∧¬QL)∨(QU∧QD∧¬QL))	→ intro 97-114
116:	((P0C∧RS)→((¬QD∧¬QL∧¬QR)∨(¬QD∧QL∧QR))) ∧((PMC∧RS)→((¬QU∧¬QL∧¬QR)∨(¬QU∧QL∧QR))) ∧((PR0∧RS)→((¬QU∧¬QD∧¬QR)∨(QU∧QD∧¬QR))) ∧((PRN∧RS)→((¬QU∧¬QD∧¬QL)∨(QU∧QD∧¬QL)))	∧ intro 96,115

□

7.2 Multiple Oceans Proof

This is a continuation of our model exploration of multiple ocean tiles, that can be found in Section 6.5.

Theorem 3. $(W(r,c) \wedge (O(r,c) \rightarrow \ \textit{exactly-one}(LU(r,c), LD(r,c), LL(r,c), LR(r,c))))$
$\leftrightarrow (W(r,c) \wedge (W(r,c) \wedge O(r,c) \rightarrow \ \textit{exactly-one}(LU(r,c), LD(r,c), LL(r,c), LR(r,c))))$

Proof. Let $PW = W(r,c)$, $PO = O(r,c)$, and $PE = \text{exactly-one}(LU(r,c), LD(r,c), LL(r,c), LR(r,c))$.

1:	PW∧(PW∧PO→PE)	assumption
2:	PW∧PO→PE	∧ elim 1
3:	PW	∧ elim 1
4:	PO	assumption
5:	PW∧PO	∧ intro 3,4
6:	PE	→ elim 2,5
7:	PO→PE	→ intro 4–6
8:	PW∧(PO→PE)	∧ intro 3,7
9:	PW∧(PW∧PO→PE)→PW∧(PO→PE)	→ intro 1–8
10:	PW∧(PO→PE)	assumption
11:	PO→PE	∧ elim 10
12:	PW	∧ elim 10
13:	PW∧PO	assumption
14:	PO	∧ elim 13
15:	PE	→ elim 11,14
16:	PW∧PO→PE	→ intro 13–15
17:	PW∧(PW∧PO→PE)	∧ intro 12,16
18:	PW∧(PO→PE)→PW∧(PW∧PO→PE)	→ intro 10–17
19:	(PW∧(PW∧PO→PE)→PW∧(PO→PE))∧(PW∧(PO→PE)→PW∧(PW∧PO→PE))	∧ intro 9,18

□

7.3 Removing Water Proof

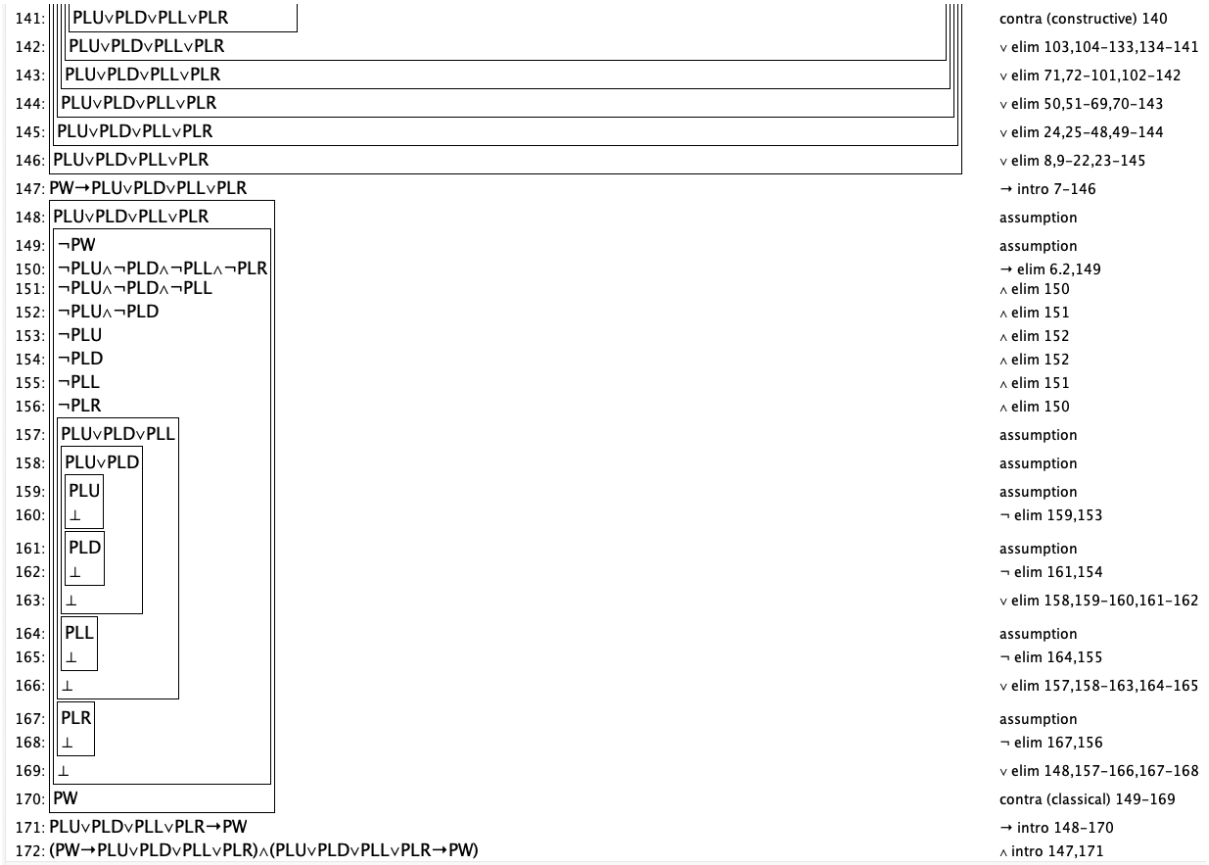
This is a continuation of Section 6.7. Here, we prove that $W(r,c) \leftrightarrow LU(r,c) \vee LD(r,c) \vee LL(r,c) \vee LR(r,c)$.

Theorem 4. $PS \wedge PW \rightarrow (PLU \wedge PLD \wedge \neg PLL \wedge \neg PLR) \vee (\neg PLU \wedge \neg PLD \wedge PLL \wedge PLR), PC \wedge PW \rightarrow (PLU \wedge \neg PLD \wedge PLL \wedge \neg PLR) \vee (PLU \wedge \neg PLD \wedge \neg PLL \wedge PLR) \vee (\neg PLU \wedge PLD \wedge PLL \wedge \neg PLR) \vee (\neg PLU \wedge PLD \wedge \neg PLL \wedge PLR), PB \wedge PW \rightarrow (PLU \wedge PLD \wedge \neg PLL \wedge \neg PLR) \vee (\neg PLU \wedge \neg PLD \wedge PLL \wedge PLR) \vee (PLU \wedge PLD \wedge PLL \wedge PLR), PM \wedge PW \rightarrow (PLU \wedge \neg PLD \wedge \neg PLL \wedge \neg PLR) \vee (\neg PLU \wedge PLD \wedge \neg PLL \wedge \neg PLR) \vee (\neg PLU \wedge \neg PLD \wedge PLL \wedge \neg PLR) \vee (\neg PLU \wedge \neg PLD \wedge \neg PLL \wedge PLR), PO \wedge PW \rightarrow (PLU \wedge \neg PLD \wedge \neg PLL \wedge \neg PLR) \vee (\neg PLU \wedge PLD \wedge \neg PLL \wedge \neg PLR) \vee (\neg PLU \wedge \neg PLD \wedge PLL \wedge \neg PLR) \vee (\neg PLU \wedge \neg PLD \wedge \neg PLL \wedge PLR), \neg PS \wedge \neg PC \wedge \neg PB \wedge \neg PM \wedge \neg PO \rightarrow \neg PW, \neg PW \rightarrow \neg PLU \wedge \neg PLD \wedge \neg PLL \wedge \neg PLR \vdash (PW \rightarrow PLU \vee PLD \vee PLL \vee PLR) \wedge (PLU \vee PLD \vee PLL \vee PLR \rightarrow PW)$

Proof. Let $PS = S(r,c)$, $PC = C(r,c)$, $PB = B(r,c)$, $PM = M(r,c)$, $PO = O(r,c)$, $PLU = LU(r,c)$, $PLD = LD(r,c)$, $PLL = LL(r,c)$, $PLD = LD(r,c)$, $PW = W(r,c)$.

1:	PS∧PW→(PLU∧PLD∧¬PLL∧¬PLR)∨(¬PLU∧¬PLD∧PLL∧PLR)	premise
2:	PC∧PW→(PLU∧¬PLD∧PLL∧¬PLR)∨(PLU∧¬PLD∧¬PLL∧PLR)∨(¬PLU∧PLD∧PLL∧¬PLR)∨(¬PLU∧PLD∧¬PLL∧PLR)	premise
3:	PB∧PW→(PLU∧PLD∧¬PLL∧¬PLR)∨(¬PLU∧¬PLD∧PLL∧PLR)∨(PLU∧PLD∧PLL∧PLR)	premise
4:	PM∧PW→(PLU∧¬PLD∧¬PLL∧¬PLR)∨(¬PLU∧PLD∧¬PLL∧¬PLR)∨(¬PLU∧¬PLD∧PLL∧¬PLR)∨(¬PLU∧¬PLD∧¬PLL∧PLR)	premise
5:	PO∧PW→(PLU∧¬PLD∧¬PLL∧¬PLR)∨(¬PLU∧PLD∧¬PLL∧¬PLR)∨(¬PLU∧¬PLD∧PLL∧¬PLR)∨(¬PLU∧¬PLD∧¬PLL∧PLR)	premise
6:	¬PS∧¬PC∧¬PB∧¬PM∧¬PO→¬PW, ¬PW→¬PLU∧¬PLD∧¬PLL∧¬PLR	premises
7:	PW	assumption
8:	PS∨¬PS	Theorem P∨¬P
9:	PS	assumption
10:	PS∧PW	∧ intro 9,7
11:	(PLU∧PLD∧¬PLL∧¬PLR)∨(¬PLU∧¬PLD∧PLL∧PLR)	→ elim 1,10
12:	PLU∧PLD∧¬PLL∧¬PLR	assumption
13:	PLU∧PLD∧¬PLL	∧ elim 12
14:	PLU∧PLD	∧ elim 13
15:	PLD	∧ elim 14
16:	PLU∨PLD	∨ intro 15
17:	PLU∨PLD∨PLL	∨ intro 16
18:	PLU∨PLD∨PLL∨PLR	∨ intro 17
19:	¬PLU∧¬PLD∧PLL∧PLR	assumption
20:	PLR	∧ elim 19
21:	PLU∨PLD∨PLL∨PLR	∨ intro 20
22:	PLU∨PLD∨PLL∨PLR	∨ elim 11,12–18,19–21
23:	¬PS	assumption
24:	PC∨¬PC	Theorem P∨¬P
25:	PC	assumption
26:	PC∧PW	∧ intro 25,7
27:	(PLU∧¬PLD∧PLL∧¬PLR)∨(PLU∧¬PLD∧¬PLL∧PLR)∨(¬PLU∧PLD∧PLL∧¬PLR)∨(¬PLU∧PLD∧¬PLL∧PLR)	→ elim 2,26
28:	(PLU∧¬PLD∧PLL∧¬PLR)∨(PLU∧¬PLD∧¬PLL∧PLR)∨(¬PLU∧PLD∧PLL∧¬PLR)	assumption
29:	(PLU∧¬PLD∧PLL∧¬PLR)∨(PLU∧¬PLD∧¬PLL∧PLR)	assumption
30:	PLU∧¬PLD∧PLL∧¬PLR	assumption
31:	PLU∧¬PLD∧PLL	∧ elim 30
32:	PLL	∧ elim 31
33:	PLU∨PLD∨PLL	∨ intro 32
34:	PLU∨PLD∨PLL∨PLR	∨ intro 33
35:	PLU∧¬PLD∧¬PLL∧PLR	assumption
36:	PLR	∧ elim 35
37:	PLU∨PLD∨PLL∨PLR	∨ intro 36
38:	PLU∨PLD∨PLL∨PLR	∨ elim 29,30–34,35–37
39:	¬PLU∧PLD∧PLL∧¬PLR	assumption
40:	¬PLU∧PLD∧PLL	∧ elim 39
41:	PLL	∧ elim 40
42:	PLU∨PLD∨PLL	∨ intro 41
43:	PLU∨PLD∨PLL∨PLR	∨ intro 42
44:	PLU∨PLD∨PLL∨PLR	∨ elim 28,29–38,39–43
45:	¬PLU∧PLD∧¬PLL∧PLR	assumption
46:	PLR	∧ elim 45
47:	PLU∨PLD∨PLL∨PLR	∨ intro 46
48:	PLU∨PLD∨PLL∨PLR	∨ elim 27,28–44,45–47
49:	¬PC	assumption
50:	PB∨¬PB	Theorem P∨¬P
51:	PB	assumption
52:	PB∧PW	∧ intro 51,7
53:	(PLU∧PLD∧¬PLL∧¬PLR)∨(¬PLU∧¬PLD∧PLL∧PLR)∨(PLU∧PLD∧PLL∧PLR)	→ elim 3,52
54:	(PLU∧PLD∧¬PLL∧¬PLR)∨(¬PLU∧¬PLD∧PLL∧PLR)	assumption
55:	PLU∧PLD∧¬PLL∧¬PLR	assumption
56:	PLU∧PLD∧¬PLL	∧ elim 55
57:	PLU∧PLD	∧ elim 56
58:	PLD	∧ elim 57
59:	PLU∨PLD	∨ intro 58
60:	PLU∨PLD∨PLL	∨ intro 59
61:	PLU∨PLD∨PLL∨PLR	∨ intro 60
62:	¬PLU∧¬PLD∧PLL∧PLR	assumption
63:	PLR	∧ elim 62
64:	PLU∨PLD∨PLL∨PLR	∨ intro 63
65:	PLU∨PLD∨PLL∨PLR	∨ elim 54,55–61,62–64
66:	PLU∧PLD∧PLL∧PLR	assumption
67:	PLR	∧ elim 66
68:	PLU∨PLD∨PLL∨PLR	∨ intro 67
69:	PLU∨PLD∨PLL∨PLR	∨ elim 53,54–65,66–68
70:	¬PB	assumption

71:	PM∨¬PM	Theorem P∨¬P
72:	PM	assumption
73:	PM∧PW	∧ intro 72,7
74:	(PLU∧¬PLD∧¬PLL∧¬PLR)∨(¬PLU∧PLD∧¬PLL∧¬PLR)∨(¬PLU∧¬PLD∧PLL∧¬PLR)∨(¬PLU∧¬PLD∧¬PLL∧PLR)	→ elim 4,73
75:	(PLU∧¬PLD∧¬PLL∧¬PLR)∨(¬PLU∧PLD∧¬PLL∧¬PLR)∨(¬PLU∧¬PLD∧PLL∧¬PLR)	assumption
76:	(PLU∧¬PLD∧¬PLL∧¬PLR)∨(¬PLU∧PLD∧¬PLL∧¬PLR)	assumption
77:	PLU∧¬PLD∧¬PLL∧¬PLR	assumption
78:	PLU∧¬PLD∧¬PLL	∧ elim 77
79:	PLU∧¬PLD	∧ elim 78
80:	PLU	∧ elim 79
81:	PLU∨PLD	∨ intro 80
82:	PLU∨PLD∨PLL	∨ intro 81
83:	PLU∨PLD∨PLL∨PLR	∨ intro 82
84:	¬PLU∧PLD∧¬PLL∧¬PLR	assumption
85:	¬PLU∧PLD∧¬PLL	∧ elim 84
86:	¬PLU∧PLD	∧ elim 85
87:	PLD	∧ elim 86
88:	PLU∨PLD	∨ intro 87
89:	PLU∨PLD∨PLL	∨ intro 88
90:	PLU∨PLD∨PLL∨PLR	∨ intro 89
91:	PLU∨PLD∨PLL∨PLR	∨ elim 76,77–83,84–90
92:	¬PLU∧¬PLD∧PLL∧¬PLR	assumption
93:	¬PLU∧¬PLD∧PLL	∧ elim 92
94:	PLL	∧ elim 93
95:	PLU∨PLD∨PLL	∨ intro 94
96:	PLU∨PLD∨PLL∨PLR	∨ intro 95
97:	PLU∨PLD∨PLL∨PLR	∨ elim 75,76–91,92–96
98:	¬PLU∧¬PLD∧¬PLL∧PLR	assumption
99:	PLR	∧ elim 98
100:	PLU∨PLD∨PLL∨PLR	∨ intro 99
101:	PLU∨PLD∨PLL∨PLR	∨ elim 74,75–97,98–100
102:	¬PM	assumption
103:	PO∨¬PO	Theorem P∨¬P
104:	PO	assumption
105:	PO∧PW	∧ intro 104,7
106:	(PLU∧¬PLD∧¬PLL∧¬PLR)∨(¬PLU∧PLD∧¬PLL∧¬PLR)∨(¬PLU∧¬PLD∧PLL∧¬PLR)∨(¬PLU∧¬PLD∧¬PLL∧PLR)	→ elim 5,105
107:	(PLU∧¬PLD∧¬PLL∧¬PLR)∨(¬PLU∧PLD∧¬PLL∧¬PLR)∨(¬PLU∧¬PLD∧PLL∧¬PLR)	assumption
108:	(PLU∧¬PLD∧¬PLL∧¬PLR)∨(¬PLU∧PLD∧¬PLL∧¬PLR)	assumption
109:	PLU∧¬PLD∧¬PLL∧¬PLR	assumption
110:	PLU∧¬PLD∧¬PLL	∧ elim 109
111:	PLU∧¬PLD	∧ elim 110
112:	PLU	∧ elim 111
113:	PLU∨PLD	∨ intro 112
114:	PLU∨PLD∨PLL	∨ intro 113
115:	PLU∨PLD∨PLL∨PLR	∨ intro 114
116:	¬PLU∧PLD∧¬PLL∧¬PLR	assumption
117:	¬PLU∧PLD∧¬PLL	∧ elim 116
118:	¬PLU∧PLD	∧ elim 117
119:	PLD	∧ elim 118
120:	PLU∨PLD	∨ intro 119
121:	PLU∨PLD∨PLL	∨ intro 120
122:	PLU∨PLD∨PLL∨PLR	∨ intro 121
123:	PLU∨PLD∨PLL∨PLR	∨ elim 108,109–115,116–122
124:	¬PLU∧¬PLD∧PLL∧¬PLR	assumption
125:	¬PLU∧¬PLD∧PLL	∧ elim 124
126:	PLL	∧ elim 125
127:	PLU∨PLD∨PLL	∨ intro 126
128:	PLU∨PLD∨PLL∨PLR	∨ intro 127
129:	PLU∨PLD∨PLL∨PLR	∨ elim 107,108–123,124–128
130:	¬PLU∧¬PLD∧¬PLL∧PLR	assumption
131:	PLR	∧ elim 130
132:	PLU∨PLD∨PLL∨PLR	∨ intro 131
133:	PLU∨PLD∨PLL∨PLR	∨ elim 106,107–129,130–132
134:	¬PO	assumption
135:	¬PS∧¬PC	∧ intro 23,49
136:	¬PS∧¬PC∧¬PB	∧ intro 135,70
137:	¬PS∧¬PC∧¬PB∧¬PM	∧ intro 136,102
138:	¬PS∧¬PC∧¬PB∧¬PM∧¬PO	∧ intro 137,134
139:	¬PW	→ elim 6.1,138
140:	⊥	¬ elim 7,139



□

8 First-Order Extension

8.1 Tile Linking

Instead of having mn constraints detailing the linking for every type of tile, we can condense them into a single constraint by using two nested universal quantifiers $\forall r.\forall c.$:

$$\text{Domain of r: } \{r \in \mathbb{Z} \mid 1 < r < m\}, \text{ Domain of c: } \{c \in \mathbb{Z} \mid 1 < c < n\}$$

$$\begin{aligned}\forall r.\forall c.(S(r,c) \wedge W(r,c) \rightarrow &(LU(r,c) \wedge LD(r,c) \wedge \neg LL(r,c) \wedge \neg LR(r,c)) \\ &\vee (\neg LU(r,c) \wedge \neg LD(r,c) \wedge LL(r,c) \wedge LR(r,c)))\end{aligned} \tag{8.1}$$

$$\begin{aligned}\forall r.\forall c.(C(r,c) \wedge W(r,c) \rightarrow &(LU(r,c) \wedge \neg LD(r,c) \wedge LL(r,c) \wedge \neg LR(r,c)) \\ &\vee (LU(r,c) \wedge \neg LD(r,c) \wedge \neg LL(r,c) \wedge LR(r,c)) \\ &\vee (\neg LU(r,c) \wedge LD(r,c) \wedge LL(r,c) \wedge \neg LR(r,c)) \\ &\vee (\neg LU(r,c) \wedge LD(r,c) \wedge \neg LL(r,c) \wedge LR(r,c)))\end{aligned} \tag{8.2}$$

$$\begin{aligned}\forall r.\forall c.(B(r,c) \wedge W(r,c) \rightarrow &(LU(r,c) \wedge LD(r,c) \wedge \neg LL(r,c) \wedge \neg LR(r,c)) \\ &\vee (\neg LU(r,c) \wedge \neg LD(r,c) \wedge LL(r,c) \wedge LR(r,c)) \\ &\vee (LU(r,c) \wedge LD(r,c) \wedge LL(r,c) \wedge LR(r,c)))\end{aligned} \tag{8.3}$$

$$\forall r.\forall c.(M(r,c) \wedge W(r,c) \rightarrow \text{ exactly-one}(LU(r,c), LD(r,c), LL(r,c), LR(r,c))) \tag{8.4}$$

$$\forall r.\forall c.(O(r,c) \rightarrow \text{ exactly-one}(LU(r,c), LD(r,c), LL(r,c), LR(r,c))) \tag{8.5}$$

$$\forall r.\forall c.(\neg S(r,c) \wedge \neg C(r,c) \wedge \neg B(r,c) \wedge \neg M(r,c) \wedge \neg O(r,c) \rightarrow \neg W(r,c)) \tag{8.6}$$

8.2 Win Condition

To determine if the level has a solution, we added the constraint described in Equation (4.10), which states that a level has a solution if there is a moat tile filled with water. More formally, a level has a solution if there exists a row and column such that the moat at those coordinates is filled with water. From this, we can create the following constraint:

Domain of r: $\{r \in \mathbb{Z} \mid 1 < r < m\}$, Domain of c: $\{c \in \mathbb{Z} \mid 1 < c < n\}$

$$\exists r.\exists c.(M(r,c) \wedge W(r,c)) \tag{8.7}$$

Predicate logic allows us to create a more concise constraint, rather than having to use a chain of disjunctions.

8.3 Path Length Constraints

To determine the length of a solution path, we used binary counting and the concept of "propagating a carry" to count the tiles that are links in the path. The four constraints used to accomplish this in Section 6.4 were going over every tile on the board except the first $(0,0)$, since it gets set just before the program runs into these constraints. We can represent the constraints in first-order logic by applying universal quantifiers for r and c to each of them.

Domain of r: $\{r \in \mathbb{Z} \mid 1 < r < m\}$, Domain of c: $\{c \in \mathbb{Z} \mid 1 < c < n\}$

$$\forall r.\forall c.(W(r_j,c_k) \rightarrow (\neg P(r_{j-1},c_{k-1},i-1) \wedge \neg N(r_{j-1},c_{k-1},i) \rightarrow \neg N(r_j,c_k,i))) \tag{8.8}$$
$$\forall r.\forall c.(W(r_j,c_k) \rightarrow (\neg P(r_{j-1},c_{k-1},i-1) \wedge N(r_{j-1},c_{k-1},i) \rightarrow N(r_j,c_k,i))) \tag{8.9}$$
$$\forall r.\forall c.(W(r_j,c_k) \rightarrow (P(r_{j-1},c_{k-1},i-1) \wedge \neg N(r_{j-1},c_{k-1},i) \rightarrow N(r_j,c_k,i))) \tag{8.10}$$
$$\forall r.\forall c.(W(r_j,c_k) \rightarrow (P(r_{j-1},c_{k-1},i-1) \wedge N(r_{j-1},c_{k-1},i) \rightarrow \neg N(r_j,c_k,i))) \tag{8.11}$$