Queen's UNIVERSITY

GRP_10: Overflow

Stuart Fong (20sjkf)

Nathan Fong (20njhf)

Vincent Proulx (20vp24)

Wasiq Wadud (17ww41)

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

October 29, 2022

Abstract

We are building a model for a game called 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 at the top down to the moat of a sandcastle. This is accomplished by rotating square tiles containing either a straight path, curved path, or a bridge. We created 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.

Propositions

$S(r, c)$ represents if a straight tile exists at coordinate (r,c)
$C(r, c)$ represents if a curved tile exists at coordinate (r,c)
$B(r, c)$ represents if a bridge tile exists at coordinate (r,c)
$M(r, c)$ represents if a moat tile exists at coordinate (r,c)
$O(r, c)$ represents if a ocean tile exists at coordinate (r,c)
$W(r, c)$ represents if the tile exists at coordinate (r,c)
$LU(r, c)$ represents if the tile at coordinate (r,c) is linked to the tile at (r,c-1)
$LD(r, c)$ represents if the tile at coordinate (r,c) is linked to the tile at (r,c+1)
$LL(r, c)$ represents if the tile at coordinate (r,c) is linked to the tile at (r-1,c)
$LR(r, c)$ represents if the tile at coordinate (r,c) is linked to the tile at (r+1,c)
P represents if an arrangement of tiles is a valid solution to the level
$N(i)$ represents if a solution to the level has path length i.

Constraints

1. Each of the five tile types of a straight tile, a curved tile, a bridge tile, a moat tile, and an ocean tile will have their respective constraints on how they can be connected and layed out.
   For example, straight tiles can only be linked to tiles above and below, or left and right of its position:

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

2. Since there are only five types of tiles, a tile must be at least either a straight tile, a curved tile, a bridge tile, a moat tile, or an ocean tile - otherwise it is a blank tile.
   For example, if a curved tile exists at row r and column c, there cannot be another path tile at the same location:

$$C(r,c) \rightarrow \neg S(r,c) \wedge \neg B(r,c) \wedge \neg M(r,c) \wedge \neg O(r,c)$$

3. If a tile contains water and is not the ocean tile, it must be connected to a path that leads to the ocean tile. otherwise it must be connected to another tile containing water:

$$W(r,c) \land \neg O(r,c) \leftrightarrow (W(r-1,c) \land LU(r,c)) \lor (W(r+1,c) \land LD(r,c)) \\ \lor (W(r,c-1) \land LL(r,c)) \lor (W(r,c+1) \land LR(r,c))$$

4. There are numerous linking constraints between the tiles that are the inverse of one another, the constraints consist of the following: if a tile links up, then the tile above must link down; if a tile links down, then the tile below it must link up; if a tile links left, then the tile on the left must link right; if the tile links right, then the tile on the right must link left.

   This idea can be further explored with the following propositional logic example. This example will be for the example of if Tile A is linked to Tile B, iff. B links back to A. This is only for one of the examples above, as the prompt asked for the pattern of propositional logic rather than the exhaustive list.

$$LU(r,c) \leftrightarrow LD(r-1,c)$$
$$LL(r,c) \leftrightarrow LR(r,c-1)$$

5. The user has won the game if there is water in a moat tile:

$$P \leftrightarrow (M(r,c) \land W(r,c))$$

6. When calculating the length of a solution path, we have that if there are i number of water tiles, then the solution path has length i. For example in a level of size 2x2:

$$W(1,1) \land W(1,2) \land \neg W(2,1) \land W(2,2) \rightarrow N(3)$$

Model Exploration

1. In our code we said that a bridge can connect up and down, or left and right. We intended it as up down or left right or both of the conditions true i.e. both directions, but for the case where the water passes through the bridge once, we did not restrict the path from going in 3 directions, from 1. We fixed it by using strict truth table-like conditions, where a bridge can only connect left, right, not up and not down, or up, down, not left and not right and a third condition where we have up and down and left and right.

$$B(r,c) \land W(r,c) \rightarrow LL(r,c) \land LD(r,c) \land LL(r,c) \land LR(r,c) \\ \land W(r-1,c) \land W(r+1,c) \land W(r,c-1) \land W(r,c+1)$$

2. We had that for our single direction tiles like moats, its possible that a moat can link with a moat allowing there to be water. We never restricted there from being water. We fixed it by specifying that a moat can't link with another moat.

$$M(r,c) \wedge W(r,c) \rightarrow \neg LU(r,c) \wedge \neg LD(r,c) \wedge \neg LL(r,c) \wedge \neg LR(r,c)$$

3. Once we found an algorithm for calculating the length of a solution in logic, we needed to find the solution with the maximum path length. We first created a vector of length equal to the total number of tiles in the level, where an element i is set to 1 if there exists a solution with path length i, and 0 otherwise. Once we did this, we found that it was trivial to implement in logic, where $N(i)$ is true if $N(i+1), N(i+2), \ldots, N(\text{n\_row} * \text{n\_col})$ are false. Knowing this, we decided against the implementation of finding the maximum length in logic, and instead coded it using basic Python.

4. When we were looking for the length of each path, we noticed that there were more solutions to the levels than we initially thought. This was due to the linking of the tiles that did not contain water. When it does not contain water, we had no constraints for the linking the tile, so there were $2^4 = 16$ extra solutions for each tile that did not contain water. We solved this by adding a constraint where a tile that does not contain water does not link up, down, left, or right.

$$\neg W(r,c) \rightarrow (\neg LU(r,c) \wedge \neg LD(r,c) \wedge \neg LL(r,c) \wedge \neg LR(r,c))$$

5. When calculating the length of a solution path, we initially added that there can only be exactly one length proposition that is true, as a path can only have one number for its length. We found later that this is not needed, as a solution has only one arrangement of the water tiles.

Jape Proof Ideas

1. If a tile is not a corner tile and not a straight tile and (links up or links right) and (links down or left), we know that a tile that (links up or links right) and (links down or left) is a bridge tile or a straight tile. From this information, we can deduce that the tile is a bridge tile. (Due to jape limitations, variables had to be changed for proofs: C = "Corner tile", S = "Straight tile", P = "Link left", D = "Link right", B = "Bridge tile".)

$$\neg C \wedge \neg S \wedge P \wedge D, P \wedge D \rightarrow (B \vee S) \vdash B$$

2. If an ocean tile is directly connected to a moat tile, the puzzle has a length of 2, and the game is automatically won. (Due to jape limitations, variables had to be changed for proofs: Q2 = "Path length of 2", E = "Game has been won".)

$$Q2 \rightarrow E, Q2, \vdash E$$

3. If There exists a game on a board of 2 by 2 tiles, the longest possible solution is of length 3. In this solution, the ocean tile connects to a corner tile, which connects to a moat tile. This is a proof that with the following

premises, for all games with a path length of 3 and a second tile with water on it the game is won, there exists a game with a path length of 3, and there exists an ocean tile that has water on it, as well as a corner tile with water on it, there is a solution. (Due to jape limitations, variables had to be changed for proofs: Q3 = "Path length of 3", y = "A game", z = "A first tile", x = "A second tile", G = "Ocean tile", C = "Corner tile", P1,P2,Pn = "There is water on the n$^{(}th$) tile", E = "Game has been won".)

$$\exists y.(Q3(y)), \forall y.\forall x.((Q3(y) \wedge P2(x)) \rightarrow E), \exists z.\exists x.(G(z) \wedge P1(z) \wedge C(x) \wedge P2(x)) \vdash E$$

Requested Feedback

1. When we are finding the longest path, you can see that we made a proposition for every number and brute-forced a solution that does not have a very efficient run-time complexity. Would you have an idea on how to improve this method?
2. When we were coding the length of a solution path, we could not reuse the previous water proposition class as it already had a decorator with the encoding E. We had a workaround to create an identical class WaterF to use for another encoding F. Is there another way to use the same proposition for two encodings without having to duplicate everything?
3. Do you think that the tile propositions for straight, curved, and bridge tiles are even needed? They currently exist in our current program, but we don't think they contribute to our solutions.

First-Order Extension

To extend our model to a predicate logic setting, we would use propositions that represent sets of tiles such as all the tiles that are in a solution path, or types of paths such as straight and curved tiles. We can use them to place constraints on the general behaviour of a type of tile. For example, we can create a constraint for the linking of all straight tiles such that

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

instead of adding constraints that are unique to a single tile on every tile on the grid. This method would lead to a problem where tiles on edges of the level would not be covered and we would get an index out of range error when the solver would try to link to a tile that does not exist out of the level's bounds. To fix this issue, we would need to pad the borders of the level with empty tiles so that all linking tiles of a certain type follow the same constraints.
We can also improve how we determine if a path is a valid solution with the constraint where a path is a valid solution if there exists an r and c such that a moat tile $M_{r,c}$ contains water:

$$P \rightarrow \exists r.\exists c.(W(r,c) \wedge M(r,c))$$

This again allows us to generalize a solution to apply to any moat tile without having to create unique constraints for each moat.