User Manual for Carcassonne Pre- & Post-Processing

Isabell H.

30th October 2022

Overview

If you are reading this user manual, it is likely that you are working on the Carcassonne project. I cannot give you tips regarding modelling, but I hope this post-processing script and/or data files save your time and energy in some way.

If you prefer to start from scratch, do not continue to read. If you feel these resources are not compatible with your ideas or model, do not change your model to fit them. I hope this saves people from reinventing the wheels, but there are so many ways to model this problem. Don't let this put limitation on your ideas.

This repository contains both the pre-processing and post-processing resources for the Carcassonne project. Team 7 and Team 18 collaborated on them for their final project in Fall 2022.

The dzn folder contains 12 dzn files of various map sizes. Besides the 84-tile instance, corresponding to the real game, there are 9 additional instances. They range from containing 42 to 168 tiles, which can help you understand your model's scalability.

The post-processing Python script helps to check the correctness of the model by providing a decent visualization for the results. The script can run the model through the MiniZinc API in Python, get the results for the map and identify which tile they represent. It then outputs all the tile images as a grid, and saves the resulting picture as png.

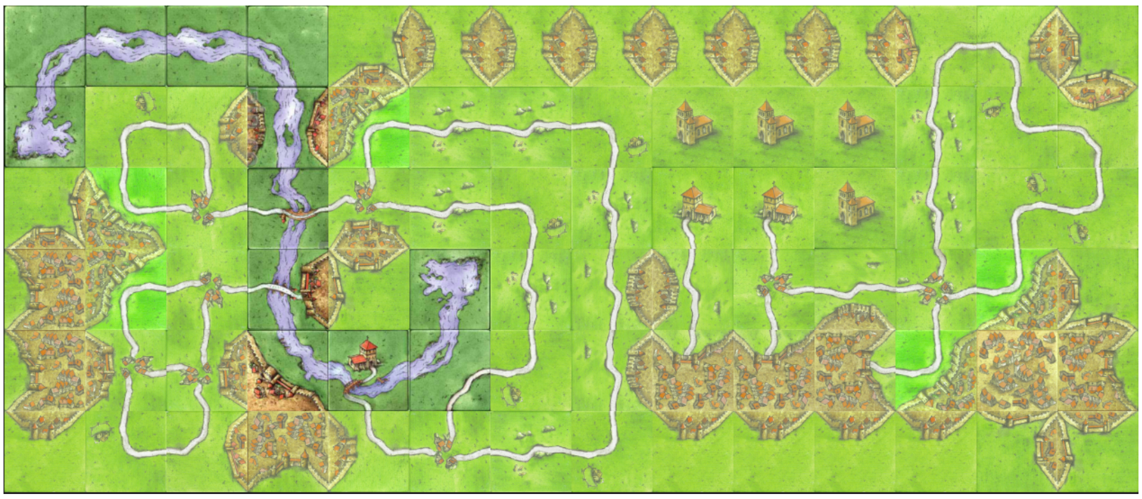

Figure 1: Output of an 84-tile map satisfying all the constraints, using post_processor.py.

Important terminology 1: We can classify possible edges of a tile into 4 types. T = castle/city/-town. F = field. P = path/road. R = river. We define them as `enum Types = {T, F, P, R};`.

Important terminology 2: We can represent every tile with an array of length 4. Each entry is of one of the enum Types, representing one of the four edges of the tile. [Left, Top, Right, Bottom]. The order goes clock-wise. For example, `[F, F, F, F]` represents an abbey.

You do not have to use these terminologies, but note that if your input/output formats differ too much, you may have to write parsers in order to use the resources provided here.

Dzn Files

The base game has 84 tiles. In case you have not noticed, all examples on Peter Novig's webpage are maps with dimension 12×7. In the dzn folder, there are 3 data files that start with 084. Those are the data files representing the real game with 84 tiles. Among them, I would suggest you try running your model for 084-12-7 or 084-14-6 first. Those two are likely to be easier/faster for backends. In the following paragraphs, I will use 084-12-7 as an example to show you how to understand the data files.

There are 84 tiles in the map.
`tileCount=84;`

There are 12 columns in the map. The length of the map is 12. Note that we do not always enforce length1>width. After implementing the constraint that all abbeys orienting right-side up, 168-14-12 and 168-12-14 can output distinct maps. You can make your own choice for symmetry-breaking or other purposes.
`length1=12;`
*`length` is a keyword in MiniZinc, so we call it `length1` here.

There are 7 rows in the map. The width of the map is 7.
`width=7;`

084-12-7 is the name for the data file, as it specifies the map size and dimensions.

There are 26 distinct tiles in the game. Two tiles are *distinct* if it is impossible to rotate one to obtain the other.
`tileCountWRotation=26;`

There are 4 possible kinds of edges for a tile. We use enum Type to represent them.
`numOfTypes=4;`

The length of the array is always 26. Their sum should add up to the number of tiles, which is 84 in this instance. The i-th entry represents the number of a specific kind of tile in this map.
`TilesCardinality = [4, 2, 9, ..., 1, 1];`

This is a 2d array with 26 rows and 4 columns. Every row represents a distinct kind of tile. The i-th entry in `TilesCardinality` is the cardinality of the i-th row here. For example, the cardinality of tile `F,P,T,P` is 4. Note that all rotations of `F,P,T,P` are not distinct from `F,P,T,P`, therefore they are counted in the same entry.
`TypeOfTiles = |F,P,T,P|P,F,F,F|...|P,P,P,P|;`

That is all for the data files. If you need to make changes to them to fit your model, feel free to do so as long as you give proper credit. Cite either this repository as a whole, or the maker of dzn files, *Isabell H*.

Python Scripts

There are two slightly different versions that share the same core functionalities. For post_processor.py, the required output format (from the model) is an 84×4 array of type enum `{T,F,P,R}`. This array should take rotation into account. One should be able to reproduce a complete map with that array. The post processing script runs the model under a given solver with a given data file, and then output an image. Change parameters for loading model, solver, and data if needed. The function tileToImage is the function that finds the corresponding image files and compute rotation at runtime. If no valid image is found, it will put an abbey ($\{F, F, F, F\}$ using our terminalogy) in its place, mainly to tell you the model needs debugging. image_grid is the function that puts together the grid, which we got from stackoverflow.

Additionally, post_processor_team_18.py can generate multiple solutions as the solver finds them. Change parameters for map size if needed.

If you need to use or alter part(s) of the scripts, feel free to do so as long as you give proper credit. Cite this repository as a whole, even if you only use one version of the script.

Good luck modelling! I wish you an enjoyable and rewarding experience with this project. Contact me (Isabell H.) if you have suggestions or issues regarding this repository.

Third-party Resources

Neither of the teams own a physical copy of the game. In order to produce actual images for post processing, we decided to find scanned pictures of the tiles online. Fortunately, there are many implementations of the Carcassonne game on github. From two of them, we found all the scanned pictures we need [1] [2]. Pictures of regular tiles are taken from `https://github.com/tsaglam/Carcassonne`. Pictures for the river tiles are from `https://github.com/istewart/carcassonne`.

References

[1] Tsaglam, "Tsaglam/carcassonne: A digital version of the board game carcassonne, implemented in java. this desktop computer game supports up to five players at the same time (shared-screen multiplayer mode)." [Online]. Available: https://github.com/tsaglam/Carcassonne

[2] Istewart, "Istewart/carcassonne: Implementation of boardgame carcassonne." [Online]. Available: https://github.com/istewart/carcassonne