Mario Level Generator Assignment: IMGD 4100

Mikel Matticoli and Diana Kumykova

Optional Extras implemented:

(1) Manual analysis of files, (2)automatic (programmatic) analysis of files, (3)automatically created transition table

External Code References:

We used the following post as a guideline to designing the algorithm that goes through the probability list of each chunk in order to get the next chunk in the sequence:
StackOverflow post on weighted randomness

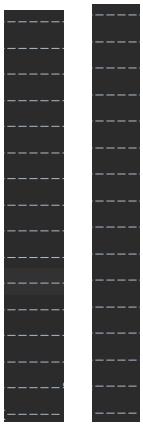
Known issues:

With some manually generated Markov tables, such as that for hopper_lvl7, sometimes a very tall block occurs right after low blocks, and it is impossible for the AI to jump onto it. This isn't because of an error in the table, but rather because that specific high block in the original level has a series of stair-like blocks leading up to, whereas in the generated levels this may not always be the case due to the nature of the weighted random procedural generation.

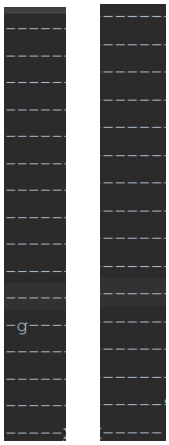
Data representation:

We chose to represent our data for each level within a text file that contains the following information about each "chunk" of the map:

The first line of the file listed the number of chunks, and height of all the chunks in rows. This was followed by a list of chunks comprised of the string representation as written in the original level, and the probability that any of the other chunks in the file will come immediately after in the form of a row of probabilities. For example, if chunk 1 had a 50% chance of coming after chunk 4, then column 1 of the probability row for chunk 4 would contain the value of 0.50. For the manual analysis of files chunks were taken as vertical slices of the map, and chosen roughly based on their "uniqueness" as judged by the person analyzing, but there was some leeway given in order to make the process more bearable. For example, the following 2 chunks were treated as similar enough to be considered the "same" for probability's sake:



While the following 2 were considered unique and had their own probability rows:



Corresponding to the data in the text files are the data structures in the code: a hashmap of chunk strings keyed to id numbers, and a 2D array that mirrors the probability table that we created in class. The array is organized so the index matches the chunk ID specified in the hashmap, and the value at any given point equals the probability for the Y-coordinate chunk to appear after the X-coordinate chunk, i.e. [1][2] = .25 means there is a 25% chance that chunk 2 appears after chunk 1 in the generated map.

The files analyzed manually were pulled from the hopper and notch folders. For these, we looked for patterns in the source level and identified major “landforms” to specify as unique chunks. We then developed the transition probabilities based on how often each other chunk appeared after the given chunk, stored as described above. For these input files, the chunks all remained the same height but varied in width.

The automatic analysis accepted any existing level file, and created the same HashMap+Array representation as stored in the file. The interface for the MarkovModel class allows for loading the chunk and transition data from a file, writing it to a file, or parsing it by processing an existing level file, all in a similar manner. This allowed us to interact with a clean, simplified representation of the data in the actual generator class.

Reflection on strengths and weaknesses:
An obvious weakness of our generator is that by only judging chunks by taking vertical slices we leave out the potential to create more varied upper and lower halves of the map generated. The variety of topology in the output level is limited to vertical slices of the input levels, which breaks with a very small dataset. This is in fact an inherent limitation in the Markov algorithm, as the output will always be some constrained random subset of the input. A benefit of this however is that it consistently generates playable levels - any given chunk will never be followed by a chunk that is not known to work in that order within an existing playable level.

Additional documentation on the project organization and data storage format are included in the README for the project:
https://github.com/matticoli/IMGD4001-MarkovMario/blob/auto-markov/README.md

What we would improve if we had time:
One limitation of our implementation is that the analyzer only processes a single file. Our first step given the time would be to allow multiple level files as inputs, so that more varied levels can be generated. This would allow us to synthesize patterns across multiple level generators and develop more complex and varied levels. An additional enhancement would be specifying chunk width. To limit scope, we decided the width of a chunk would be a vertical slice of width 1 for the purposes of the automated analysis. It would be interesting however to see how the output would change with different sized chunks with a larger dataset than what we were able to accommodate in our manual analysis.