C project report

Łukasz Jakubowski lj1019 | Maciej Kaszlewicz mkk3118 | Paweł Kroll pmk19 | Stefan Radziuk sar119

Department of Computing, Imperial College London

June 19, 2020

1 Introduction

In the following report, we present the results of our work in developing the assembler program. The second part elaborates about the Wave Function Collapse Algorithm which we implemented as our extension. Final pages comment on the experiences which we gained through this project.

2 Assembler

2.1 Structure and Implementation

Our design of this part of the exercise was inspired by the emulator implementation, we have just thought of the process "in reverse." We have incorporated the double pass solution. The key components of the structure included:

- `assemble` - the entry point of the program. Writes the output into a binary file.
- `parser` - conducting the first pass through the assembly file, detecting labels and saving them in a designated list of structs.
- `assembler` - the utility which went through the assembly lines second time and supplied those to the `tokenizer` and also supplied its output to helper functions.
- `tokenizer` - which split the strings (assembly lines) into tokens such as `opcode` which given us required level of abstraction.
- `assemble_..` - functions which translated tokenized instructions into their 32-bit equivalent representation.

Please have a look at the following diagram for a visual aid:

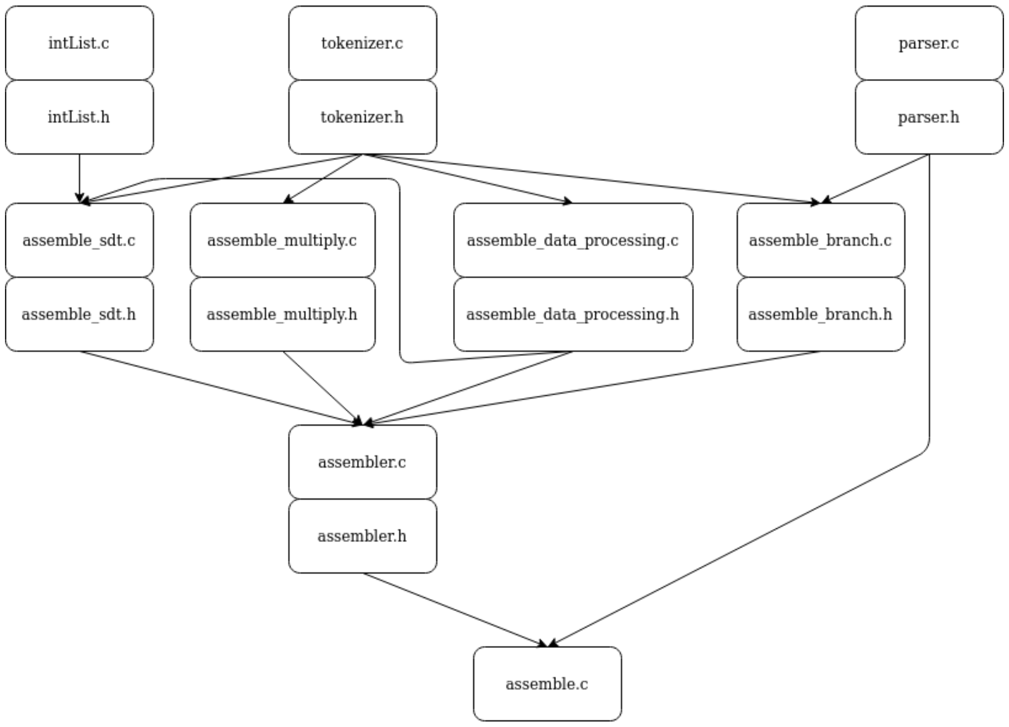

Figure 1: Diagram representing the Assembler structure, transitive. Arrows denote inclusion.

We have reused the code from the testutils library built in the previous weeks. This was highly beneficial during the testing of translation into the binary representation.

3 Extension

3.1 The Wave Function Collapse algorithm

The wave function algorithm was first described and implemented in C# by Maxim Gumin[1]. It is most commonly used for generating game assets such as maps, textures and various game art. [2]

Our implementation of the wave function collapse algorithm focuses on the *Simple Tiled Model* version of the algorithm, as opposed to the *Overlapping Model*.

The *Simple Tiled Model* focuses on finding an arrangement of given tiles satisfying a given set of rules. It can be used for generating game maps as well as game art.

In comparison, the *Overlapping Model* aims to transfer local features from a single input bitmap to a larger output bitmap it generates. Its primary applications are texture synthesis and game art generation. The *Overlapping Model* can be represented as the *Simple Tiled Model* but with an additional step of generating a tileset from the input bitmap, so we believe our implementation can be modified to implement the *Overlapping Model* with relatively few modifications.

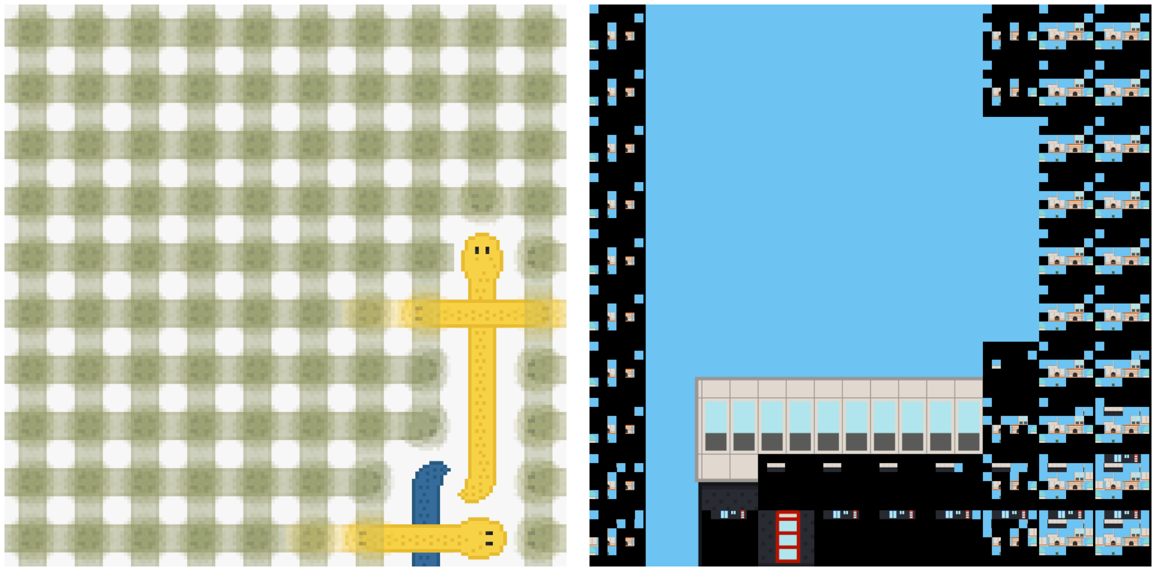

Figure 2: Superpositions as average colours and as grids

3.2 Our implementation of the algorithm

There are two terms worth mentioning before considering the design of the algorithm: *superposition* and `entropy`, both inspired by phenomena in the field of quantum mechanics.

Superpositions are collections of all the possible states of a given cell. In our implementation we represent them as 64-bit bitstrings to take advantage of C's efficient handling of bit operations. Our implementation can represent superpositions visually both as a grid of possible states and by displaying the average colour of each pixel's possible state (Figure 2).

Entropy is the count of possible states a given superposition holds. This means that we can make higher certainty guesses about low entropy cells than high entropy cells. There are two special values of entropy to have in mind:

- entropy equal one indicates a *collapsed* cell, i.e. we can be certain the state of that cell,
- entropy equal zero indicates a contradiction – collapsing the given cell to any known state would result in a breach of the rules.

Below follows a simplified outline of our implementation of the wave function collapse algorithm.

```
defn Main():
  GetRules()
  Loop until all cells have collapsed:
    Observe()
    Propagate()
  OutputObservations()
```

Figure 3: An outline of our implementation of the wave function collapse algorithm. Pseudocode adapted from [3] to more closely reflect our implementation.

`GetRules()` fetches the tiles and the corresponding rules from an XML file and generates any additional tiles as specified in the input file. In our code, it is performed by `parseXml` and various functions from `image_utils`.

`Observe()` looks for the lowest entropy cell. If that cell has not yet collapsed, it is randomly collapsed to one of its possible states. The corresponding functions in our implementation are fi`ndMinimumEntropy` and `collapseRandomly`.

`Propagate()` traverses the matrix by recursing into the neighbours of the current cell, starting from the cell selected by `Observe()`. It unsets any recently invalidated states held in each cell's superposition in accordance with the tiling rules. This stage is implemented in our code by `updateNeighbors`.

At this point it might occur that some cell's entropy drops to zero, i.e. it cannot be collapse to any state. This happens because the wave function collapse algorithm does not succeed every time it runs. Some implementations modify the algorithm by providing backtracking: each time a contradiction is reached, the algorithm returns back to the previous random decision it has taken and makes a different choice. We have chosen not to implement backtracking as it does not offer significant improvement – it is impossible to tell in polynomial time how far the backtracking has to go to achieve its goal. Since the choices at the beginning of the runtime have a greater impact on the state of the superpositions than the later choices, it seems reasonable to assume that the changes we are interested in reverting have been made at the start of the runtime. Therefore, in many cases at our scale backtracking would simply take longer than starting again from the beginning.

If the loop reaches a point where all cells have collapsed, it returns `true` and prints a success message.

3.3 Our I/O

For our program to work as an input we need a tileset in which each tile is in a separate image file and an xml file. Some sample xml files are provided in the repository, each file specifies the number of tiles in each column and row and adjacency rules for each tile. Adjacency rules describe what kind of edges does the tile have, i.e. tile separating water and lava could have `left="water", up="water_lava_edge", right="lava", bot="water_lava_edge"`.

Then tiles can only be adjacent if their edges match, i.e. tile with `right="water"` can be on the left side of the tile with `left="water"`. We believe that our format of defining the rules is more succinct and maintainable than Maxim Gumin's approach, whose implementation requires the user to list which tiles a given tile can neighbour (Maxim's approach, however, does enable the user to define some unusual rules which might be difficult to express in our format). If then tile is already decided then both styles just display it in the correct place, otherwise one of them calculates average colours of pixels of possible tiles and the second one draw mini versions of possible tiles.

3.4 Our results

Figures 2, 4, 5, 6 generated by our implementation demonstrate some common applications of the wave function algorithm. The tileset used to generate figure 6 is Overworld RPG Tileset[4]. The remaining tilesets have been designed by Stefan.

3.5 Testing our implementation

Since our implementation relies heavily on random numbers generated at runtime, we decided to test it using assertions in our code instead of creating a stand-alone test suite. We placed multiple assertions throughout our code to ensure that the required invariants were satisfied where applicable. This way we did not have to write any test cases ourselves – the randomness ensured that all modules of our implementation work correctly under multiple unique starting conditions.

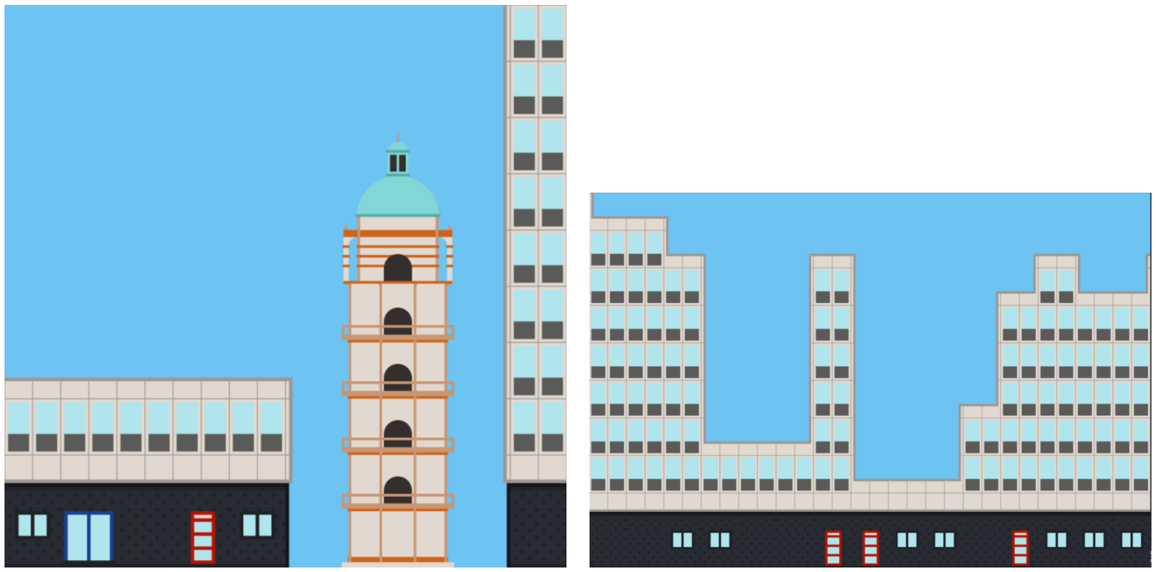
Figure 4: Our implementation's take on the South Kensington Campus

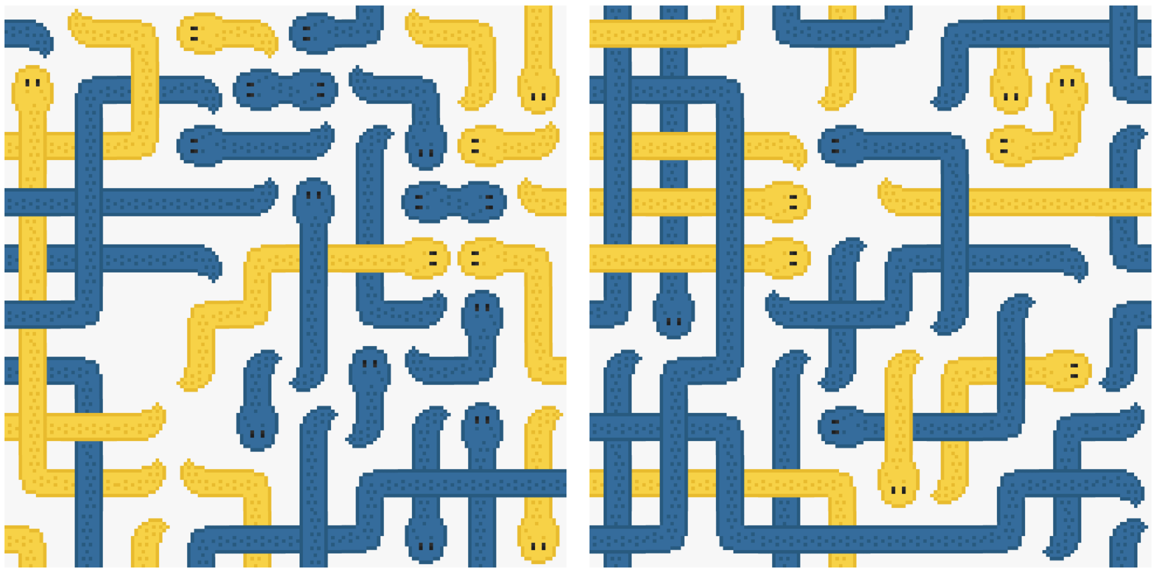
Figure 5: Our implementation's attempt at generating labyrinths inspired by the logo of the Python Programming Language

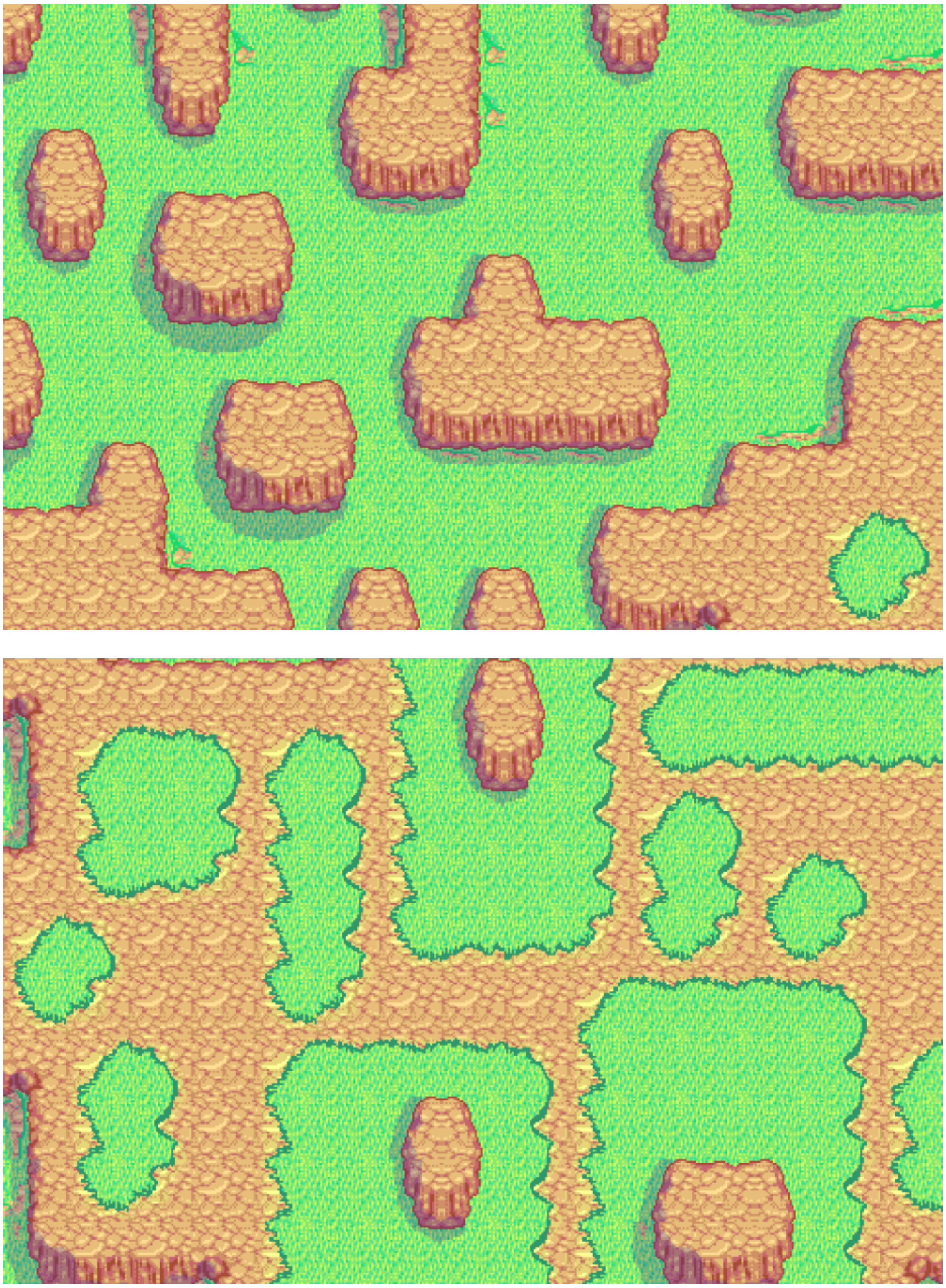

Figure 6: RPG maps generated by our implementation using the Overworld RPG Tileset[4]

4 Reflections

4.1 Group Reflection

Our group can proudly call itself a team, we are one. We had a strong feeling that every individual was willing to contribute to the project to the best of his abilities. We did not have any objections to the split of work. Of course, sometimes, specific team members were doing more than others but on the scale of the whole project timeline, it evened out. Communication channels were frequently used, which provided a smooth flow of information between us. We found summaries of video calls very useful in clearing out the confusion about details.

4.2 Individual Reflections

- Łukasz:
  The C Project was my first experience of long term programming commitment. It came to my great surprise that it differs a lot from both programming assignments and hackathons. It truly requires the understanding of Git (which, I have to admit, is not my favourite) as well as advanced communication skills. From the very beginning of our cooperation, I was positively shocked by the energy of my teammates. Maciej prepared the structure of the Emulator on the first meeting! The Extension part was especially motivating, as it gave some measurable results at the early stage of development. I am proud that we managed to push it this far. The willingness to help of all team members was on spot, I felt the professional atmosphere. Stefan deserves special recognition for having God-like skills in Version Control. I could not imagine a better team for this undertaking.

- Stefan:
  This project was the first time I have had an opportunity to develop software in a team. All of my teammates and myself have remained well-organized through the project and I believe it was crucial to making this such a great experience. To help us document our efforts we founded a Discord server with multiple topical channels which we used for communication. We held weekly Discord calls to decide what needs to be done and split the work and wrote down outtakes from the calls using Google Docs. We were quite flexible in our approach to team roles – there was no fixed leadership, which allowed each of us to lead during a part of the project best suited towards our skills. For example, while Maciek laid out the plan for implementing the Emulator, I proposed the extension topic and outlined our implementation of the wave function collapse algorithm. I have also had a lot of fun in the breaks from programming when I was designing tilesets instead.

- Maciej:
  Although it wasn't my first programming group project, it was one in which I experienced a good teamwork environment. I was impressed by how everything worked smoothly and efficiently in our team. When we started working on the extension, it allowed me to research and focus on graphics generating I was interested, even before the project started. On top of it, I could see how others work and improve my code and git control based on that. Some of the methods we used to keep everyone up to date I will use in my future group projects.

Overall I am very satisfied with how our group worked and my individual development during the project time.

- Paweł:
  I feel like we fitted well as a group. We had no problems in understanding each other and conveying our views and thoughts. I honestly thought that I would have problems using git and cooperating on a project because I have never done anything quite like this. However, git proved to be beginner-friendly and I learned it pretty quickly. The other thing, I came to realize during this project was that it is always necessary to leave some time for debugging, especially if you are working under a tight schedule. All in all, I enjoyed this experience and would love to participate in similar group projects in the future.

References

[1] Maxim Gumin. Bitmap & tilemap generation from a single example with the help of ideas from quantum mechanics. `https://github.com/mxgmn/WaveFunctionCollapse`.

[2] Hwanhee Kim, Seongtaek Lee, Hyundong Lee, Teasung Hahn, and Shinjin Kang. Automatic generation of game content using a graph-based wave function collapse algorithm. In *2019 IEEE Conference on Games (CoG)*, pages 1–4. IEEE, 2019.

[3] Isaac Karth and Adam M. Smith. Wavefunctioncollapse is constraint solving in the wild. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, FDG '17, New York, NY, USA, 2017. Association for Computing Machinery.

[4] Tayoko. Overworld rpg tileset (licensed under cc by-sa 3.0). `opengameart.org/content/overworld-rpg-tileset`.