Maze.pnm

The program regards neither every desirable situation nor every invalid situation outside the pre-set range. In this project, the program always simulates the situations with a given .pnm file devoted to a map as the input. The program uses std::stream to input the maze. When a location point and a map have been given, the program identifies whether the location point has the four closest walls, to which the program applies red, blue, brown, and green. Furthermore, every valid map consists of some 1's and some 0's; a 1 is a block and a 0 is an available position. A wall must have at least one 1's. The process of hypnosis succeeds in the target with only four closest four directly facing the target. When a point has been occupied by a 1, setting the point as a location point is prohibited by the program.

```
P1 11 15
111111111111111
111100010000001
100001110000001
100000000000001
100000000000001
100110000100001
100100001000001
100000000000001
100000000000001
100000000000001
111111111111111
```

Figure 1., The Original State in This Map

BSP Method

Instead of aggregating all the wall into a buffer by scanning the whole map, the robust method referring to BSP method only attributes the visible walls surrounding Location Point to the final output.

This method always resorts to a binary tree. In the class named BSPTree, it has a member variable named head, and some necessary member functions; such as search(), clean(), build(),

push(), etc. For the sake of making the code robust, the program has some intrusive containers implemented by using Boost.

Specifically, the type of head is BSPSegment type. This is a struct. Inside BSPSegment, there are four member variables, which are positive_side, negative_side, parent, and periphery. As the program divides every subset of the map into two parts, the program finds every periphery by using binary search in O(logN) time recursively. Every four peripheries forms a segment. Every segment is a node in the BSP tree. The principle for searching the periphery of every subset is to find the line lying on some 1's in either horizontal way or vertical way. That is, the four boundaries are the peripheries of some segments as well. The first approach is to choose the middle line of every segment. The variable named positive_side refers to the right side of a vertical periphery or the upper side of a horizontal periphery, vice versus. Either positive_side or negative_side is a BSPSegment pointer. If positive_side equals to negative_side, it indicates the current object is one of the leaves in the tree. Therefore, every segment does not have any 1's inside. The parent variable points to the previous segment, which contains not only the current segment but also some other smaller segments.
Once the binary tree has been built for a map, the program recycles it for every Location Point without scanning the map. Figure 2. shows the example of building the tree. Since every segment has four points and some bigger segments contain some smaller ones, searching for the four closest walls directly facing the target sometimes needs to back track from a leaf node to its parent node. In the figure, the tree contains all the bigger segments and smaller segments. However, the map cannot label all the bigger segments for the sake of easier reading.

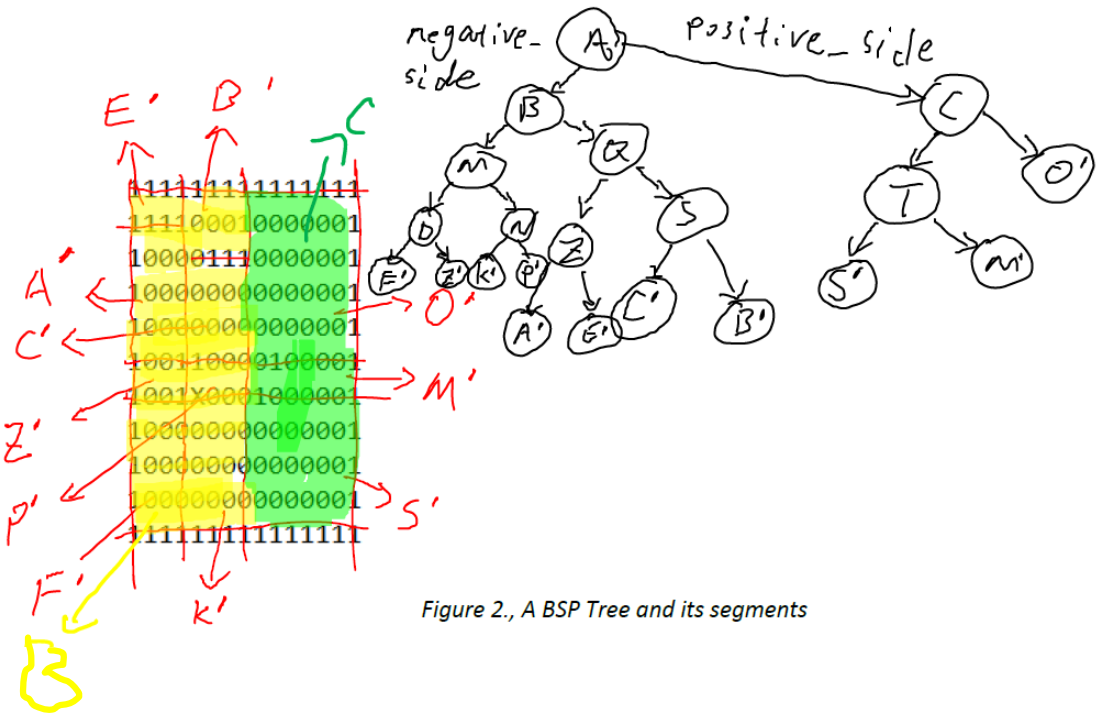

Figure 2., A BSP Tree and its segments

Naïve Method

As opposed, with a given location point in a map, the naïve method scans the area from the location piece by piece in order to find the closest four walls.
Figure 3. below shows the method being used at three location points:

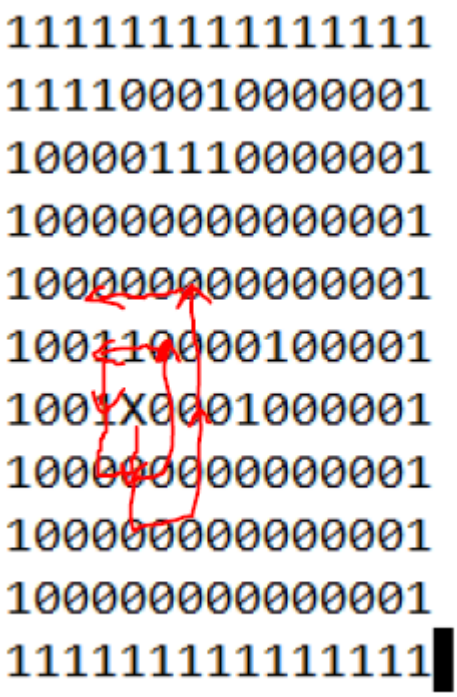

Figure 3., Naïve Method Scanning The Walls Counterclockwise