Image Processing Report

Marcus Friis & Mathias Hygum Pedersen

October 21, 2023

Contents

0.1 Intro

This report is based on the Mini Project in which it was required for groups of 2 to make a program that can calculate the points given for a certain board in the game Kingdomino. The program would have to find "Crowns" in some of the fields in the board and also be able to discern which fields are connected to properly calculate the points for each board, which will be described in the following chapters.

0.2 KingDomino

The flowchart seen in Figure 1 is the general program in kingdomino.py that runs at the start, using the different functions seen in the flowchart to end up with printing the points.

It starts with defining the input image as raw_image which will be used further on throughout the program, it then goes through the fill_board_terrains function which will be explained in section 0.3 and continues on through get_crowns from section 0.7. Both these functions use the raw_image variable and get_crowns also use the variable crowns.

Then we go through a while loop in which we continuously check if the array named "unvisited" is 0, to see if there are still cells on the board that are unvisited, and if they are, the visit_node function from section 0.5 will run again, if not, then it will go through the print_board_points function from section 0.6 and finish.

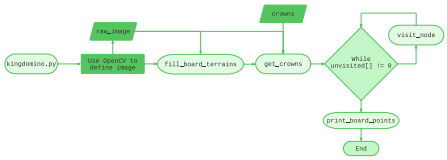

Figure 1: Flowchart of the general program, referencing to the other flowcharts throughout.

0.3 Fill Board Terrains

This function here is relying heavily on the get_best_terrain_match function from section 0.4 which does the work of actually finding the best matching terrain, as the name implies.

The function fill_board_terrains from Figure 2 starts with using the raw_image variable from earlier and converts it into the HSV color space where it then goes directly into 2 for loops, one for the 5 rows, and one for the 5 columns to make sure all the cells in the image has been accounted for. If all cells have not yet been visited it will mask a single cell, creating the variable cell and then go into the get_best_terrain_match function mentioned earlier which uses the just made cell variable. Then it goes on to the next cell, until all cells have been visited, upon which the function ends.

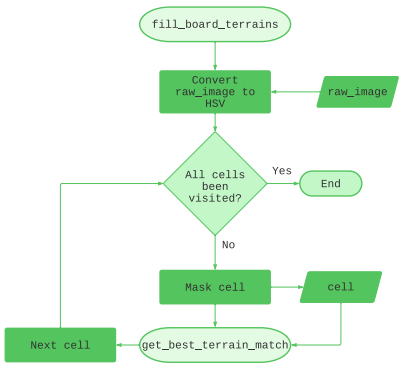

Figure 2: Flowchart filling the board with matching terrain.

0.4 Get Best Terrain Match

The function here seen in Figure 4 was used in section 0.3 and starts off with an if statement checking if the amount of thresholded pixels never exceed 2000, in which the cell would just be set as undefined. After that it goes through a match statement, similar to a switch statement it will match the amount of thresholded pixels, checking which case has the highest amount of white pixels after the threshold has been made. The threshold is done with OpenCV's inRange function which takes in a lower and upper limit for thresholding an image. Thresholding values have been found using histograms of the different cells.

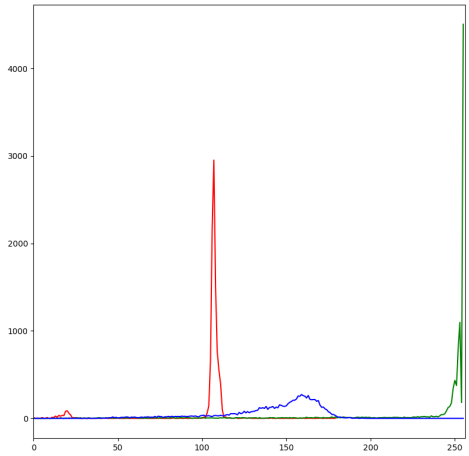

Figure 3: Histogram of an ocean tile, in the Hue-Saturation-Value colorspace.

If one of the cases are matched, it will then return that terrain type which is used to create an array which signifies the entire board and the different terrains.

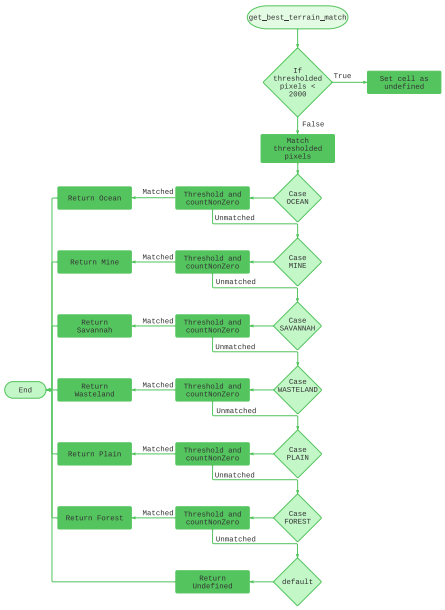

Figure 4: Flowchart that finds the best matching terrain type.

0.5 Visit Node

For the flowchart seen in Figure 5 the function here will first check if grouped_terrain is equal to None, which it only is if it is a new terrain type, in which it will create a new array for that terrain group.

Then it continues on to check if the current cell is the same terrain type as current terrain where if it is not the same it will just end, and if the terrain type is the same, it will append the current cell to the terrain group array from before, remove said cell from the unvisited cells array and append it to the visited array instead.

It will then go through all the adjacent cells, up, down, left & right to check if these cells are the same terrain type as the current cell, if the adjacent terrain type is the same it will go through the whole visit_node function again, but with the adjacent cell instead, and if the adjacent cells do not match, the function will end.

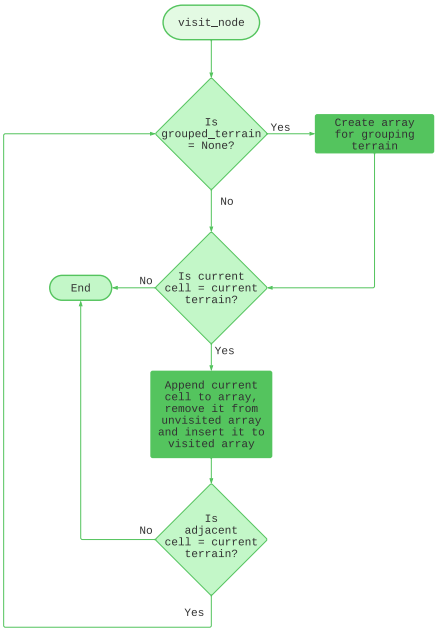

Figure 5: Flowchart of visit node to find adjacent cells.

0.6 Print Board Points

Excluding the function in section 0.7which is an imported function from a different script, this is the last function in the program and can be viewed in Figure 6.

The function starts off with checking if it has been through the list with all the connected terrain groups where if it has not it will then count all the crowns in a said terrain group and multiply this amount with the amount of cells in the terrain group array to calculate the total points adding up the sum of points each time it goes through a terrain group.

When it has been through all these connected terrain groups it will then take the sum variable that was made before and print it as the toal amount of points across the board and then end.

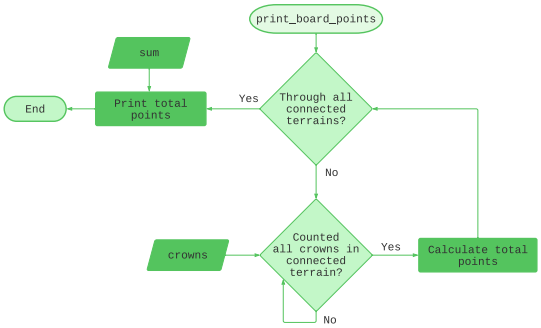

Figure 6: Flowchart of the calculate and print function.

0.7 Get Crowns

The get_crowns function here as mentioned in an earlier section is imported from crown_detection.py which was made as a seperate script to better organize the project and the flowchart is seen in Figure 7.

To start off with, we blur the raw_image that was defined from the start and also using a gaussian filter defined in crown_detection.py which makes the variable blur_image which is then sharpened with a 3x3 kernel thus creating a sharpened image which is the image in which crowns will be found.

An array is then created housing a close up cutout of the crown which is then rotated by 90 degrees until there is a crown image for each rotation in an array with these "templates". Then an array for all possible locations is created. To find these locations it goes through a for loop that checks if all the different orientations of the crown have been tested on the sharpened image from before thus finding the locations and inserting them in the locations array. When all the orientations have been tested it will then make a grouping of all the locations to make it more precise since the locations array is a bit of a mess until they are grouped up in squares made up of the crown image width's length.

Then it will outline the possible crowns with squares with the same dimensions as mentioned before and count the amount of crowns that are found in each cell, creating an array with each cell and their crown count.

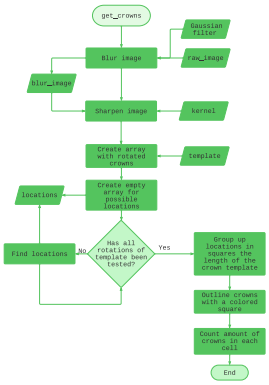

Figure 7: Flowchart of how the crowns are found.

0.8 Results

The first 10 images of the given dataset for kingdomino will be used to determine the performance of the image detection and classification algorithms used in this project.

The crown detection with rotated templatematching was able to detect 98 out of 98 crowns on the first 10 images. These are also some of the more ideal images to make template matching, since the images are well lighted and in good quality. On image, "14.jpg", a crown is not being detected and if the matchthresholding is set at a lower rate, non-crowns are being detected also.

As for terrain classification, on the 10 first images the algorithms were able to classify 197 out of 250 tiles correctly. This gives the algorithm a 78.8% successrate. However the dataset used, is small and the images are more ideal than the later images. Therefore it is expected to have lower successrate with other lower quality images. Most of the errors can be categorized into 3 categories. The starting tile have not been accounted for yet in this iteration of the algorithm and the empty tiles were challenging to detect. Furthermore the threshold for the Savannah (yellow) tiles were poorly chosen, which gave it worse performance as well as the combination of some tile-types and houses.

To improve the performance of the algorithms, a more in depth analysis of the training dataset could have been conducted. More precise thresholding values, based on more terrain tiles, could potentially have improved the classification of the terrains. Additionally, some edge detection and BLOB analysis on the images for crowns could potentially have found the blurrier crowns. However, an improved image acquisition, with more consistent lighting and higher quality images, would have made the algorithms more consistent.

0.9 Appendix

Source code and images used in the report: Link to GitHub