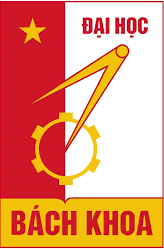
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

--SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING--

**---**🙠🕮🙢**---**



**REPORT FOR PROJECT 4:** Floodfill algorithm

**Course name:** Data structures and Algorithms

**Class:** 136401

**Group:** 4

**Members:** Nguyễn Anh Duy – 20210274 Phùng Minh Chiến – 20213565 Nguyễn Danh Huy – 20213571 Nguyễn Đức Phúc Hoàng –20210385

**Table of work resume:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Name** | **Student ID and email** | **Work assignment** | **Completion** |
| 1 | **Nguyễn Anh Duy** | 20210274 –[duy.na210274@sis.hust.edu.vn](mailto:duy.na210274@sis.hust.edu.vn) | Report | 100% |
| 2 | **Nguyễn Danh Huy** | 20213571 –huy.nd213571@sis.hust.edu.vn | Handle mouse event + load image by Opencv | 100% |
| 3 | **Nguyễn Đức Phúc Hoàng** | 20210385 –hoang.ndp210385@sis.hust.edu.vn | Design Interface | 100% |
| 4 | **Phùng Minh Chiến** | 20213565 – chien.pm213565@sis.hust.edu.vn | Implement algorithm | 100% |

# Problem/system introduction

# \* How scanline floodfill algorithm using a stack works in our project

The floodfill algorithm starts from a pixel, it replaces itself with a new color and it does the same thing with its neighboring pixels, and so on until it reaches borders or pixels with a different color.

The algorithm uses the stack to keep track of the spans (regions) of connected pixels that need to be processed. The scan function helps in finding the connected spans in a given line. The valid function checks if a given pixel is within the image bounds and has the original color.

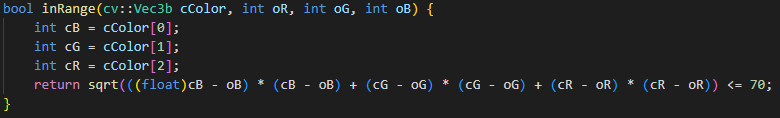
The scan function is used to scan a single horizontal line of pixels in the image and find the start and end points of the connected region on that line. The valid function is used to check if a given pixel is valid to be processed.

The time complexity of a scanline flood fill algorithm using a stack can be considered to be O(n), where n is the number of pixels in the image. The space complexity of the algorithm is O(m), where m is the maximum number of pixels that can be added to the stack at any given time.

In the worst case, m can be proportional to the size of the image, in which case the space complexity would be O(n). However, in practice, m is much smaller than n, so the space complexity is often considered to be linear with respect to m, rather than n.

# System analysis and activity description

* **System analysis**
  + To implement floodfill algorithm for images, we have to access each pixel in the image to extract the color for the purpose of algorithm processing 🡪 Opencv library is used for accessing the pixel matrix of an image
  + On the other hand, we have to get the pixel which the mouse is tracking on when we click on the image, and we also need to choose a color to change after the algorithm 🡪 We use Qt for detecting mouse clicks and allow user to choose any color that constructed by 3 indexes: oR, oG, oB
  + Two colors can be different about the r, g, b indexes but seem to be the same with each other, so we create inRange function to optimize the algorithm.

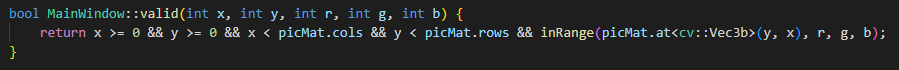


* + This inRange function takes in a color represented as a cv::Vec3b object cColor and three integer values oR, oG, and oB representing a target color, and returns a boolean value indicating whether the input color is within the specified range of the target color.
  + In this program we use span filling technique, as well as the scanline floodfill algorithm using stack instead of recursion. The basic idea behind span filling is to scan the image from the beginning pixel (seed), and for each scan line, find the start and end pixels of the region to be filled. The region between the start and end points is then filled with the desired color

## Code description

## Valid check

First, we must check if a given pixel at coordinates (x, y) is valid to be processed by a floodfill algorithm. The function returns a boolean value indicating whether the pixel is valid or not.

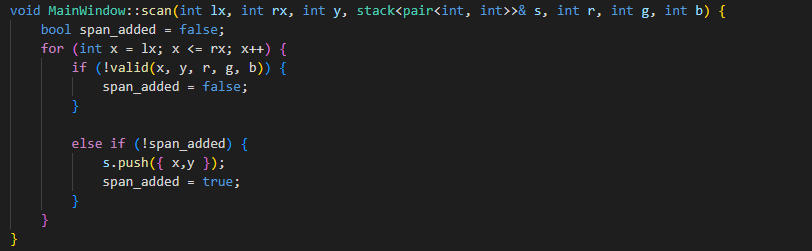


The first check x >= 0 && y >= 0 && x < picMat.cols && y < picMat.rows verifies that the pixel is within the bounds of the image, i.e., its x and y coordinates are non-negative and within the dimensions of the image represented by the picMat matrix

The final check inRange(picMat.at<cv::Vec3b>(y, x), r, g, b) verifies that the color of the pixel at (x, y) is equal to the target color (r, g, b), then returns a boolean value indicating whether the color of the pixel falls within the specified range.

Overall, the valid function returns true if the pixel is within the bounds of the image and has the same color as the seed color. Otherwise, it returns false.

## Scan with stack structure



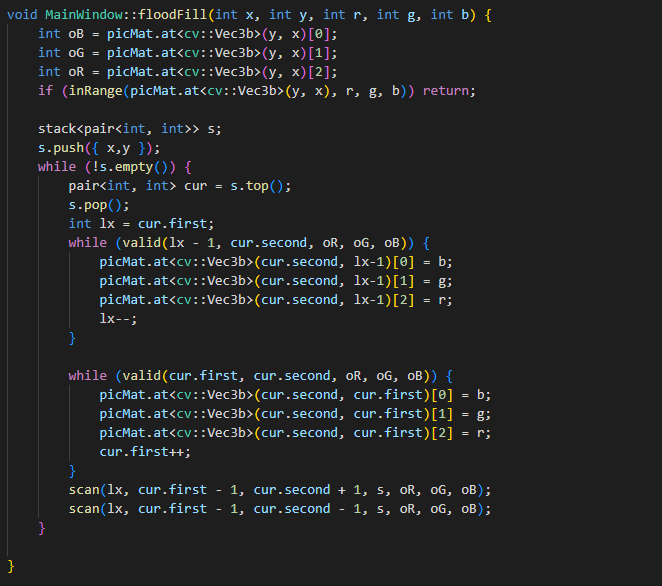
This part uses the stack data structure to keep track of the pixels that need to be processed. The scan function is used to scan a single horizontal line of pixels in the image, starting from lx and ending at rx. The current y-coordinate is y.

The function checks the validity of each pixel on the scan line. If a pixel is not valid (i.e., its color does not match the target color (r, g, b)), then the span\_added flag is set to false.

If a valid pixel is found and the span\_added flag is false, the function pushes the current x and y coordinates onto the stack (s.push({x, y})). This indicates that the current pixel is part of a connected region of the target color, and its neighbors should also be processed. The span\_added flag is then set to true.

The scan function does not actually color the pixels, but instead uses the stack to keep track of the pixels that need to be processed in a later step.

## Scanline floodfill algorithm

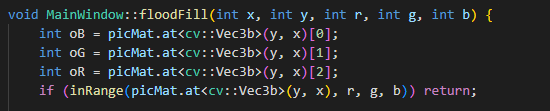


Given an initial point (x, y), it finds all the connected pixels that are similar to the color of the initial pixel and changes their color to the desired color (r, g, b).

The algorithm works as follows:

- The original color of the initial point (x, y) is obtained and stored in variables oR, oG, oB.

The function inRange is called to check if the color of the initial pixel is already the desired color. If it is, the function returns immediately.



- A stack is initialized and the initial point is pushed onto the stack.

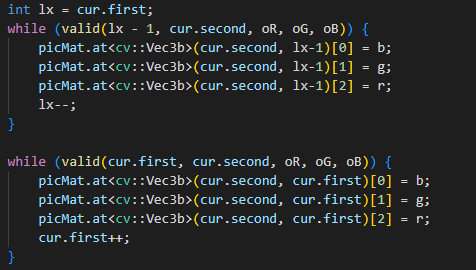


- The algorithm enters a while loop, which continues until the stack is empty.

The top element of the stack is popped and its x and y values are stored in variables cur.first and cur.second.



- A while loop starts from the current x value and moves to the left until it reaches the leftmost edge of the connected region. For each pixel, the color is changed to the desired color and the x value is decremented. A second while loop starts from the leftmost edge of the connected region and moves to the right until it reaches the rightmost edge. For each pixel, the color is changed to the desired color and the x value is incremented.

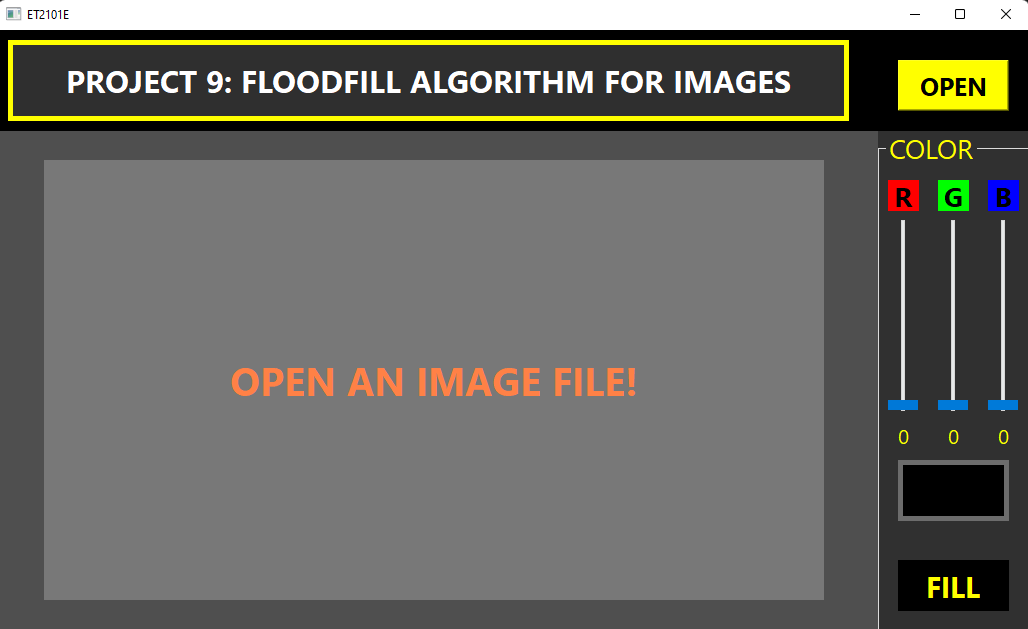


- The scan function is called twice to examine the two adjacent lines above and below the current line. The scan function pushes a valid pixel’s position values of any connected spans in these lines onto the stack.

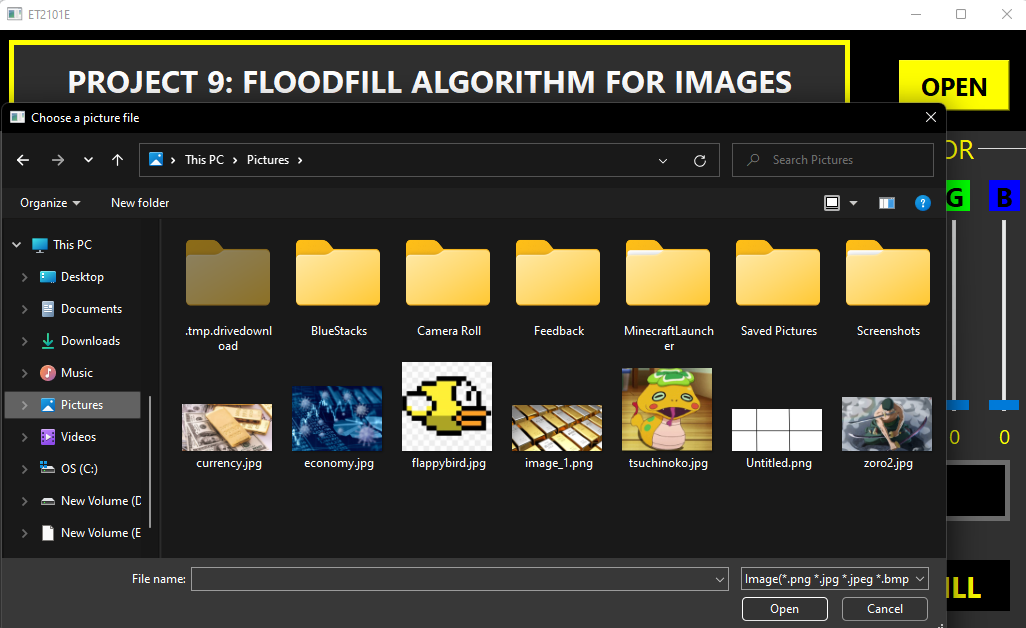


- The while loop continues until the stack is empty, and all connected pixels have been processed.

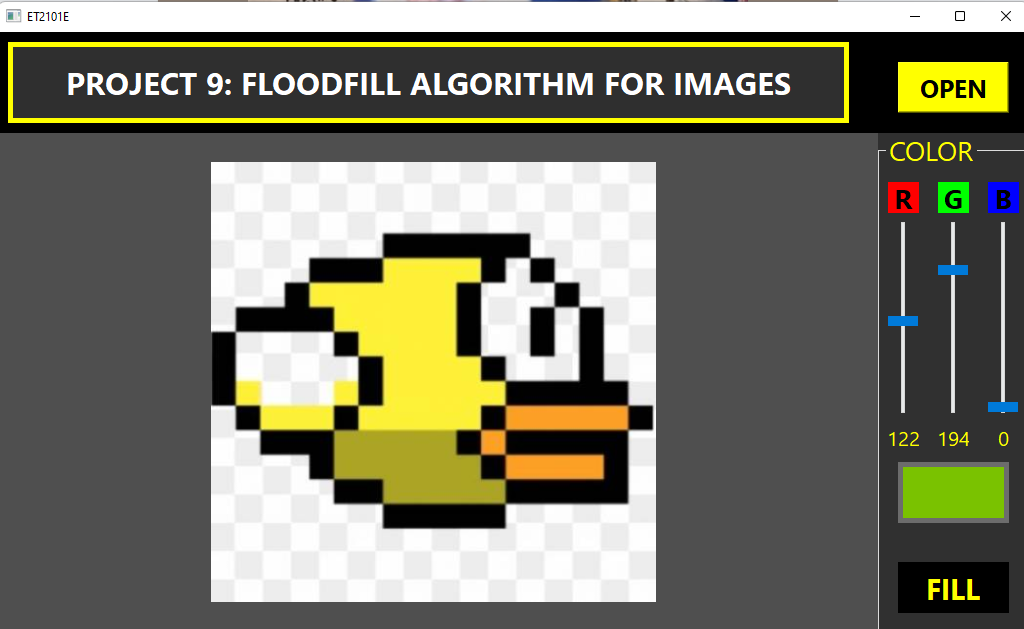
## Activity description



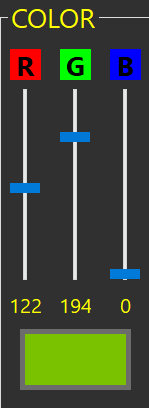
* + Click “open” button to choose an image from your device



* + When an image is chosen, it will be scaled to the main window by our algorithms



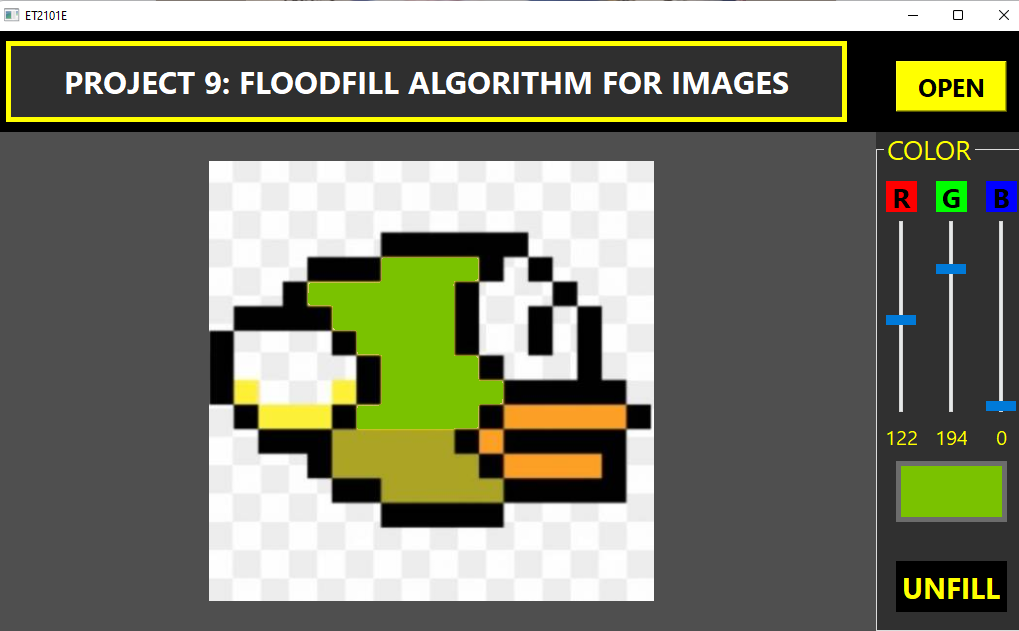
* + Click “fill” button to change the mouse from normal to brush and vice virsa



* + You can change 3 pull bars “R”, “G”, “B” to choose your favourite color for coloring



* + Finally, when the mouse change into ethe brush-shaped, you can click on the image where you want to color and see the result on main window



# Self-assessment of results, limitation

# There are some cases we cannot color the area if its color is not too different from the desired color

# Cannot undo;

# There are a few pixels in chosen area may not be colored because of  the differences of r, g, b indexes and interpolation of pixel matrix when it is scaled from the original image