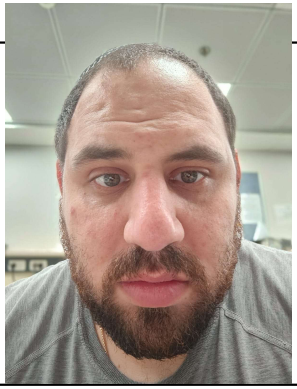
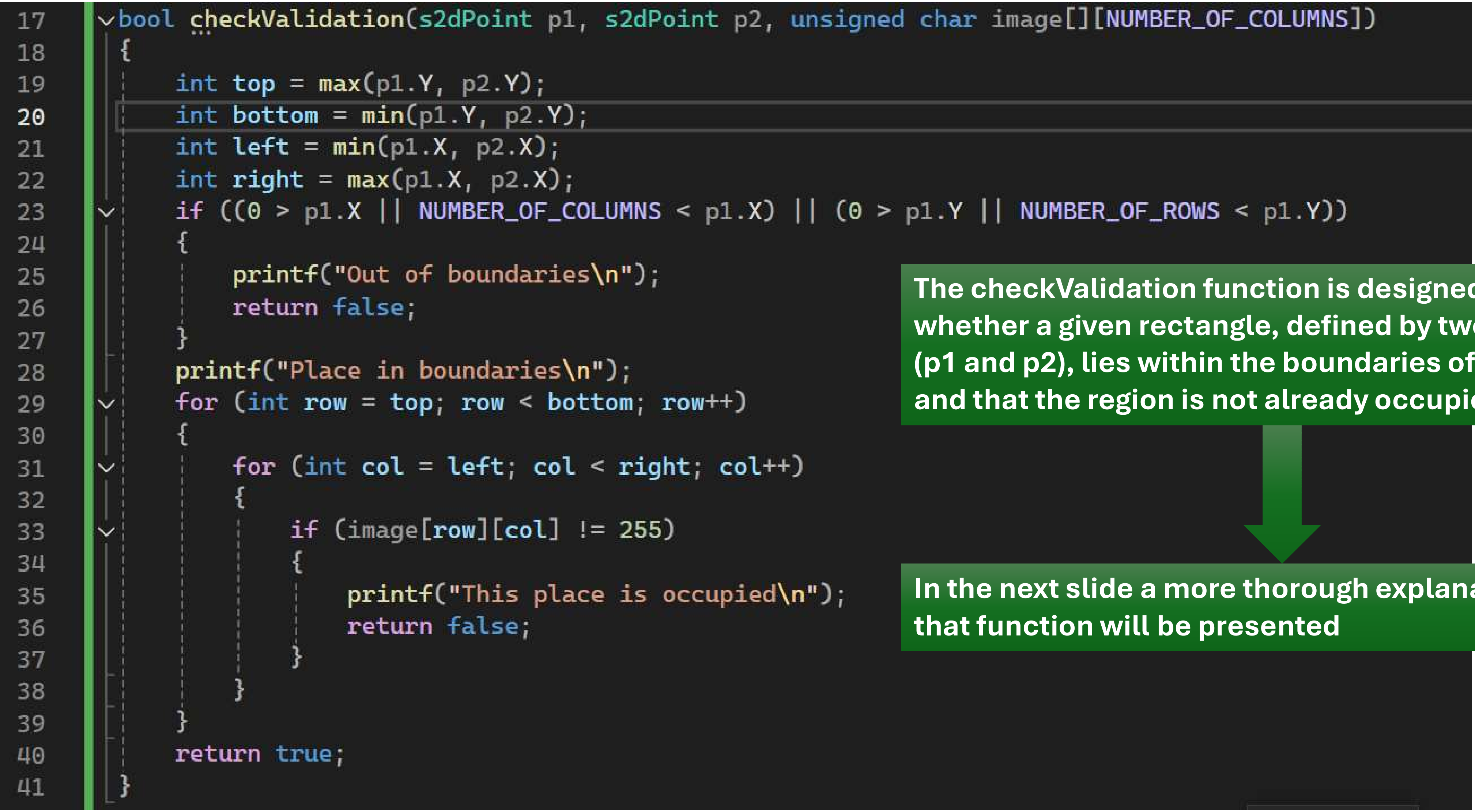
Course: Image Processing 31651
Assignment #11
Synthetic Image Creation (Part 1)

	ID (4 last digits)	Shorten Name	Photo of the student
Student #1	**1950**	shienfeld	
Student #2	**2210**	pony	
Student #3	**7939**	akimov	

Part 1 of Introduction to 11.1 – apply “checkValidation” function



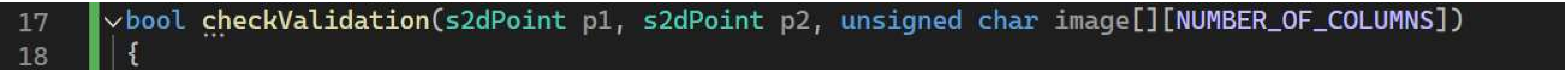
```c
bool checkValidation(s2dPoint p1, s2dPoint p2, unsigned char image[][NUMBER_OF_COLUMNS])
{
    int top = max(p1.Y, p2.Y);
    int bottom = min(p1.Y, p2.Y);
    int left = min(p1.X, p2.X);
    int right = max(p1.X, p2.X);
    if ((0 > p1.X || NUMBER_OF_COLUMNS < p1.X) || (0 > p1.Y || NUMBER_OF_ROWS < p1.Y))
    {
        printf("Out of boundaries\n");
        return false;
    }
    printf("Place in boundaries\n");
    for (int row = top; row < bottom; row++)
    {
        for (int col = left; col < right; col++)
        {
            if (image[row][col] != 255)
            {
                printf("This place is occupied\n");
                return false;
            }
        }
    }
    return true;
}
```

The checkValidation function is designed to verify whether a given rectangle, defined by two points (p1 and p2), lies within the boundaries of an image and that the region is not already occupied.

In the next slide a more thorough explanation about that function will be presented

Part 2 of Introduction to 11.1 – apply “checkValidation” function

In the next few slides we will present the “checkValidation” function a bit deeper



```
bool checkValidation(s2dPoint p1, s2dPoint p2, unsigned char image[][NUMBER_OF_COLUMNS])
{
```

First step- Parameters of the function:

s2dPoint p1 → is the first point defining one corner of the rectangle.

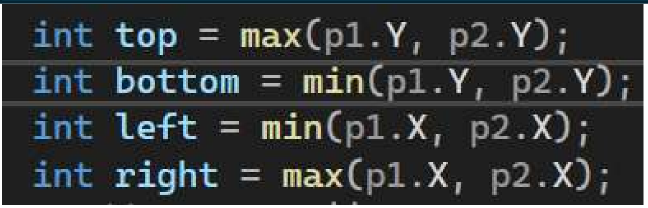
s2dPoint p2 → is the second point defining the opposite corner of the rectangle.

unsigned char image[][NUMBER_OF_COLUMNS] → is A 2D array representing the image.

The function returns a Boolean output : true if the rectangle is valid (within boundaries and not occupied), otherwise false.

```
return true;
```

Second step - define Rectangle Boundaries:

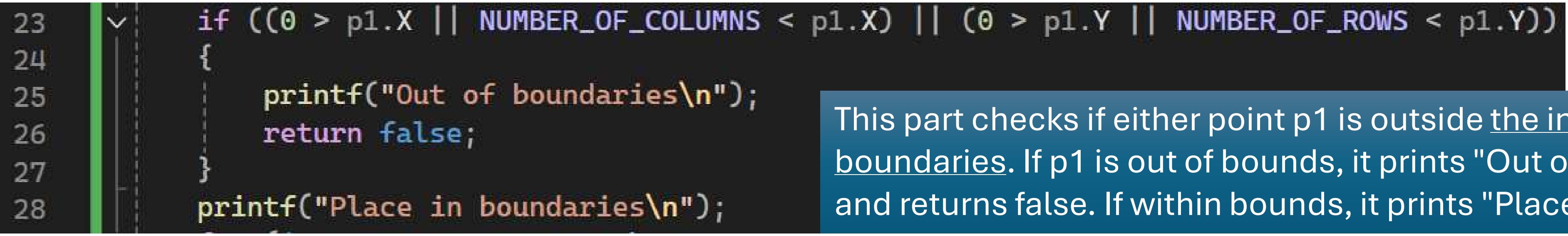


```
int top = max(p1.Y, p2.Y);
int bottom = min(p1.Y, p2.Y);
int left = min(p1.X, p2.X);
int right = max(p1.X, p2.X);
```

top: The maximum Y-coordinate between p1 and p2.
bottom: The minimum Y-coordinate between p1 and p2.
left: The minimum X-coordinate between p1 and p2.
right: The maximum X-coordinate between p1 and p2.

Part 3 of Introduction to 11.1 – apply “checkValidation” function

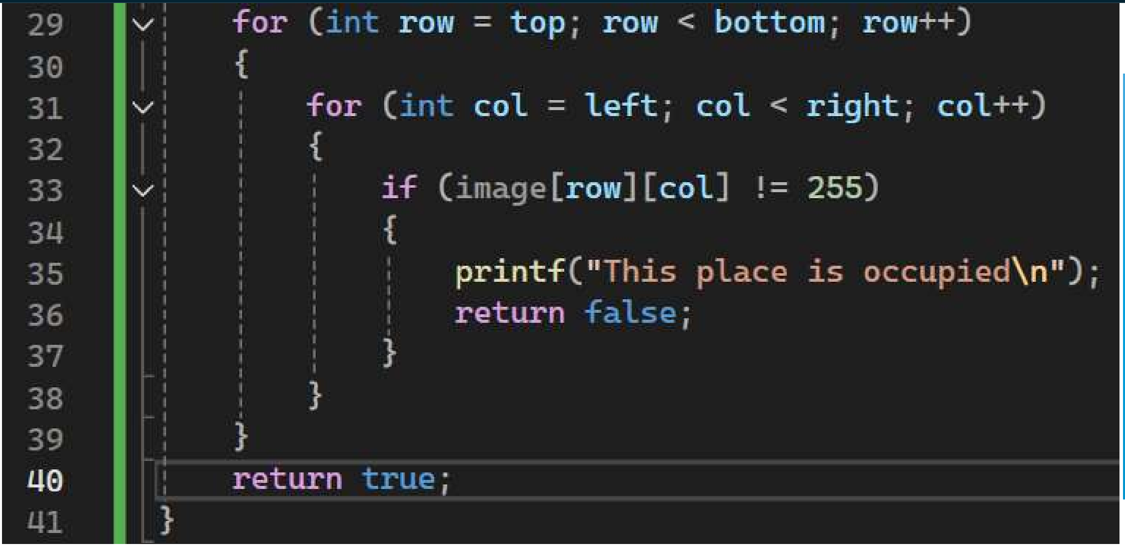
third step – check if a defined point is in or out of bounds:



```
    if ((0 > p1.X || NUMBER_OF_COLUMNS < p1.X) || (0 > p1.Y || NUMBER_OF_ROWS < p1.Y))
    {
        printf("Out of boundaries\n");
        return false;
    }
    printf("Place in boundaries\n");
```

This part checks if either point p1 is outside the image boundaries. If p1 is out of bounds, it prints "Out of boundaries" and returns false. If within bounds, it prints "Place in boundaries".

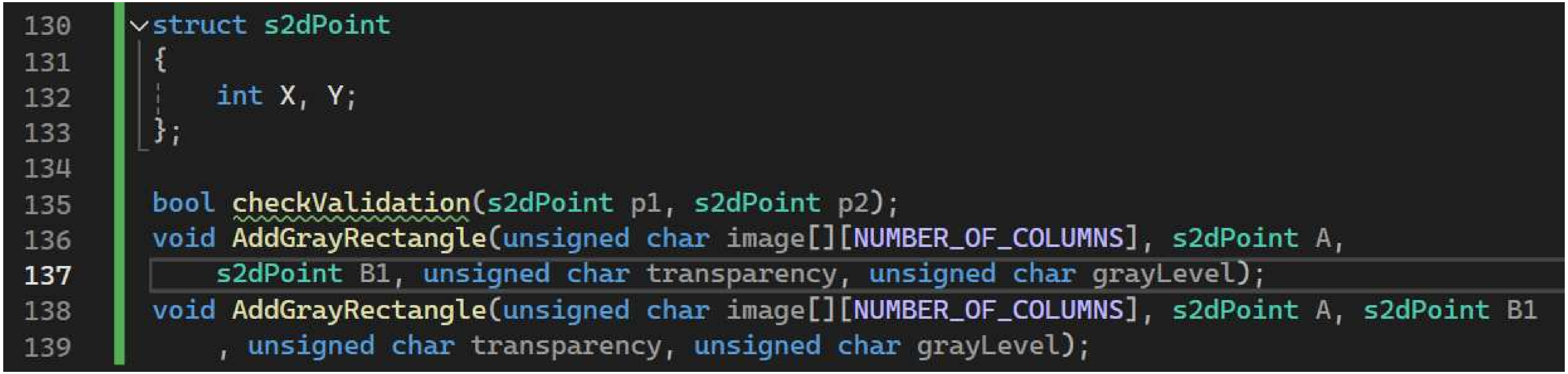
fourth step – checking occupancy of pixels within the rectangle:



```
    for (int row = top; row < bottom; row++)
    {
        for (int col = left; col < right; col++)
        {
            if (image[row][col] != 255)
            {
                printf("This place is occupied\n");
                return false;
            }
        }
    }
    return true;
}
```

This loop iterates over the region defined by the rectangle:
Outer loop: Iterates from top to bottom (Y-coordinates).
Inner loop: Iterates from left to right (X-coordinates).
It checks if any pixel within the rectangle is not equal to 255 (assuming 255 represents an unoccupied pixel in a grayscale image).
If it finds an occupied pixel, it prints "This place is occupied" and returns false.

Part 4 of Introduction to 11.1 – apply “s2dpoint” struct and the declarations of the relevant functions in the code



```
struct s2dPoint
{
    int X, Y;
};

bool checkValidation(s2dPoint p1, s2dPoint p2);
void AddGrayRectangle(unsigned char image[][NUMBER_OF_COLUMNS], s2dPoint A,
    s2dPoint B1, unsigned char transparency, unsigned char grayLevel);
void AddGrayRectangle(unsigned char image[][NUMBER_OF_COLUMNS], s2dPoint A, s2dPoint B1
    , unsigned char transparency, unsigned char grayLevel);
```

1) **The s2dPoint structure is defined at the header file to represent a 2D point with X and Y coordinates.**
2) **The s2dPoint structure’s Usage is to store the coordinates of points in the image processing functions.**

The AddGrayRectangle input parameters are:

a) unsigned char image[][NUMBER_OF_COLUMNS] → A 2D array representing the image.
b) s2dPoint A → The first corner point of the rectangle.
c) s2dPoint B1 → The opposite corner point of the rectangle.
d) unsigned char transparency → The transparency level for the rectangle.
e) unsigned char grayLevel → The gray level for the rectangle.

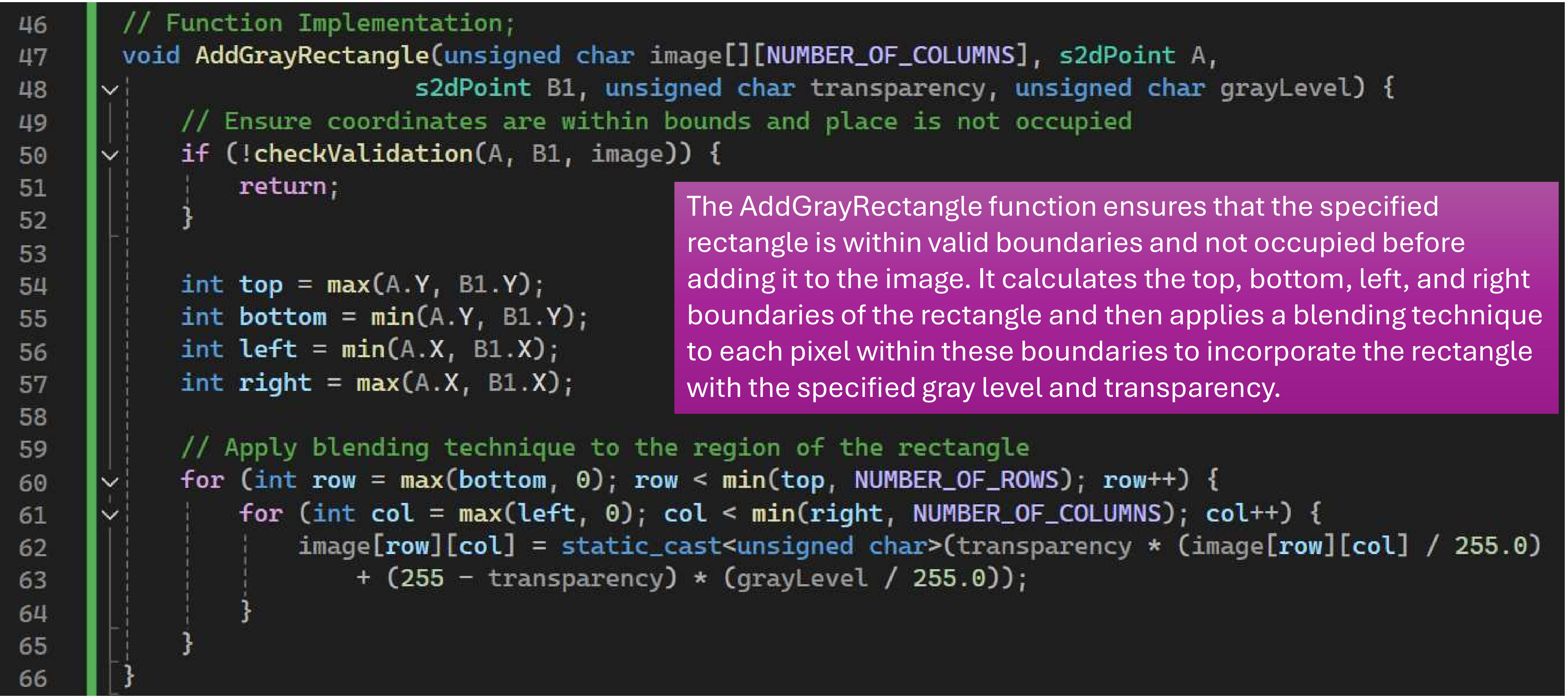
This function will add a gray rectangle to the image with the specified transparency and gray level, using the points A and B1 to define the rectangle's bounds.

11.1 Code of the function “AddGrayRectangle”

Numerical values of: NUMBER_OF_ROWS and NUMBER_OF COLUMNS
Defined in the file ImProcInPlainC.h

Pay attention that in the code that For each pixel, blend the new gray level using the formula:

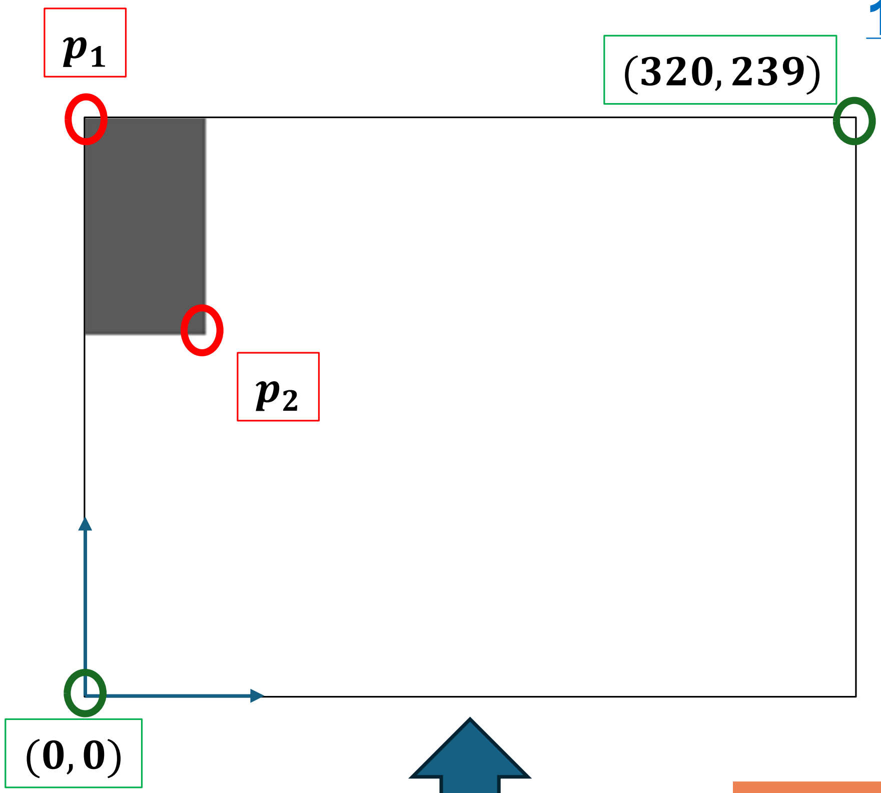
image[row][col] = static_cast<unsigned char>(transparency * (image[row][col] / 255.0) + (255 - transparency) * (grayLevel / 255.0));



```
// Function Implementation;
void AddGrayRectangle(unsigned char image[][NUMBER_OF_COLUMNS], s2dPoint A,
                    s2dPoint B1, unsigned char transparency, unsigned char grayLevel) {
    // Ensure coordinates are within bounds and place is not occupied
    if (!checkValidation(A, B1, image)) {
        return;
    }

    int top = max(A.Y, B1.Y);
    int bottom = min(A.Y, B1.Y);
    int left = min(A.X, B1.X);
    int right = max(A.X, B1.X);

    // Apply blending technique to the region of the rectangle
    for (int row = max(bottom, 0); row < min(top, NUMBER_OF_ROWS); row++) {
        for (int col = max(left, 0); col < min(right, NUMBER_OF_COLUMNS); col++) {
            image[row][col] = static_cast<unsigned char>(transparency * (image[row][col] / 255.0)
                + (255 - transparency) * (grayLevel / 255.0));
        }
    }
}
```

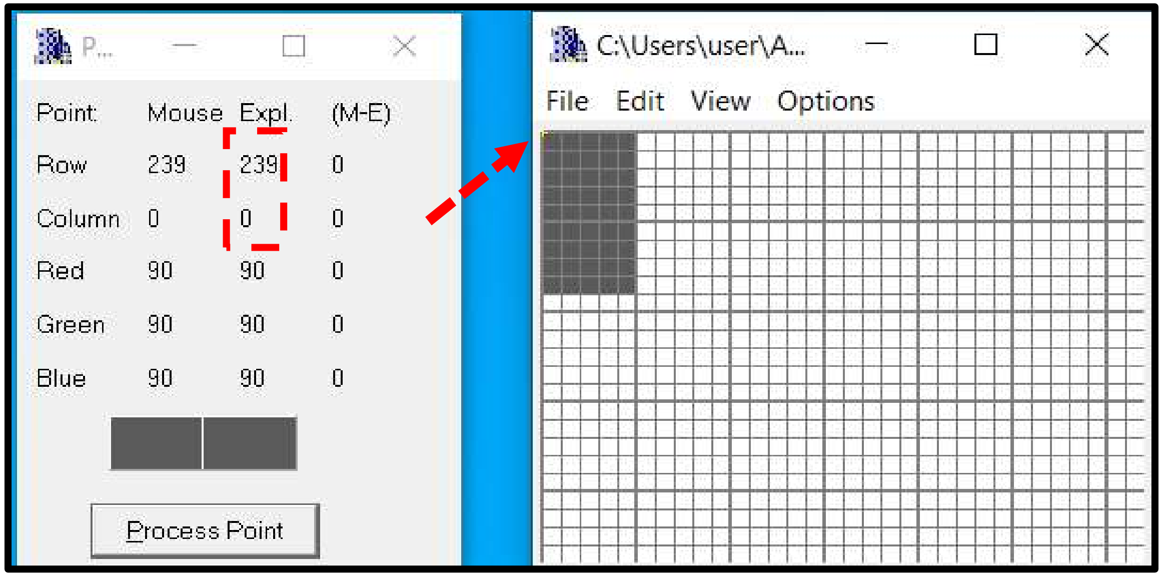
The AddGrayRectangle function ensures that the specified rectangle is within valid boundaries and not occupied before adding it to the image. It calculates the top, bottom, left, and right boundaries of the rectangle and then applies a blending technique to each pixel within these boundaries to incorporate the rectangle with the specified gray level and transparency.

11.2 List of situations tested – part 1

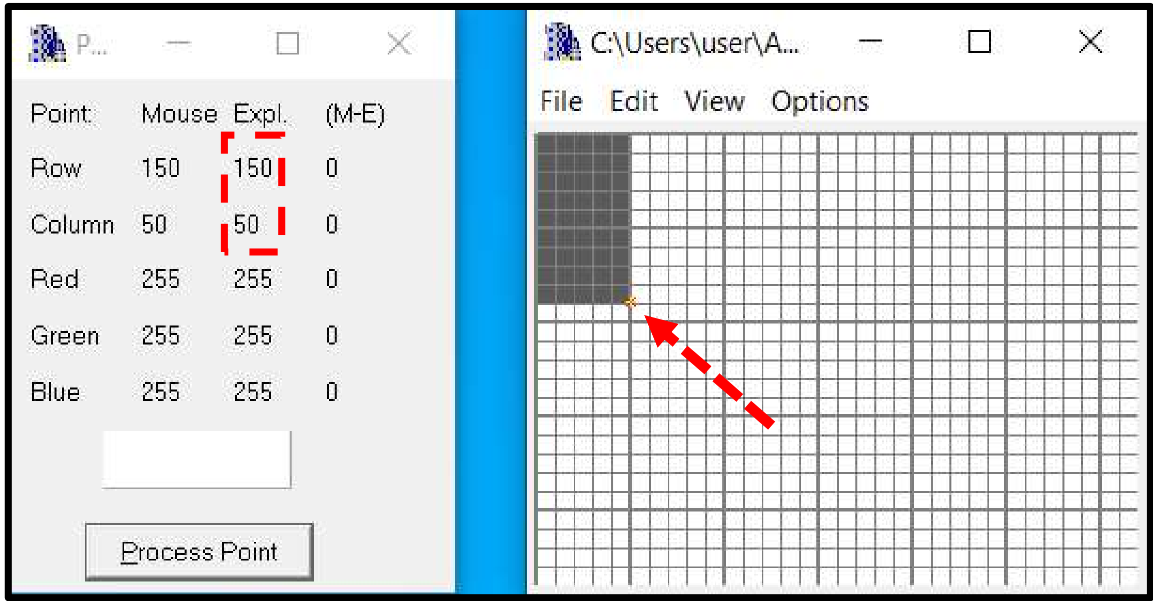


$$p_1 = (0,240)$$
$$p_2 = (50,150)$$

This test image will be listed as #1

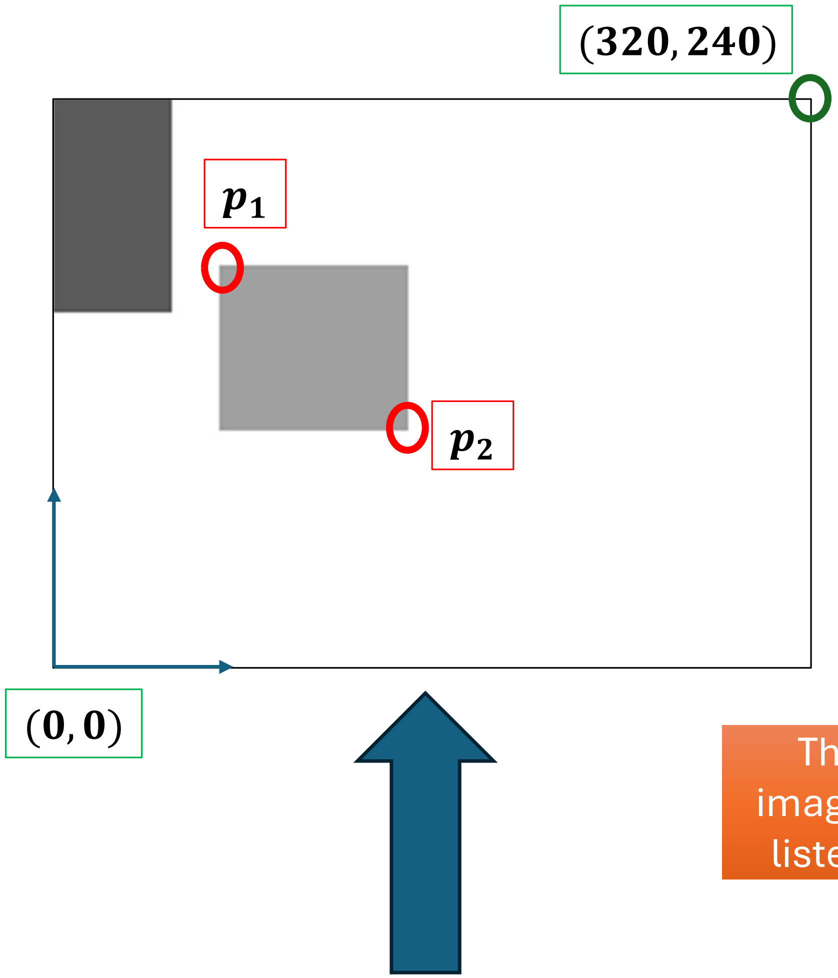


Point:	Mouse	Expl.	(M-E)
Row	239	239	0
Column	0	0	0
Red	90	90	0
Green	90	90	0
Blue	90	90	0



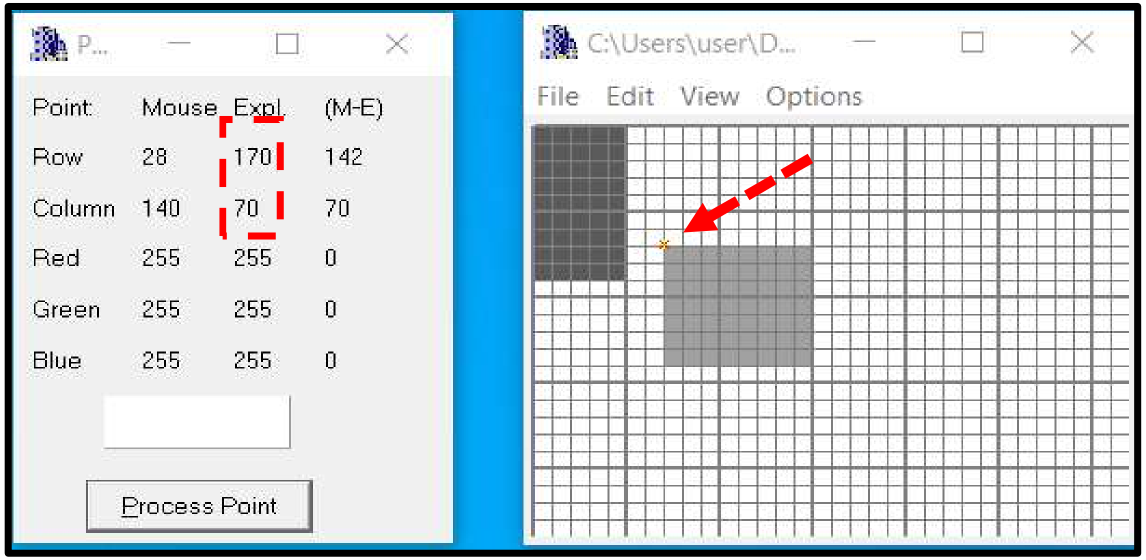
Point:	Mouse	Expl.	(M-E)
Row	150	150	0
Column	50	50	0
Red	255	255	0
Green	255	255	0
Blue	255	255	0

11.2 List of situations tested – part 2

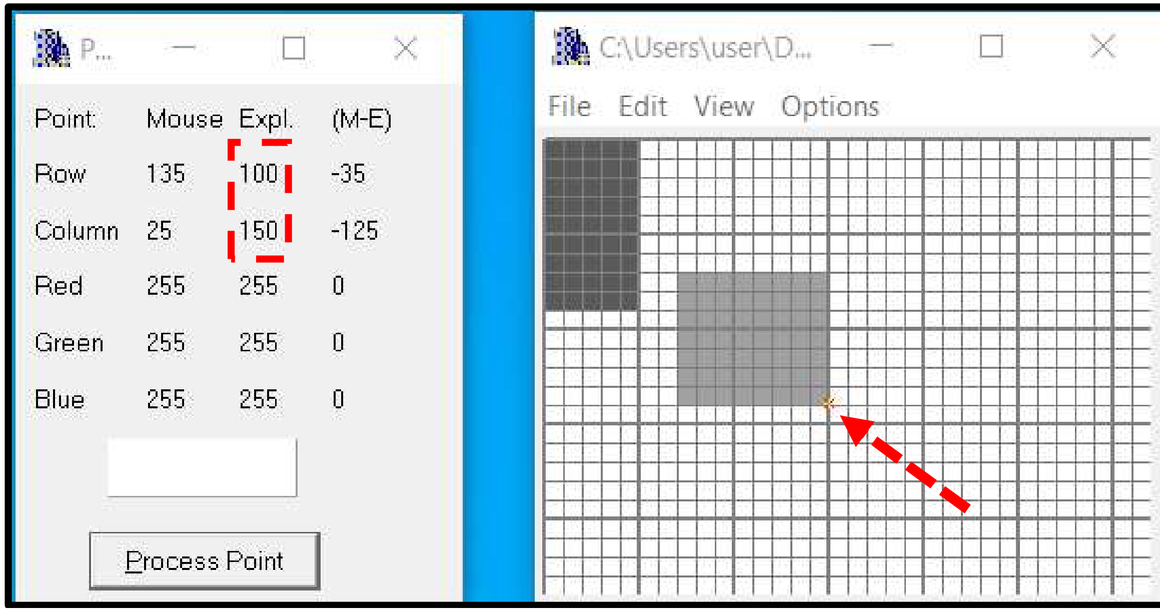


$p_1 = (70,170)$
$p_2 = (150,100)$

This test image will be listed as #2

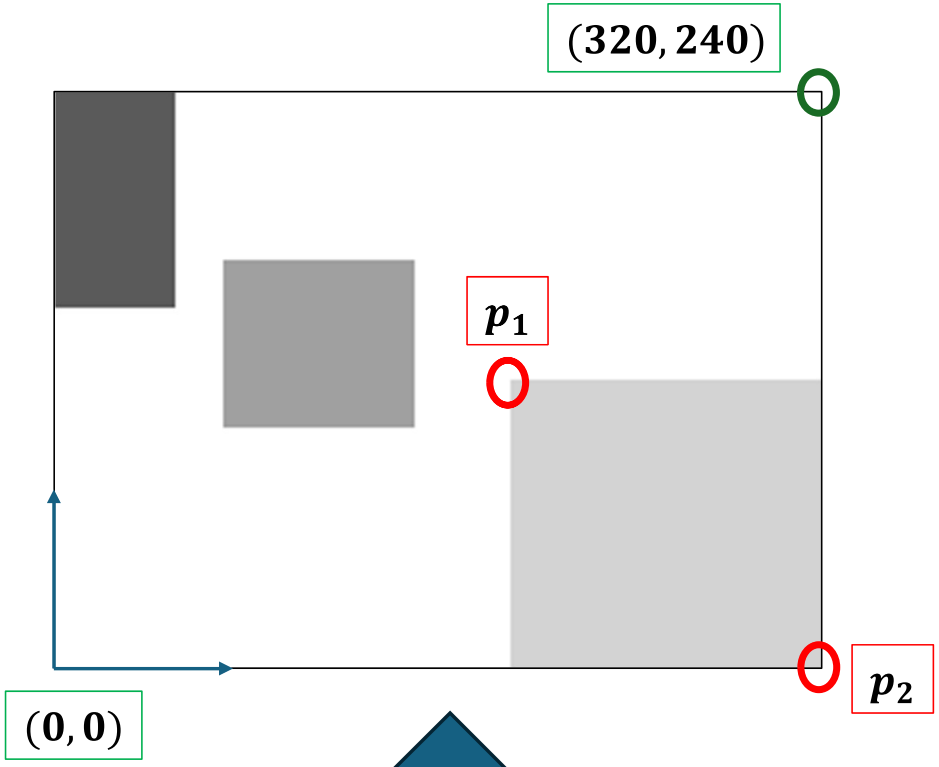


Point:	Mouse	Expl.	(M-E)
Row	28	170	142
Column	140	70	70
Red	255	255	0
Green	255	255	0
Blue	255	255	0



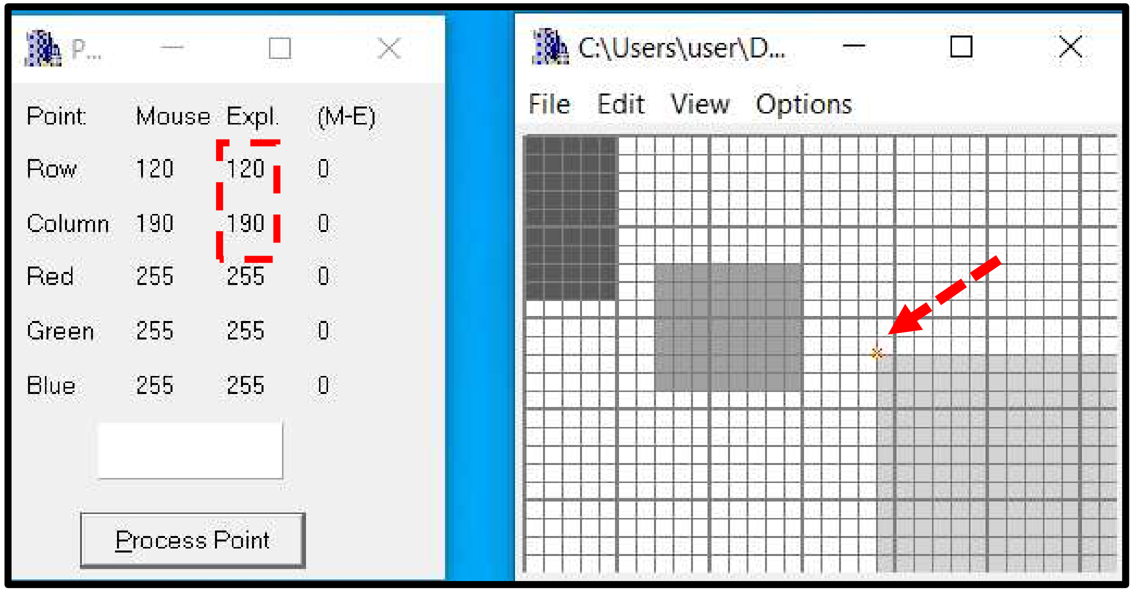
Point:	Mouse	Expl.	(M-E)
Row	135	100	-35
Column	25	150	-125
Red	255	255	0
Green	255	255	0
Blue	255	255	0

11.2 List of situations tested – part 3

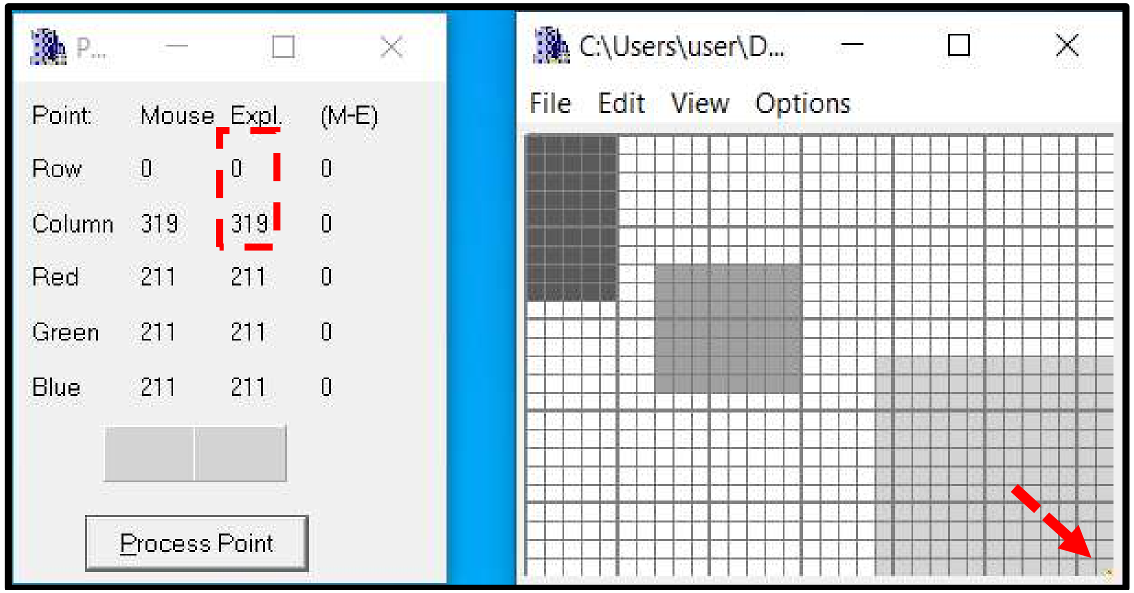


$p_1 = (190,120)$
$p_2 = (320,0)$

This test image will be listed as #3

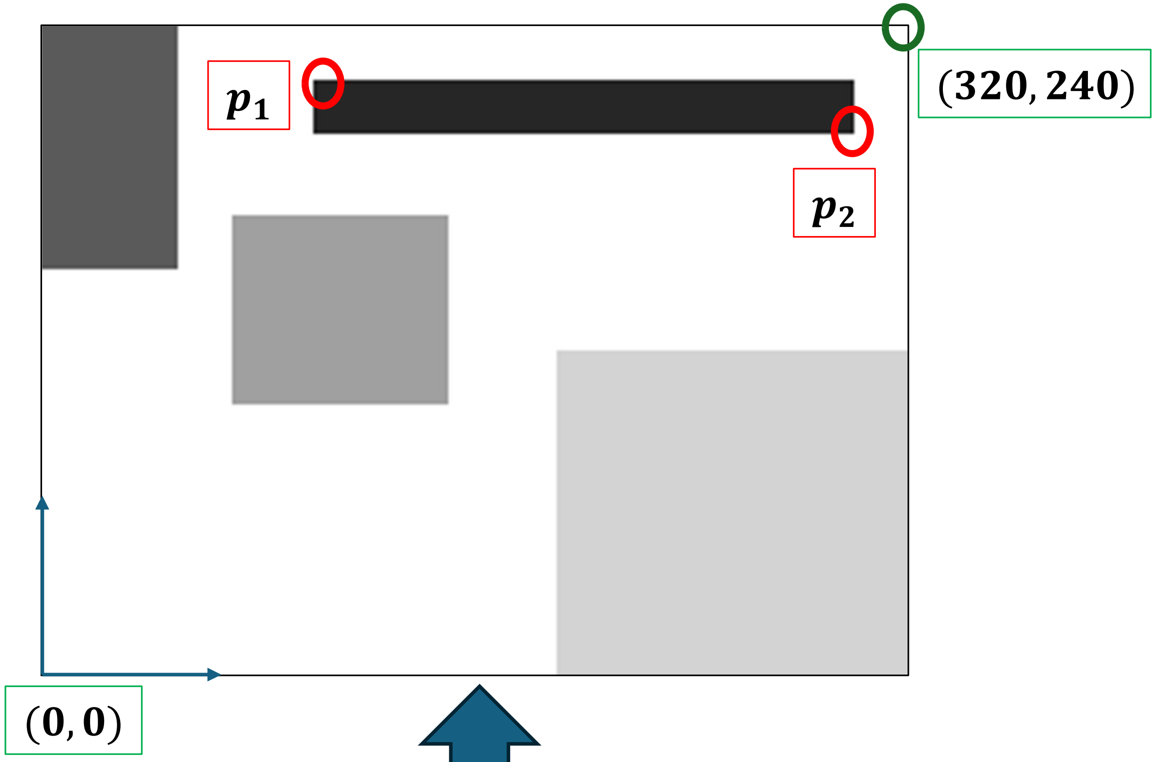


```
Point:   Mouse  Expl.  (M-E)
Row      120    120    0
Column   190    190    0
Red      255    255    0
Green    255    255    0
Blue     255    255    0
```



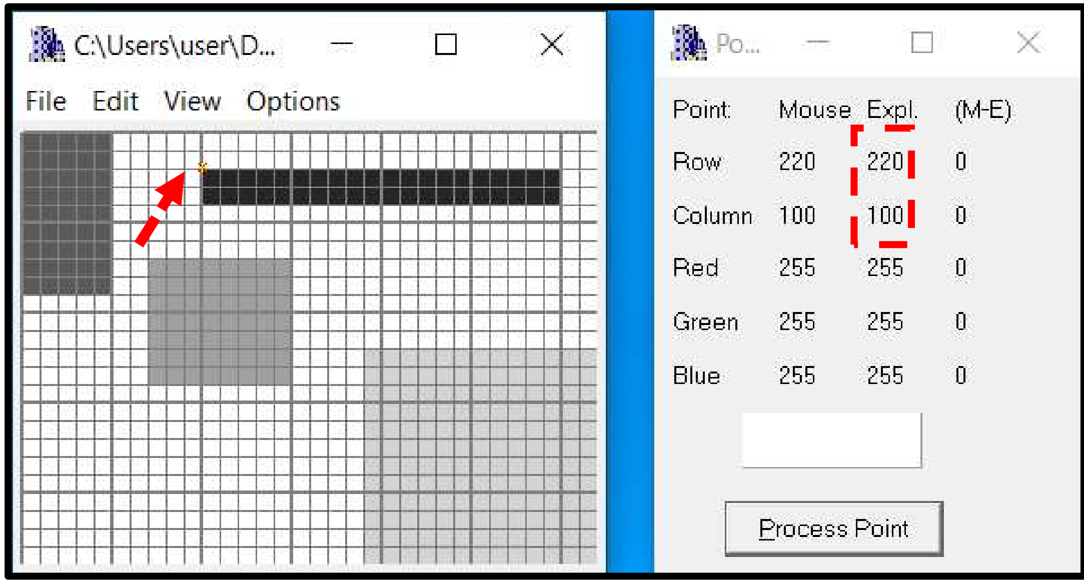
```
Point:   Mouse  Expl.  (M-E)
Row      0      0      0
Column   319    319    0
Red      211    211    0
Green    211    211    0
Blue     211    211    0
```

11.2 List of situations tested – part 4

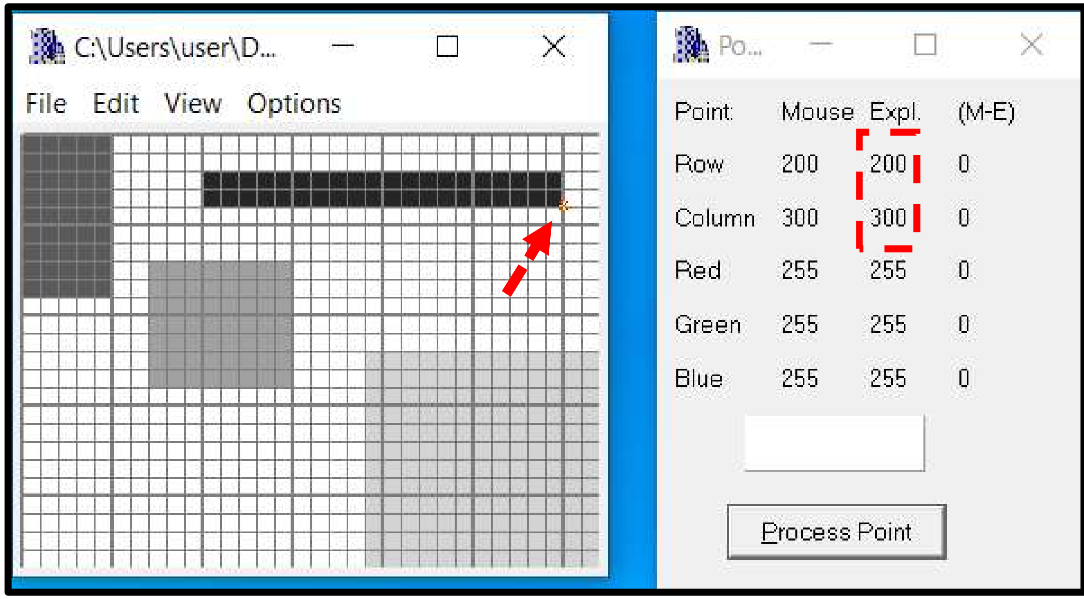


$$p_1 = (100,220)$$
$$p_2 = (300,200)$$

This test image will be listed as #4

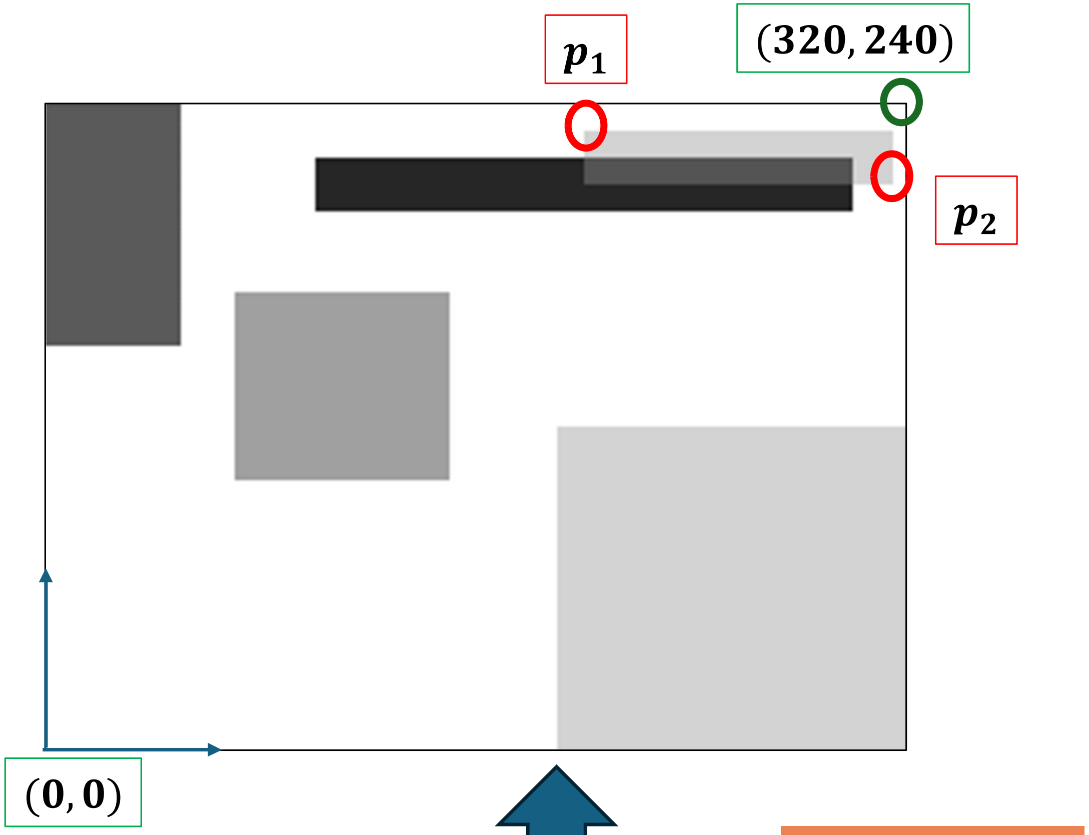


Point:	Mouse	Expl.	(M-E)
Row	220	220	0
Column	100	100	0
Red	255	255	0
Green	255	255	0
Blue	255	255	0



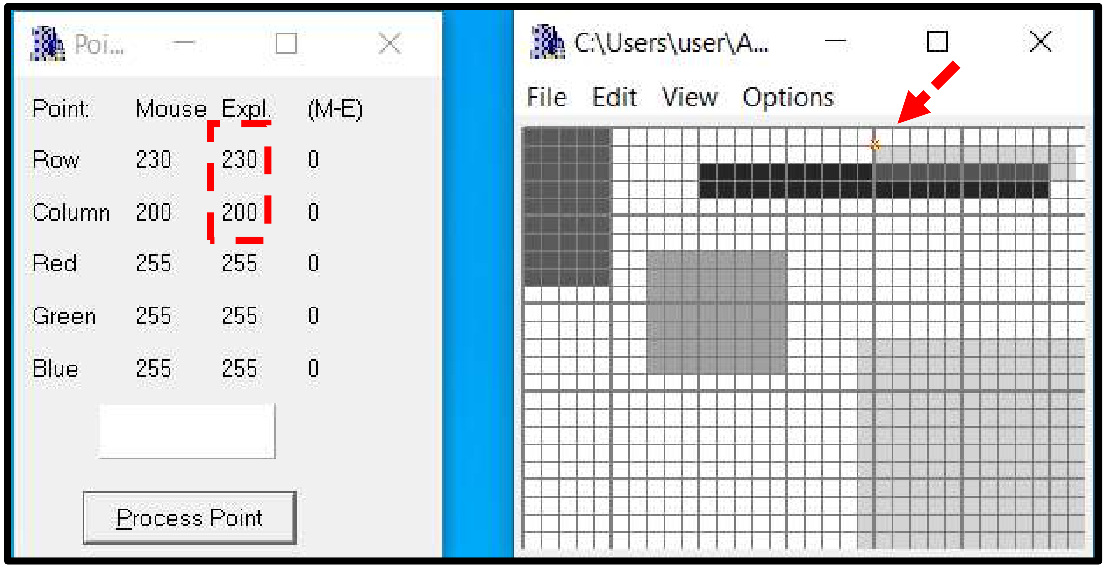
Point:	Mouse	Expl.	(M-E)
Row	200	200	0
Column	300	300	0
Red	255	255	0
Green	255	255	0
Blue	255	255	0

11.2 List of situations tested – part 5

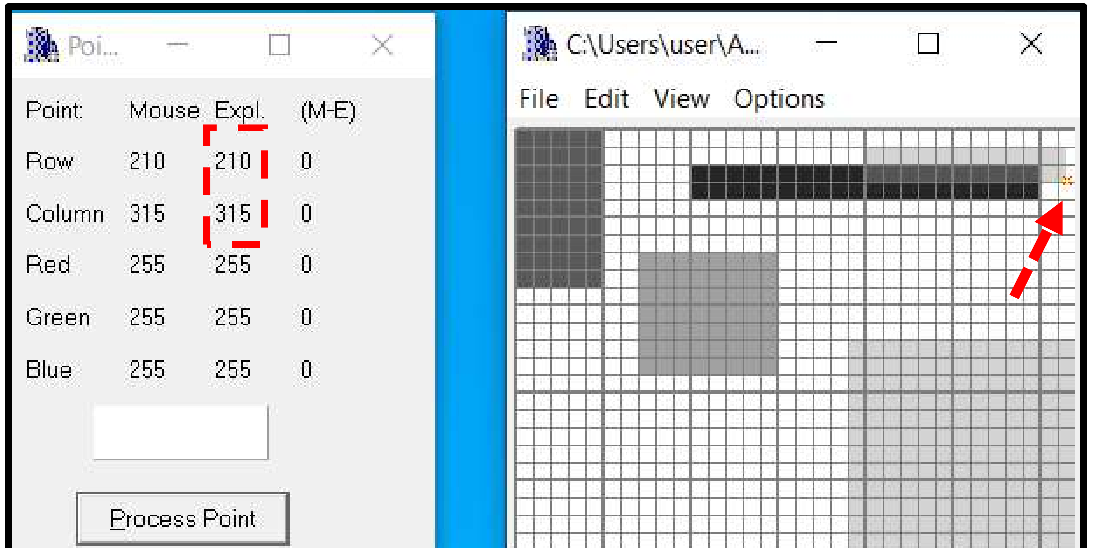


$p_1 = (200,230)$
$p_2 = (315,210)$

This test image will be listed as #5

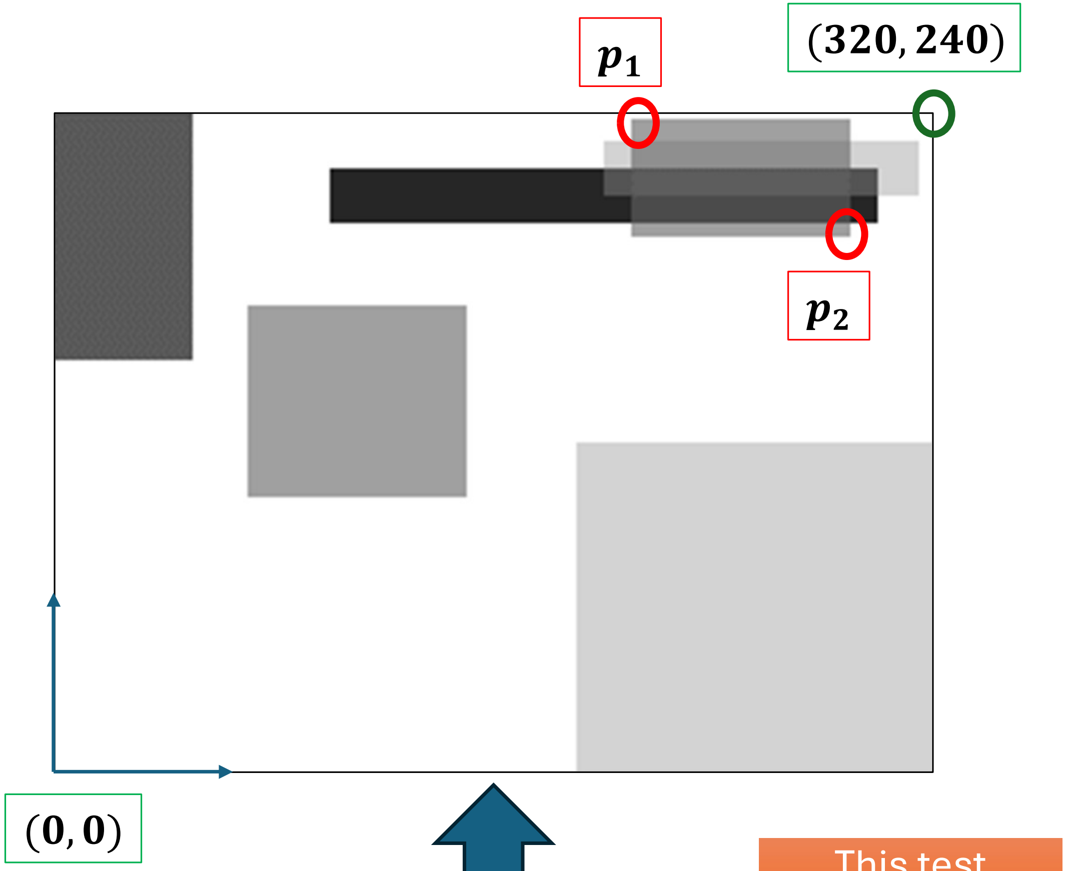


Point:	Mouse	Expl.	(M-E)
Row	230	230	0
Column	200	200	0
Red	255	255	0
Green	255	255	0
Blue	255	255	0



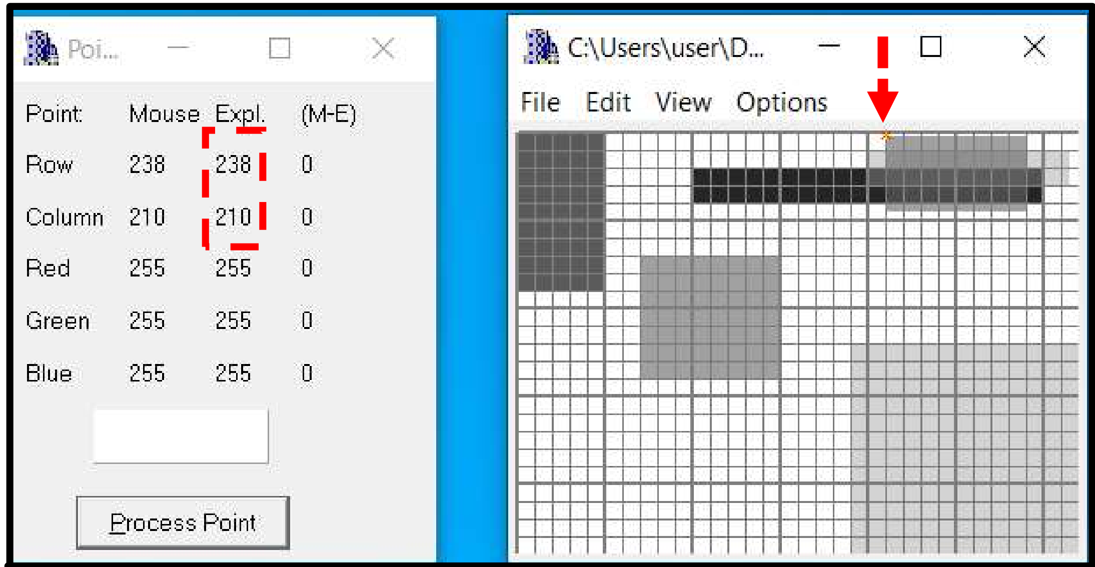
Point:	Mouse	Expl.	(M-E)
Row	210	210	0
Column	315	315	0
Red	255	255	0
Green	255	255	0
Blue	255	255	0

11.2 List of situations tested – part 6

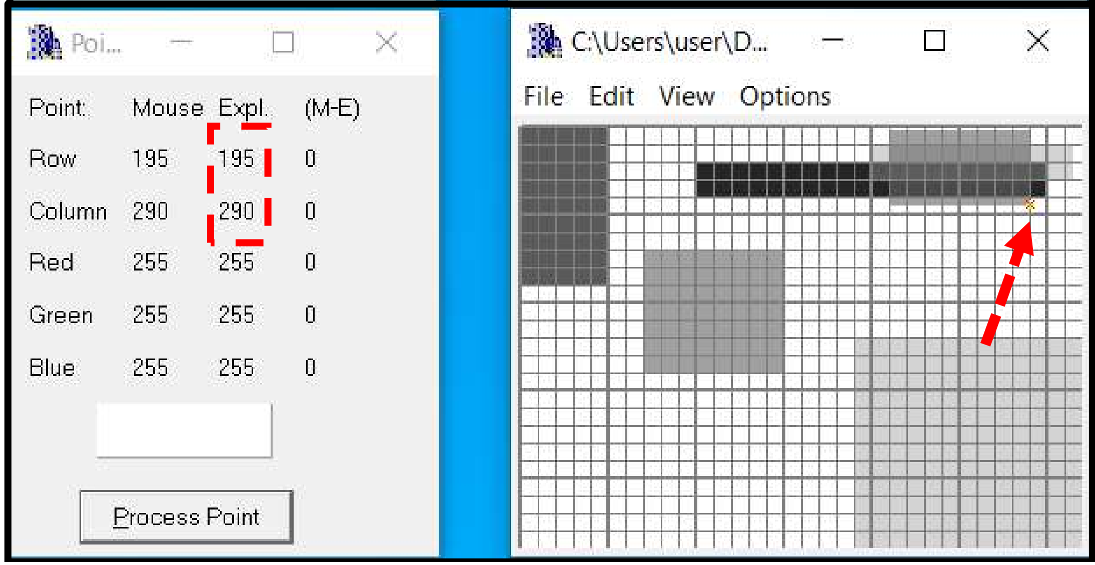


This test image will be listed as #5

$$p_1 = (210,238)$$
$$p_2 = (290,195)$$

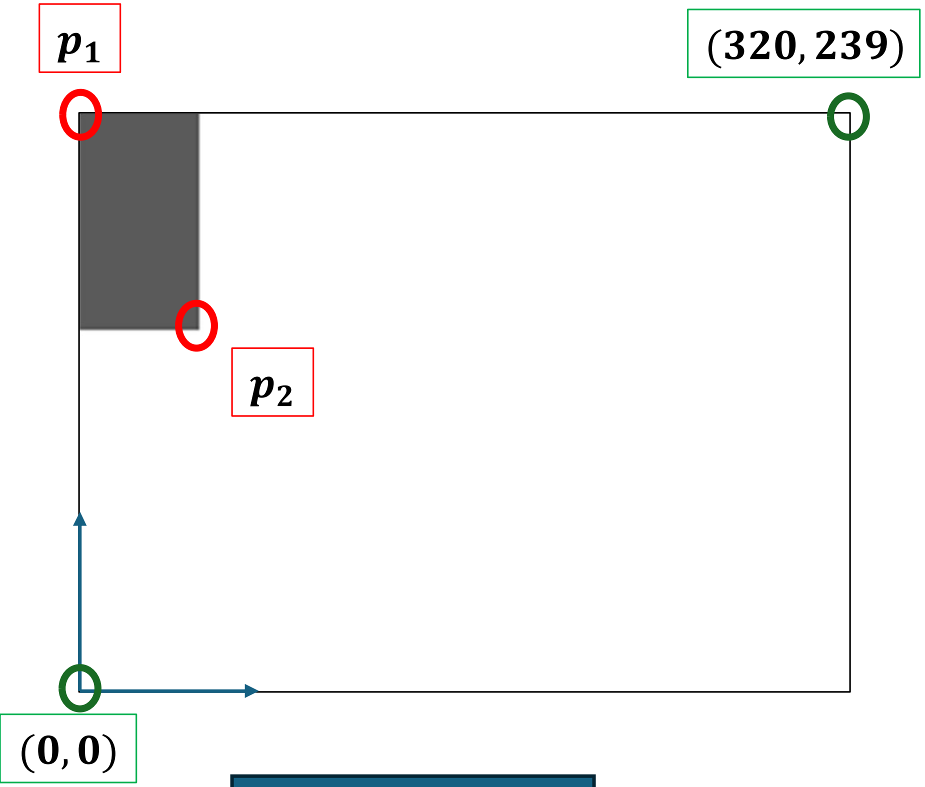


```
Point:   Mouse  Expl.  (M-E)
Row      238    238    0
Column   210    210    0
Red      255    255    0
Green    255    255    0
Blue     255    255    0
```



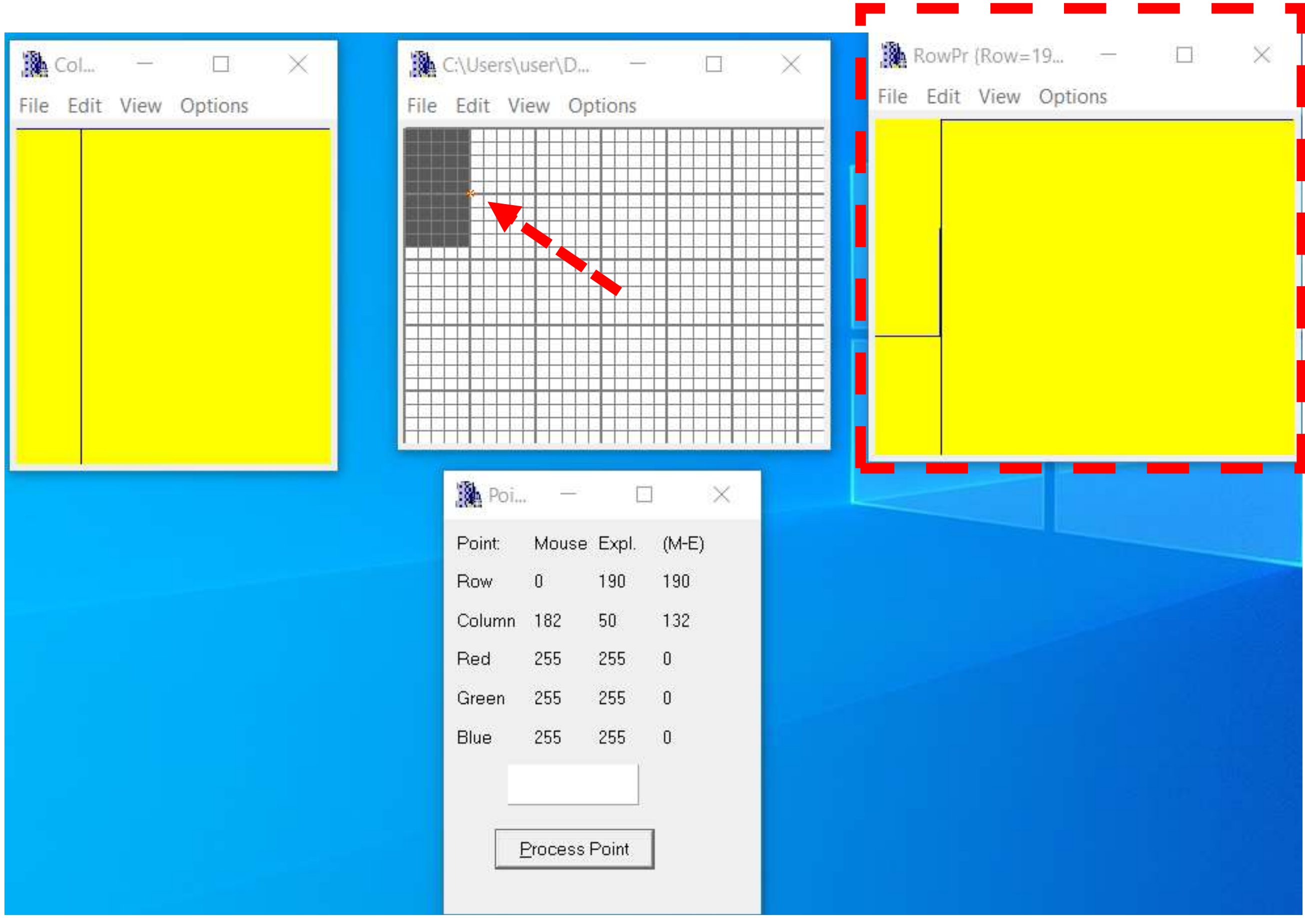
```
Point:   Mouse  Expl.  (M-E)
Row      195    195    0
Column   290    290    0
Red      255    255    0
Green    255    255    0
Blue     255    255    0
```

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 1



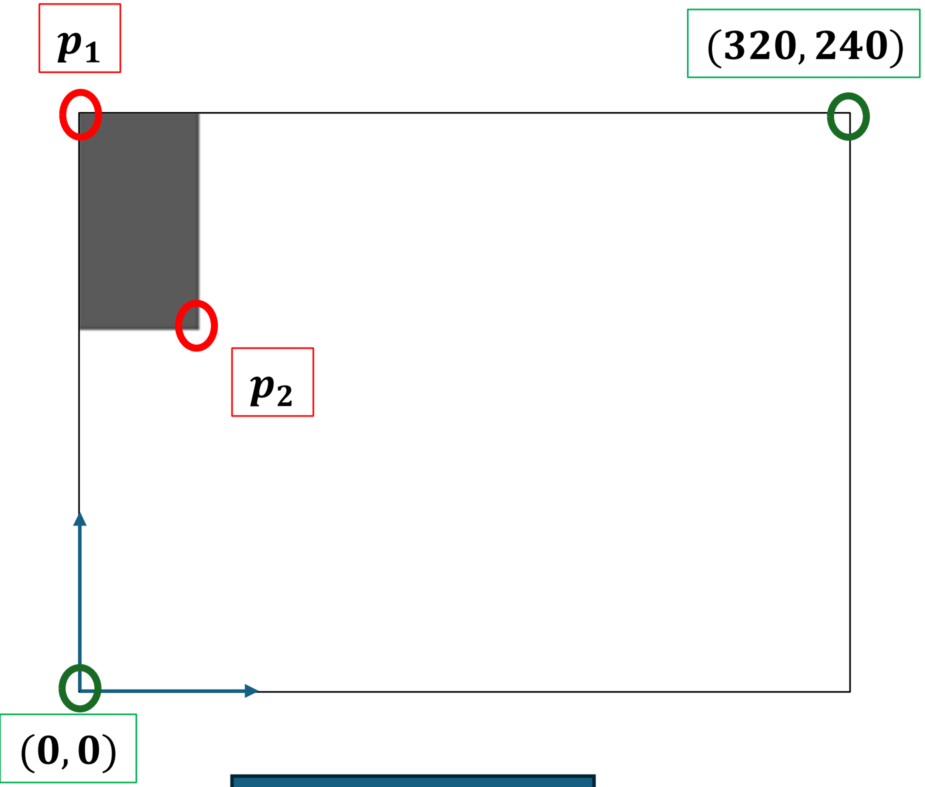
$p_1 = (0,239)$
$p_2 = (50,150)$

Regarding **test image #1** – we look at the border of the rectangle and can see a clear step (like a step function) in the rows view.



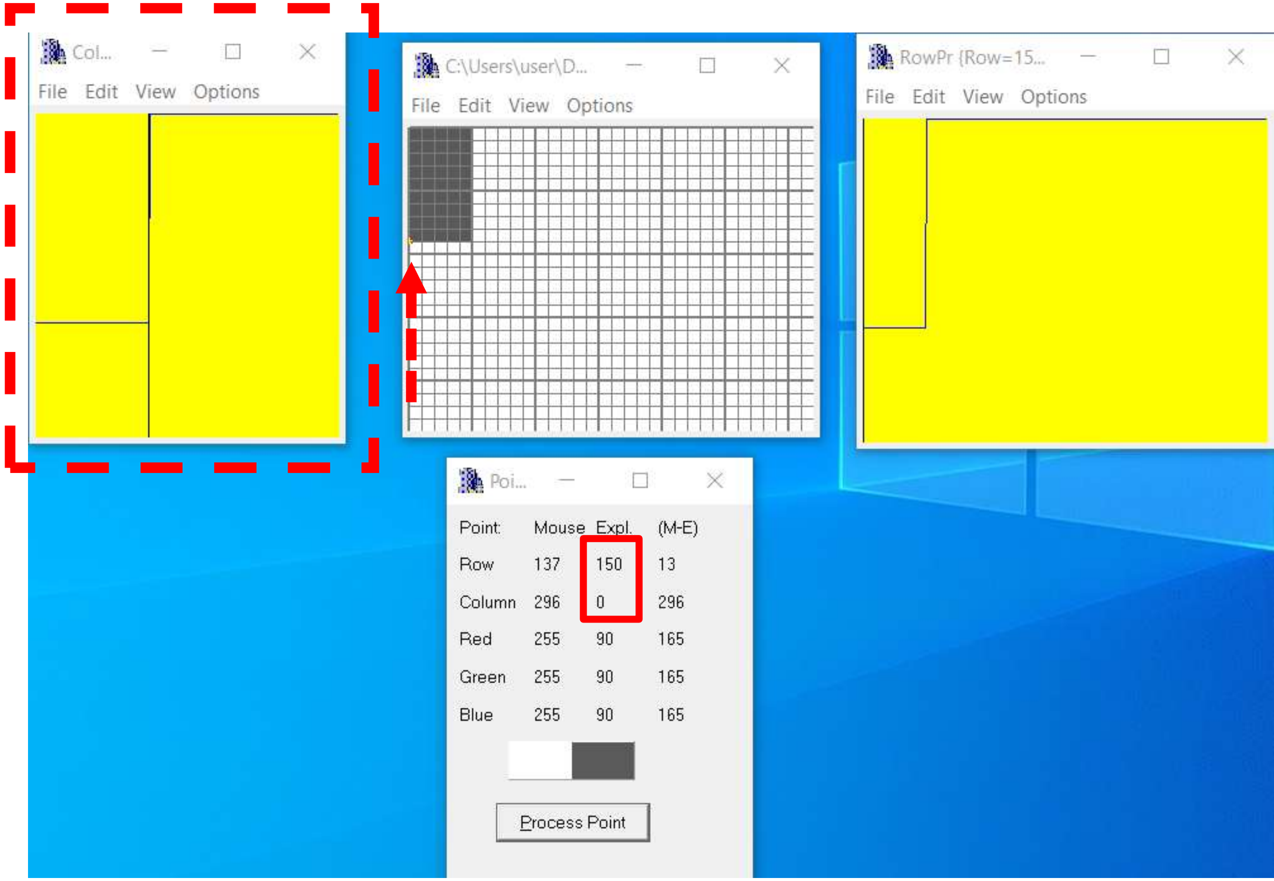
Point:	Mouse	Expl.	(M-E)
Row	0	190	190
Column	182	50	132
Red	255	255	0
Green	255	255	0
Blue	255	255	0

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 2



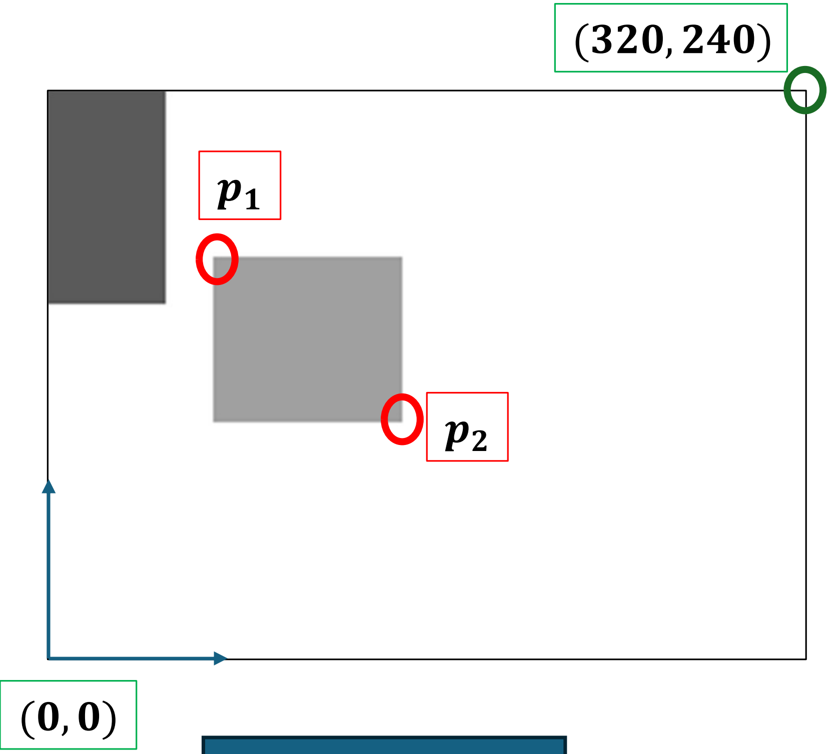
$$p_1 = (0,239)$$
$$p_2 = (50,150)$$

Regarding **test image #1** – we look at the border of the rectangle and can see a clear step (like a step function) in the columns view.



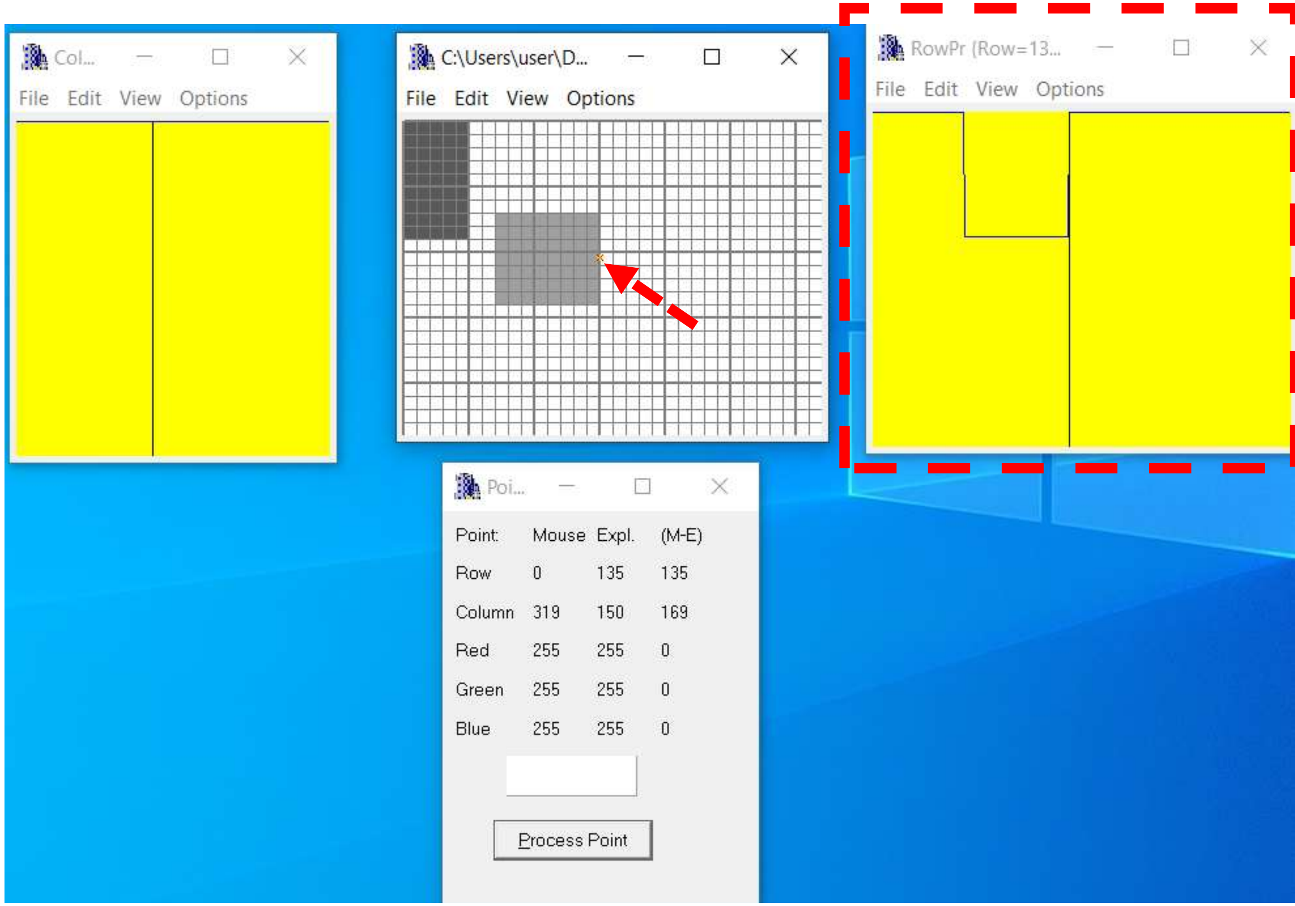
Point	Mouse	Expl.	(M-E)
Row	137	150	13
Column	296	0	296
Red	255	90	165
Green	255	90	165
Blue	255	90	165

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 3



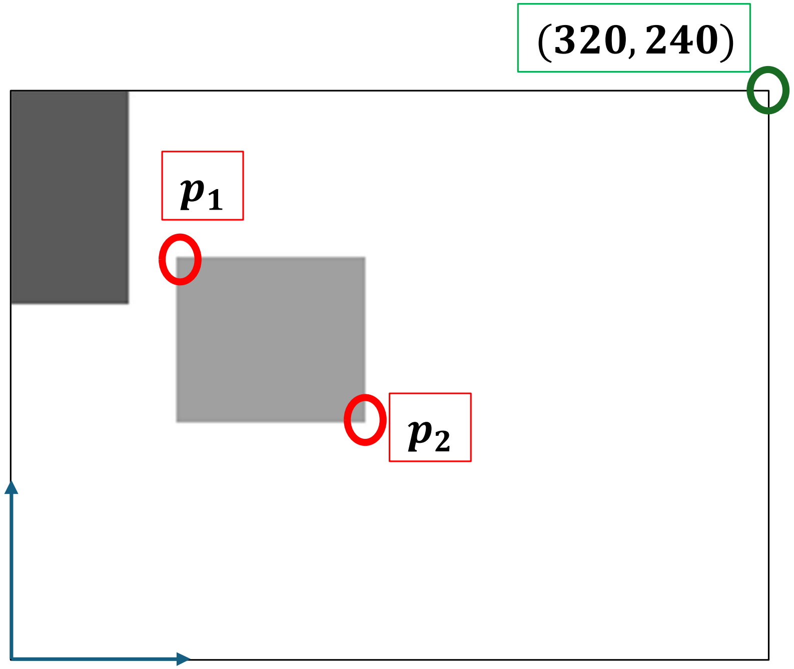
$p_1 = (70,170)$
$p_2 = (150,100)$

Regarding **test image #2** – we look at the border of the rectangle and can see a clear hole (like a potential hole) in the rows view. The hole is between the borders of the second rectangle



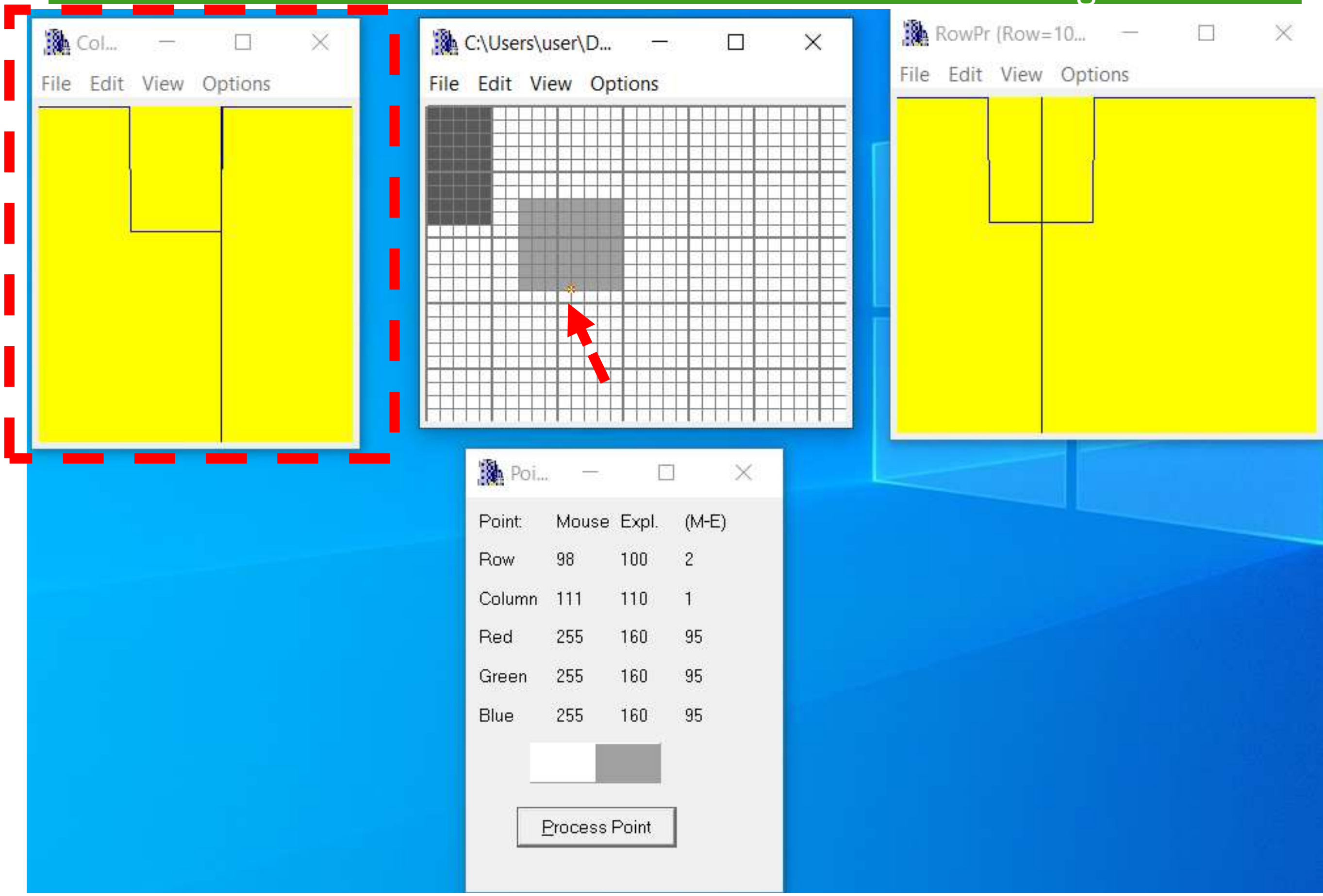
Point:	Mouse	Expl.	(M-E)
Row	0	135	135
Column	319	150	169
Red	255	255	0
Green	255	255	0
Blue	255	255	0

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 4



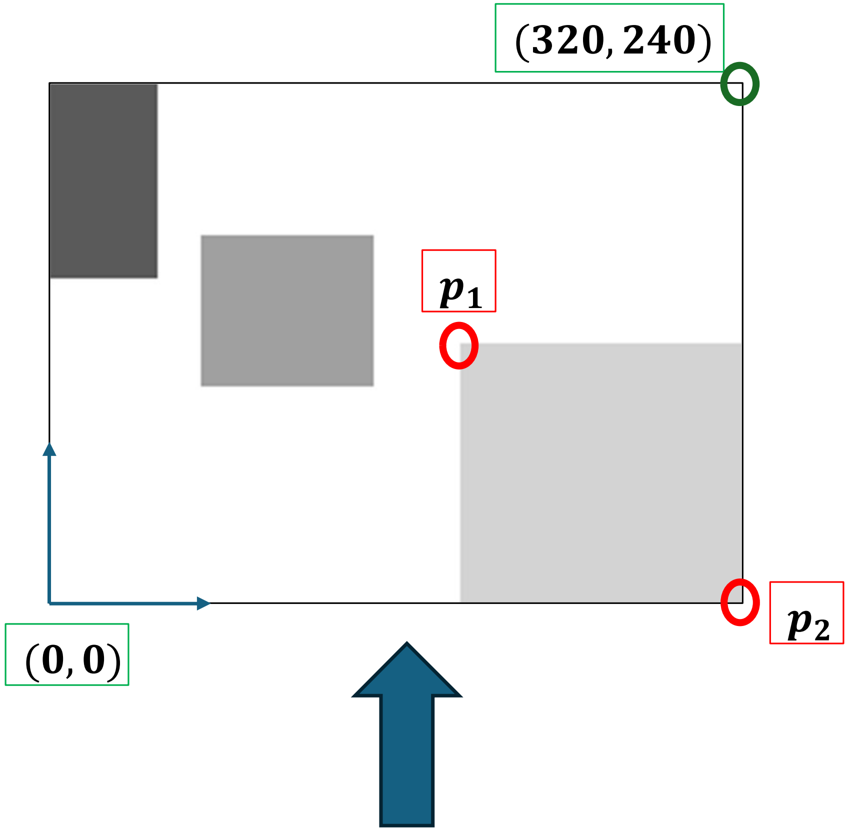
$$p_1 = (70{,}170)$$
$$p_2 = (150{,}100)$$

Regarding **test image #2** – we look at the border of the rectangle and can see a clear hole (like a potential hole) in the columns view. The hole is between the borders of the second rectangle



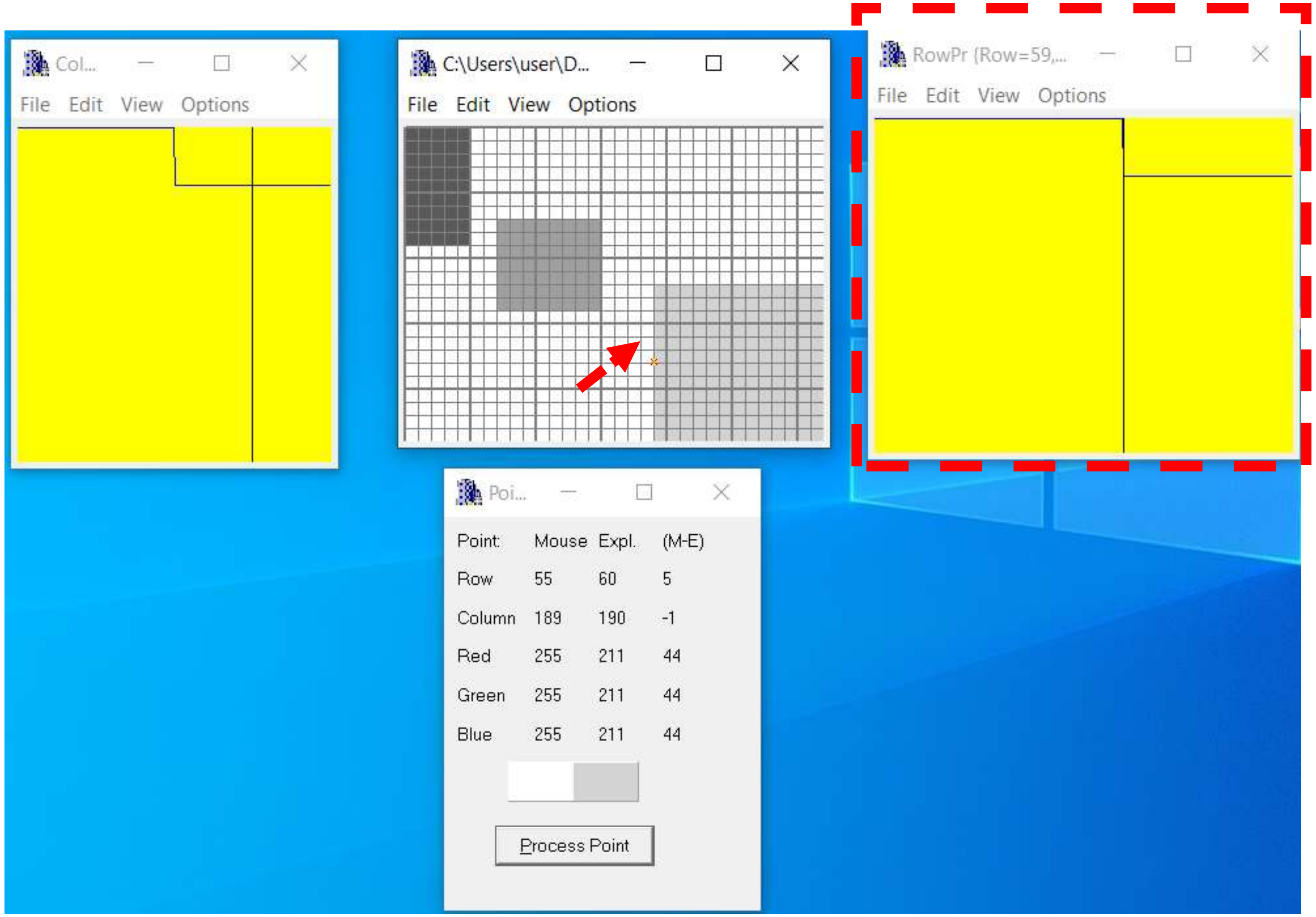
Point:	Mouse	Expl.	(M-E)
Row	98	100	2
Column	111	110	1
Red	255	160	95
Green	255	160	95
Blue	255	160	95

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 5



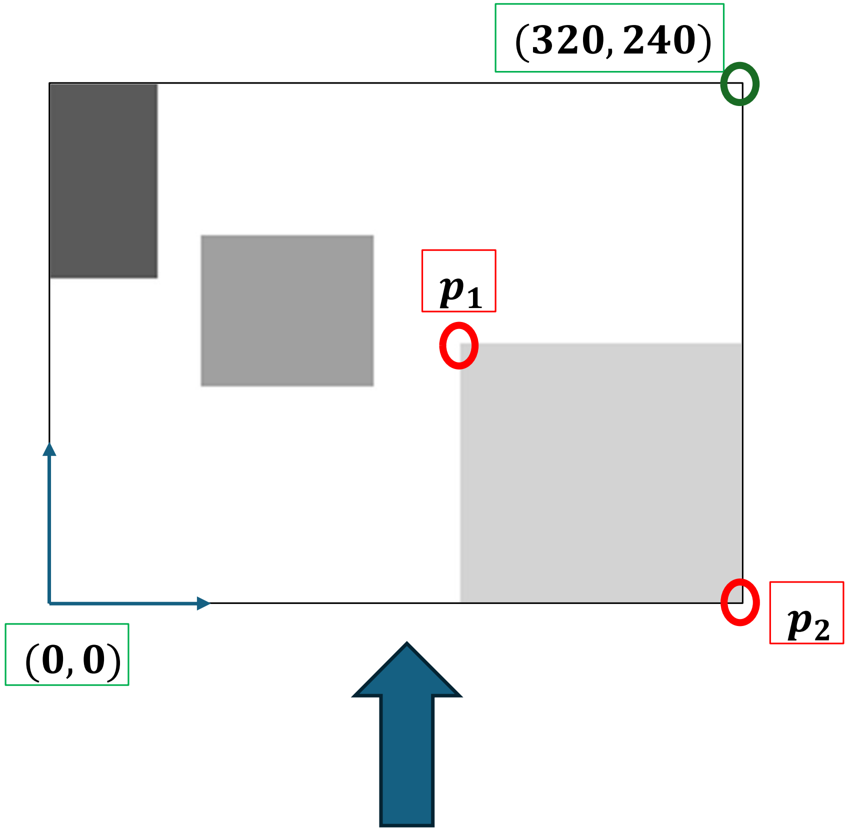
$p_1 = (190,120)$
$p_2 = (320,0)$

Regarding **test image #3** – we look at the border of the rectangle and can see a step small in the rows view. The step is between the borders of the third rectangle



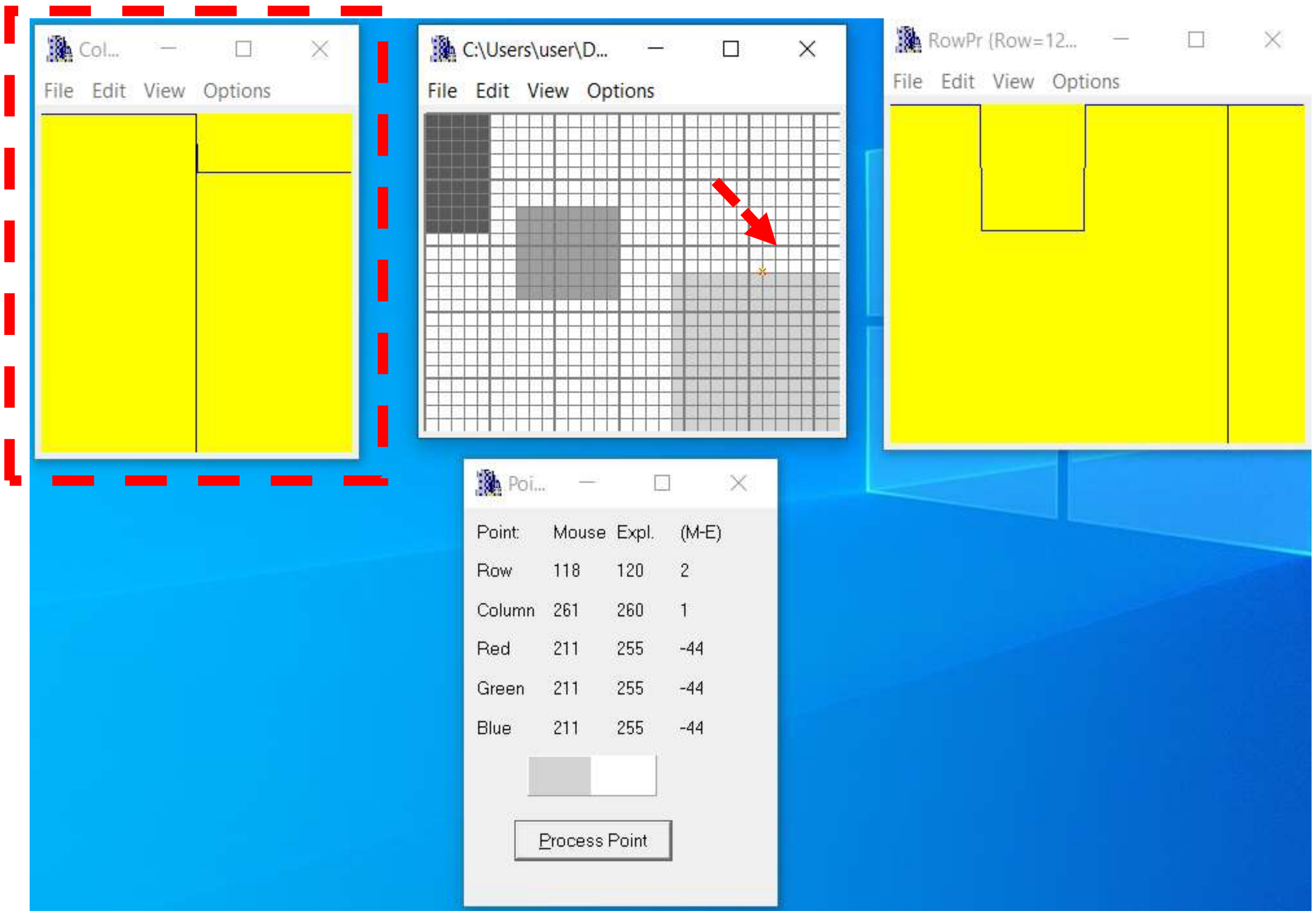
Point	Mouse	Expl.	(M-E)
Row	55	60	5
Column	189	190	-1
Red	255	211	44
Green	255	211	44
Blue	255	211	44

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 6



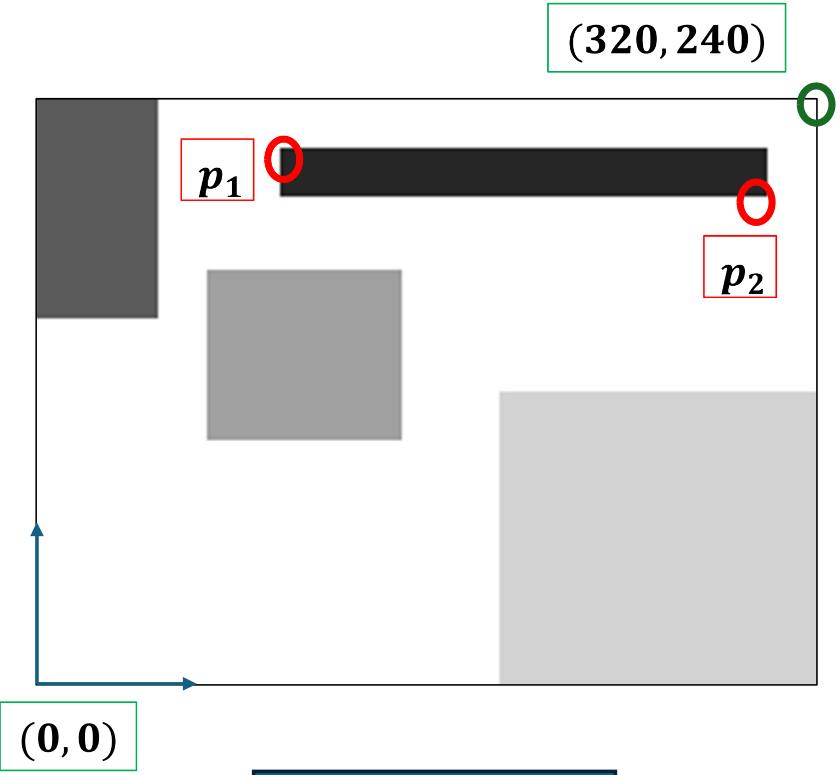
$$p_1 = (190,120)$$
$$p_2 = (320,0)$$

Regarding **test image #3** – we look at the border of the rectangle and can see a step small in the columns view. The step is between the borders of the third rectangle

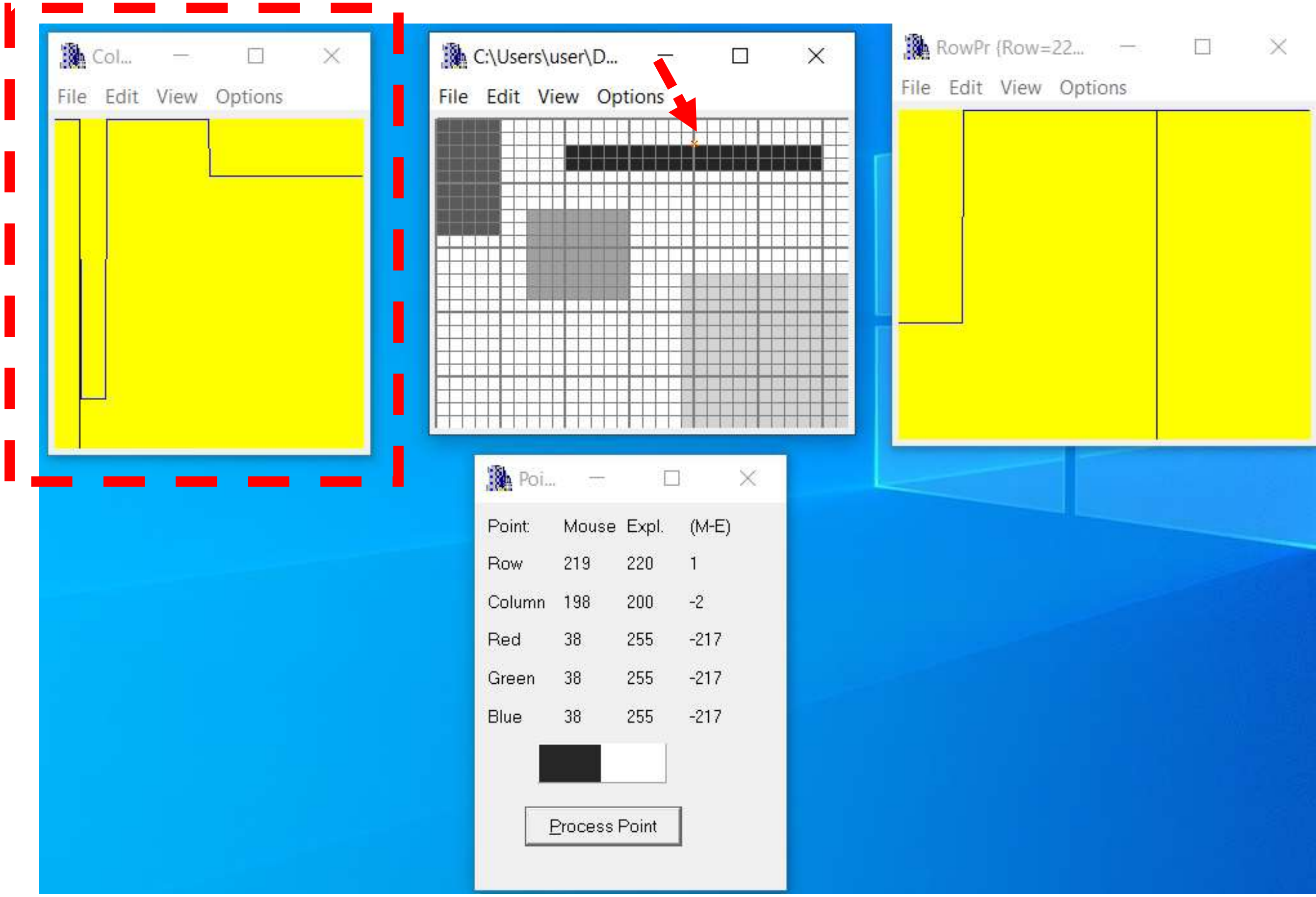


```
Point   Mouse  Expl.  (M-E)
Row     118    120    2
Column  261    260    1
Red     211    255    -44
Green   211    255    -44
Blue    211    255    -44
```

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 7

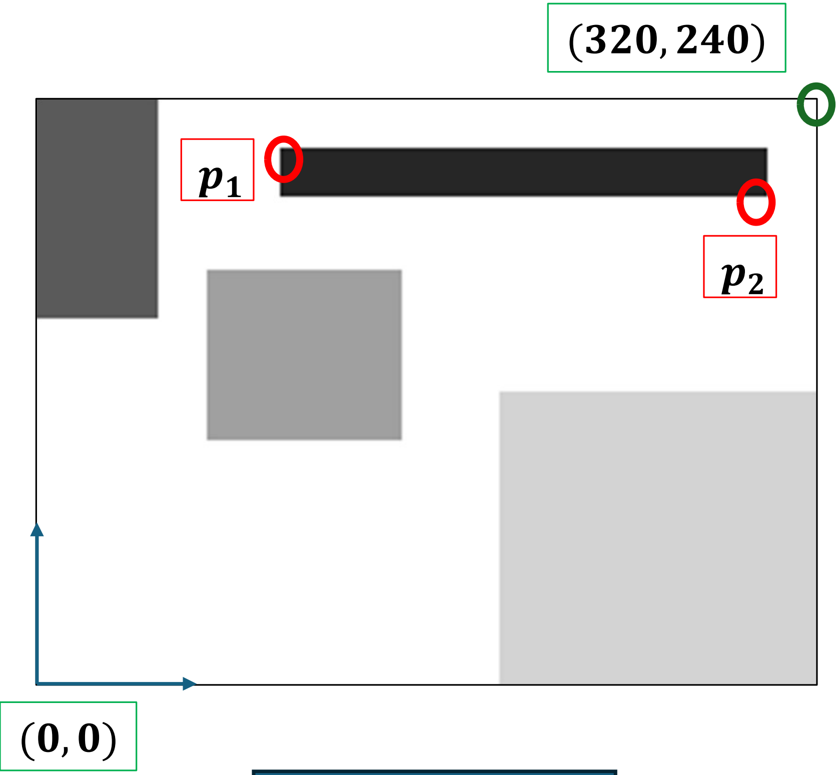


$$p_1 = (100{,}220)$$
$$p_2 = (300{,}200)$$

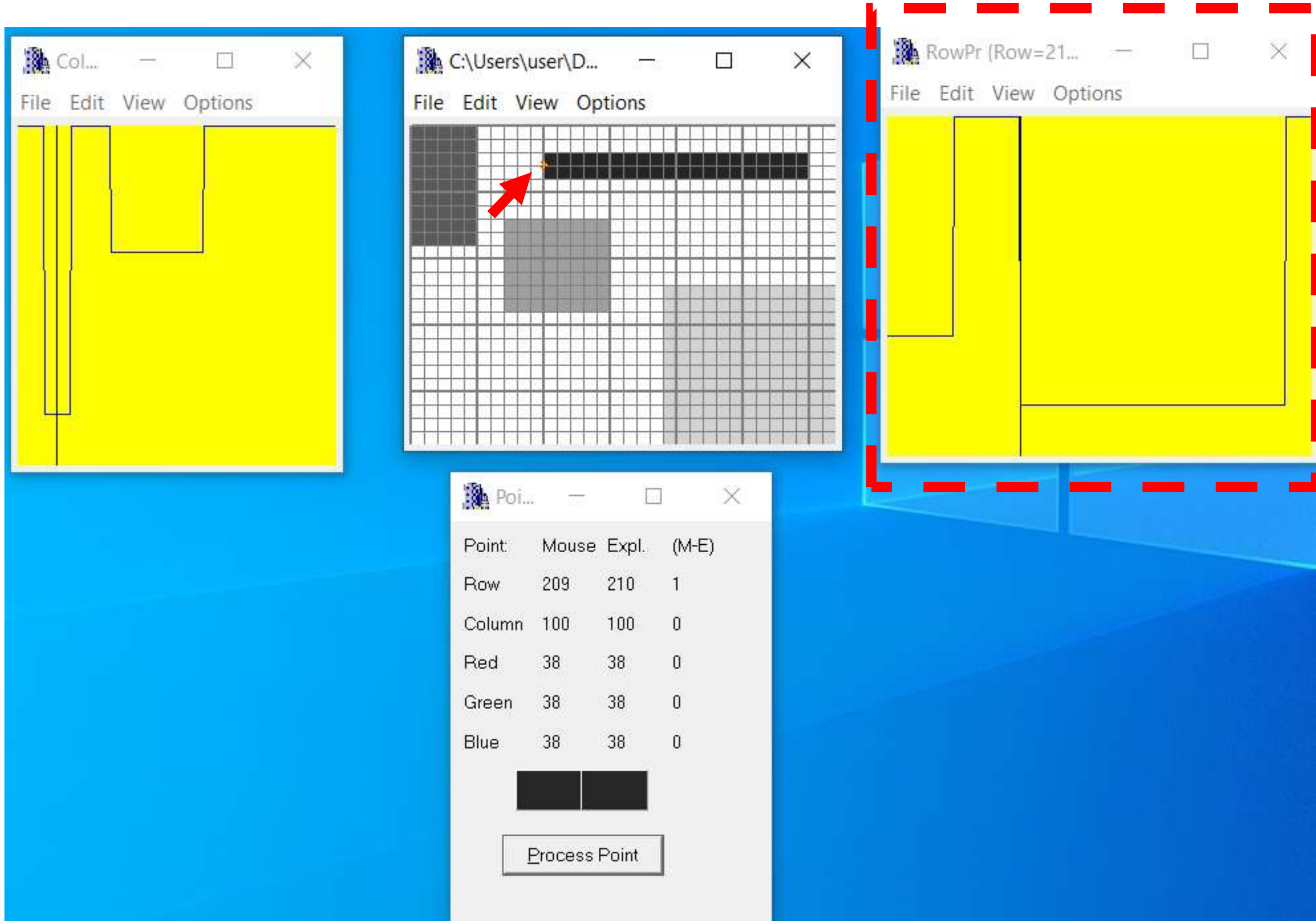


```
Point    Mouse  Expl.  (M-E)
Row      219    220    1
Column   198    200    -2
Red      38     255    -217
Green    38     255    -217
Blue     38     255    -217
```

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 8

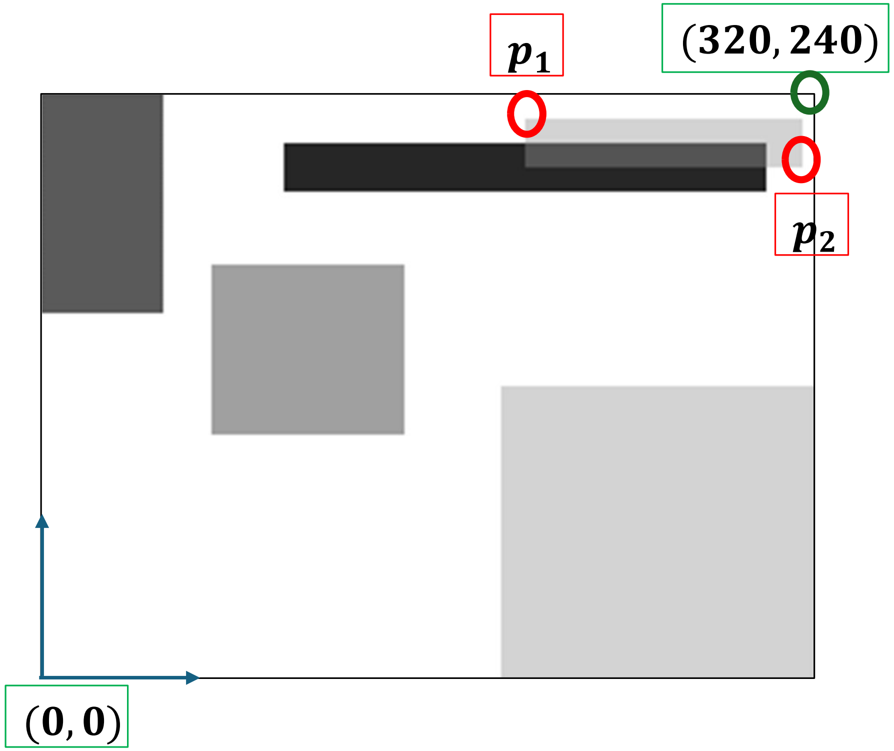


$p_1 = (100,220)$
$p_2 = (300,200)$

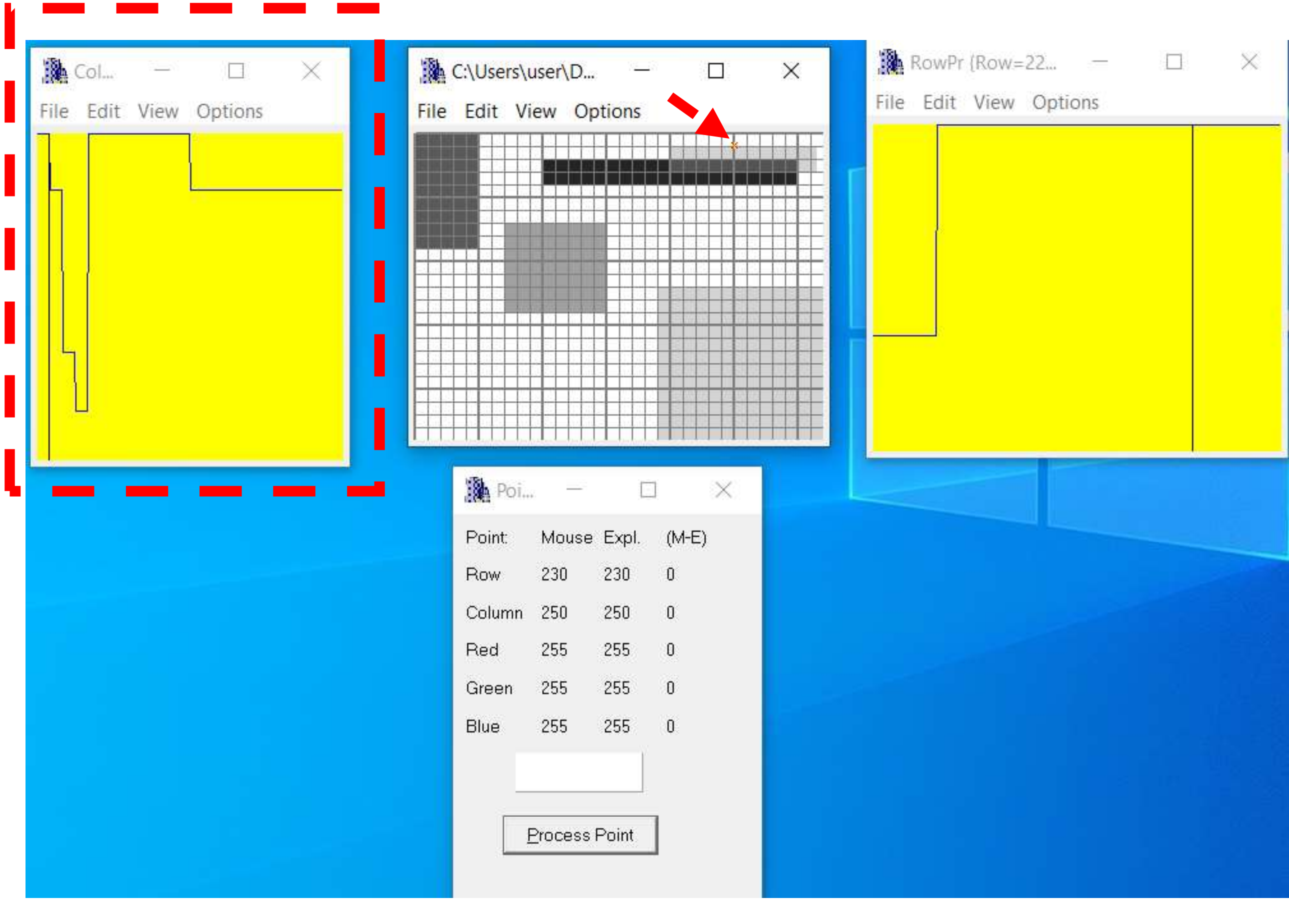


Point	Mouse	Expl.	(M-E)
Row	209	210	1
Column	100	100	0
Red	38	38	0
Green	38	38	0
Blue	38	38	0

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 9

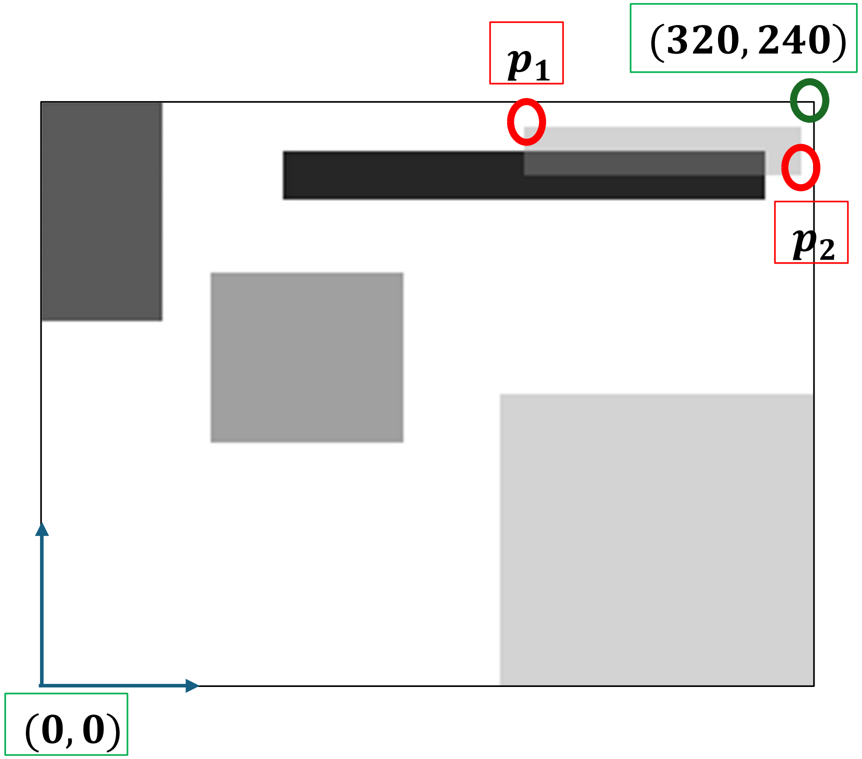


$$p_1 = (200,230)$$
$$p_2 = (315,210)$$

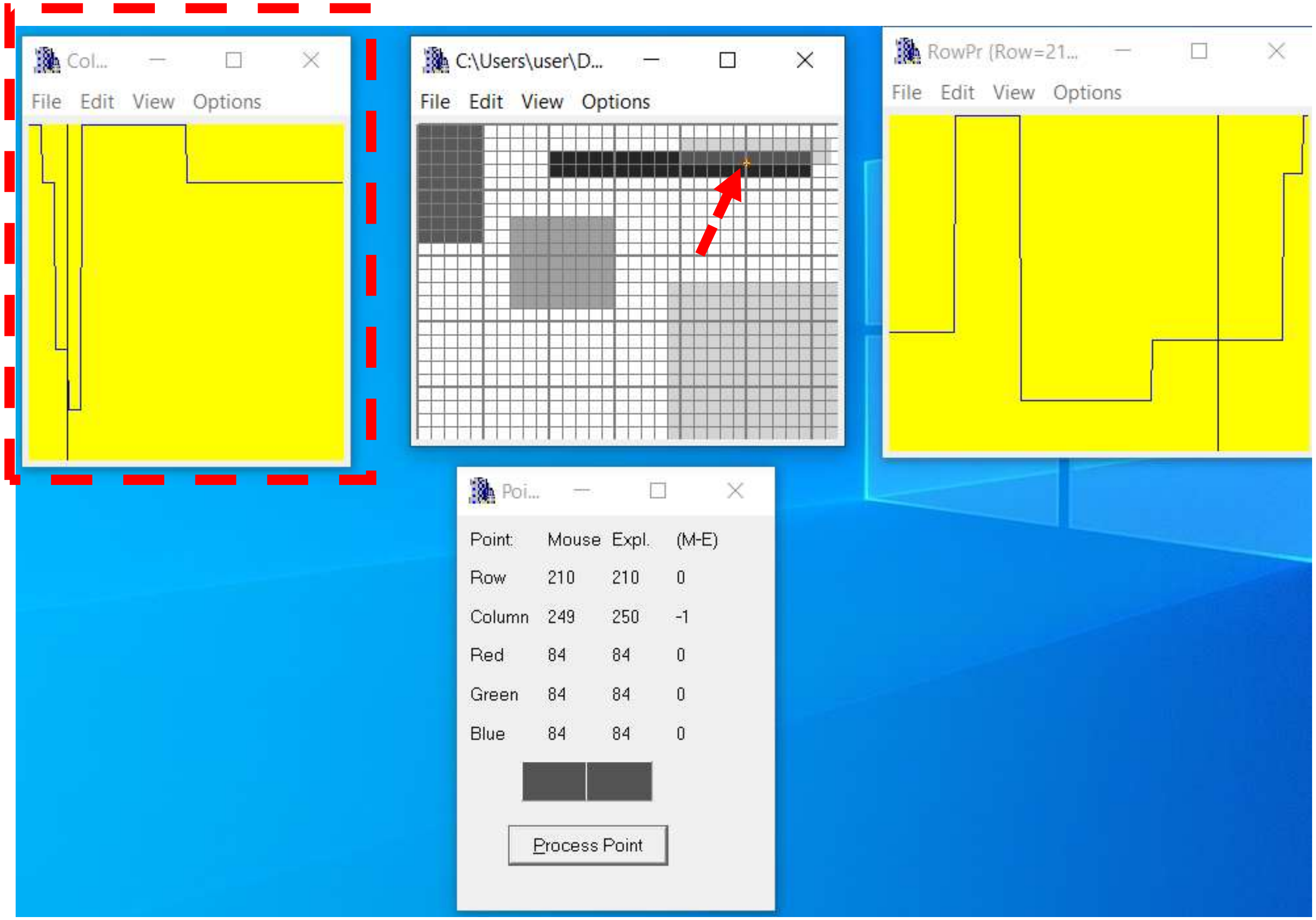


Point:	Mouse	Expl.	(M-E)
Row	230	230	0
Column	250	250	0
Red	255	255	0
Green	255	255	0
Blue	255	255	0

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 10

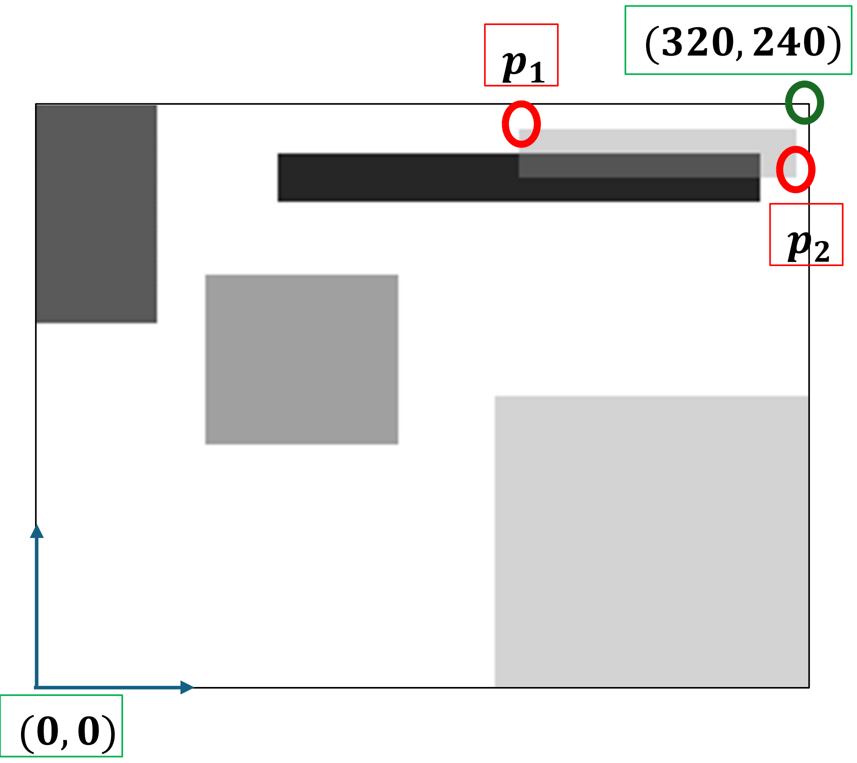


$p_1 = (200,230)$
$p_2 = (315,210)$

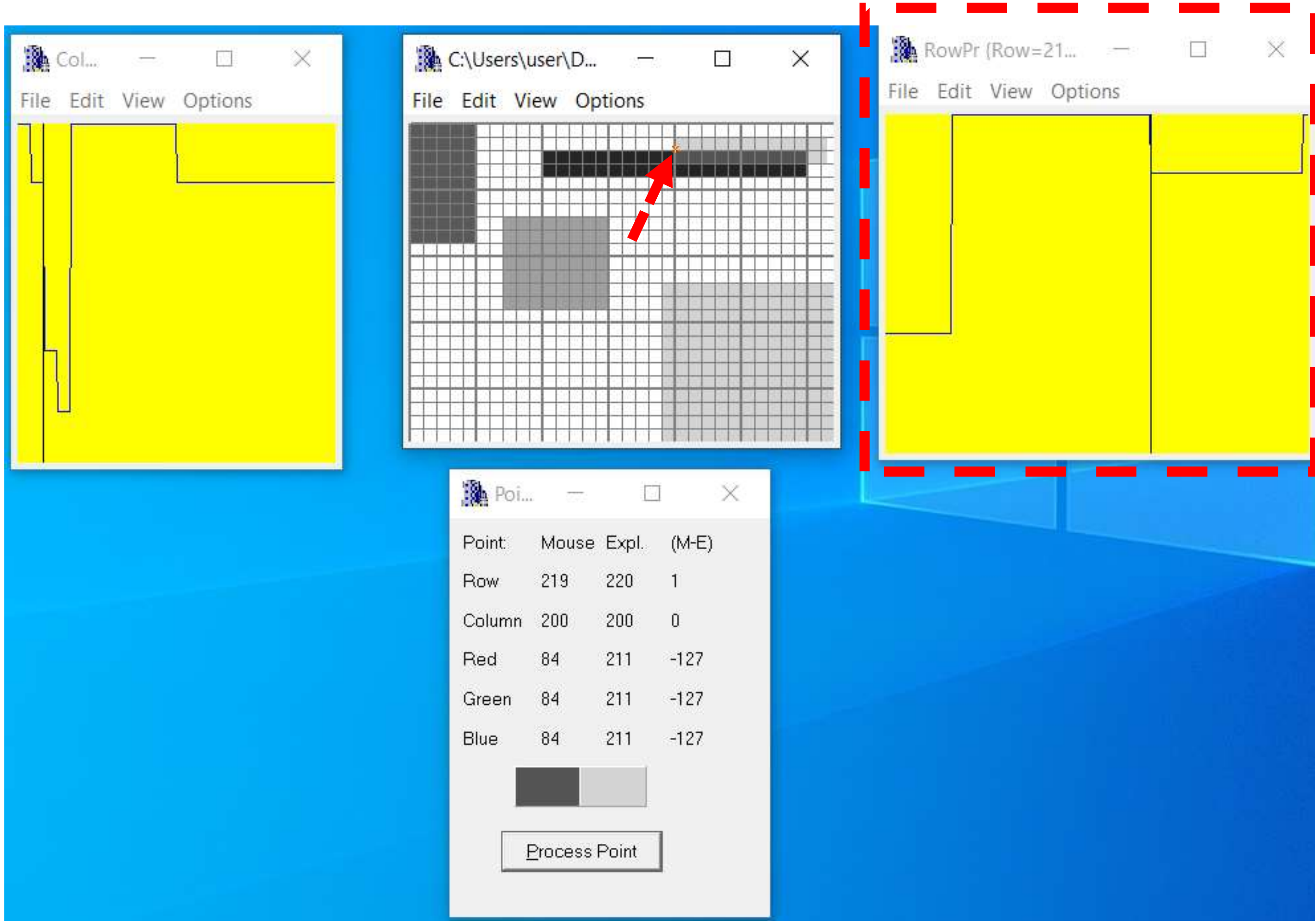


```
Point    Mouse  Expl.  (M-E)
Row      210    210    0
Column   249    250    -1
Red      84     84     0
Green    84     84     0
Blue     84     84     0
```

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 11

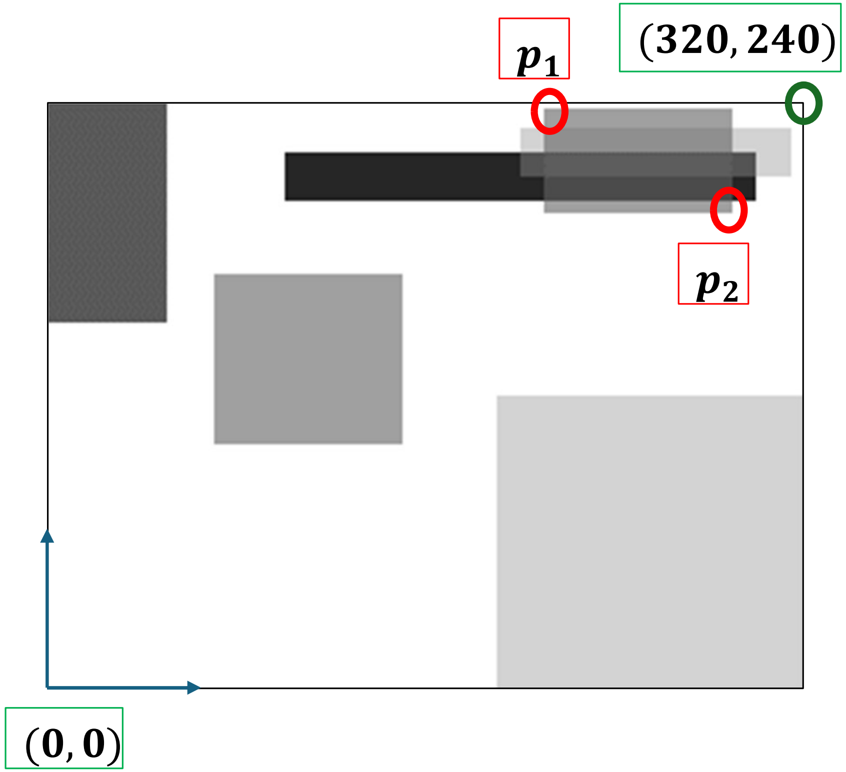


$p_1 = (200,230)$
$p_2 = (315,210)$

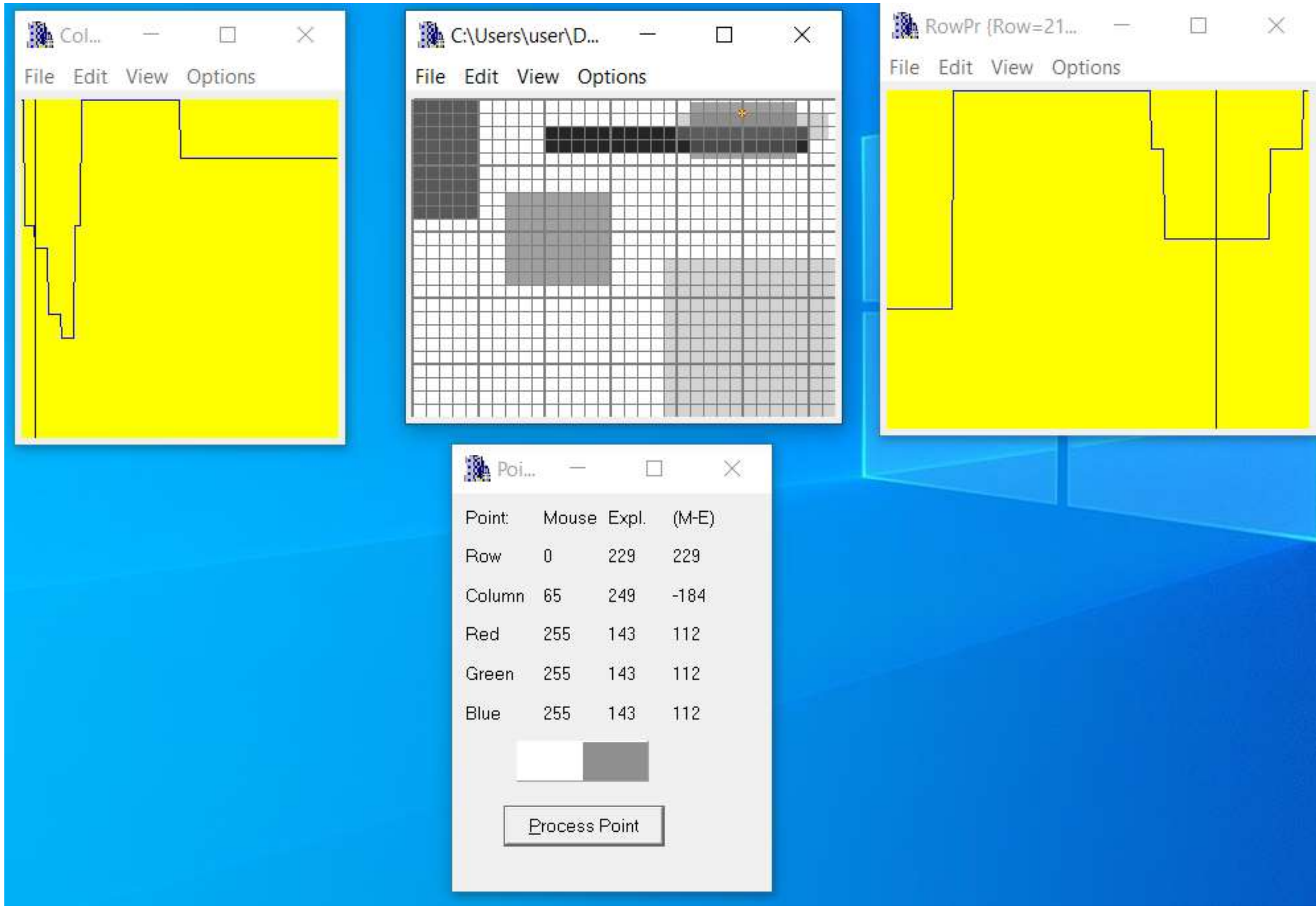


Point:	Mouse	Expl.	(M-E)
Row	219	220	1
Column	200	200	0
Red	84	211	-127
Green	84	211	-127
Blue	84	211	-127

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 12

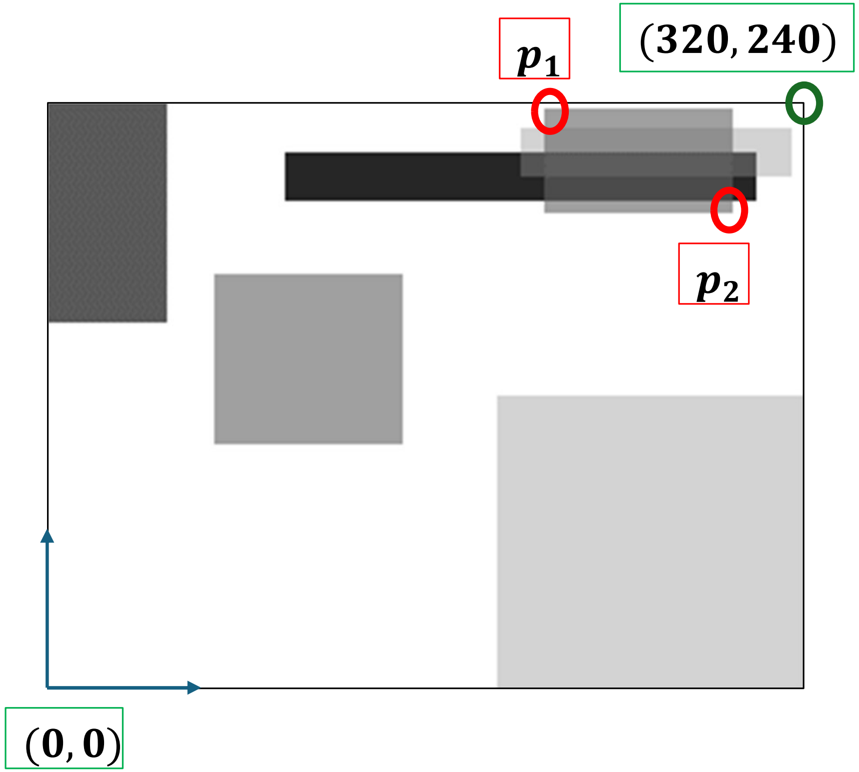


$p_1 = (210,238)$
$p_2 = (290,195)$

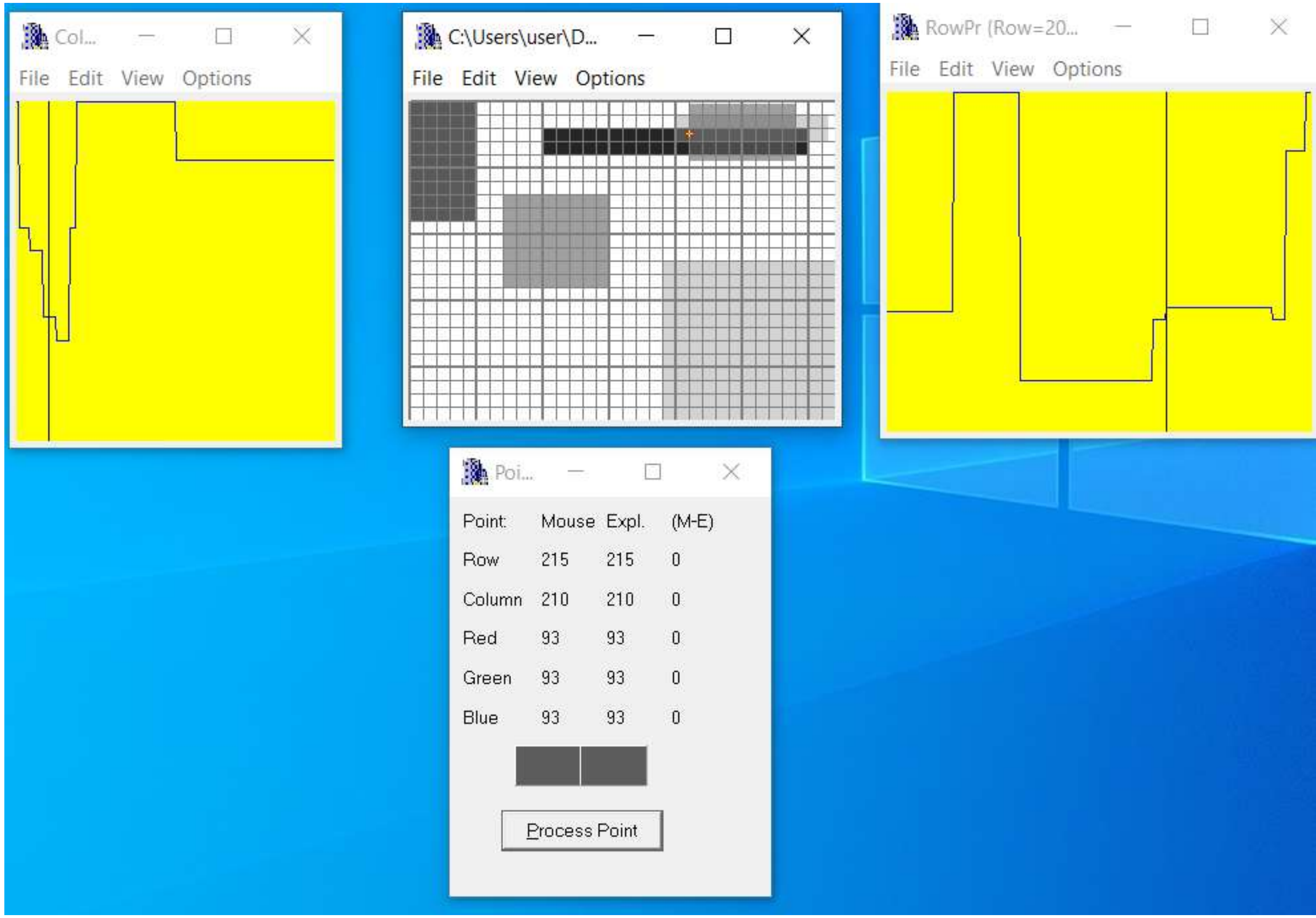


```
Point:   Mouse  Expl.  (M-E)
Row      0      229    229
Column   65     249    -184
Red      255    143    112
Green    255    143    112
Blue     255    143    112
```

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 13

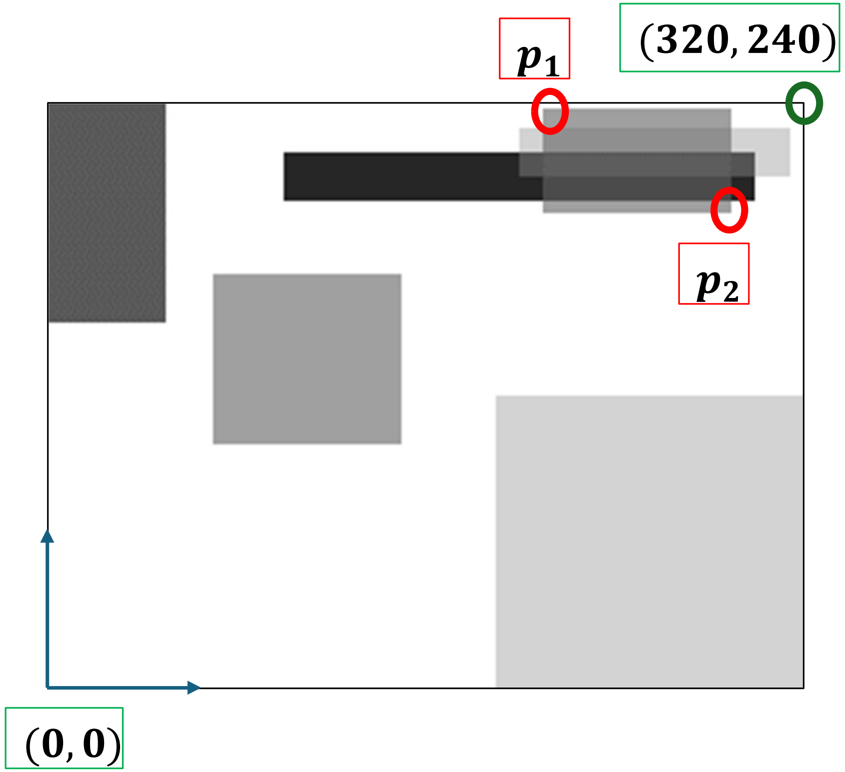


$p_1 = (210,238)$
$p_2 = (290,195)$

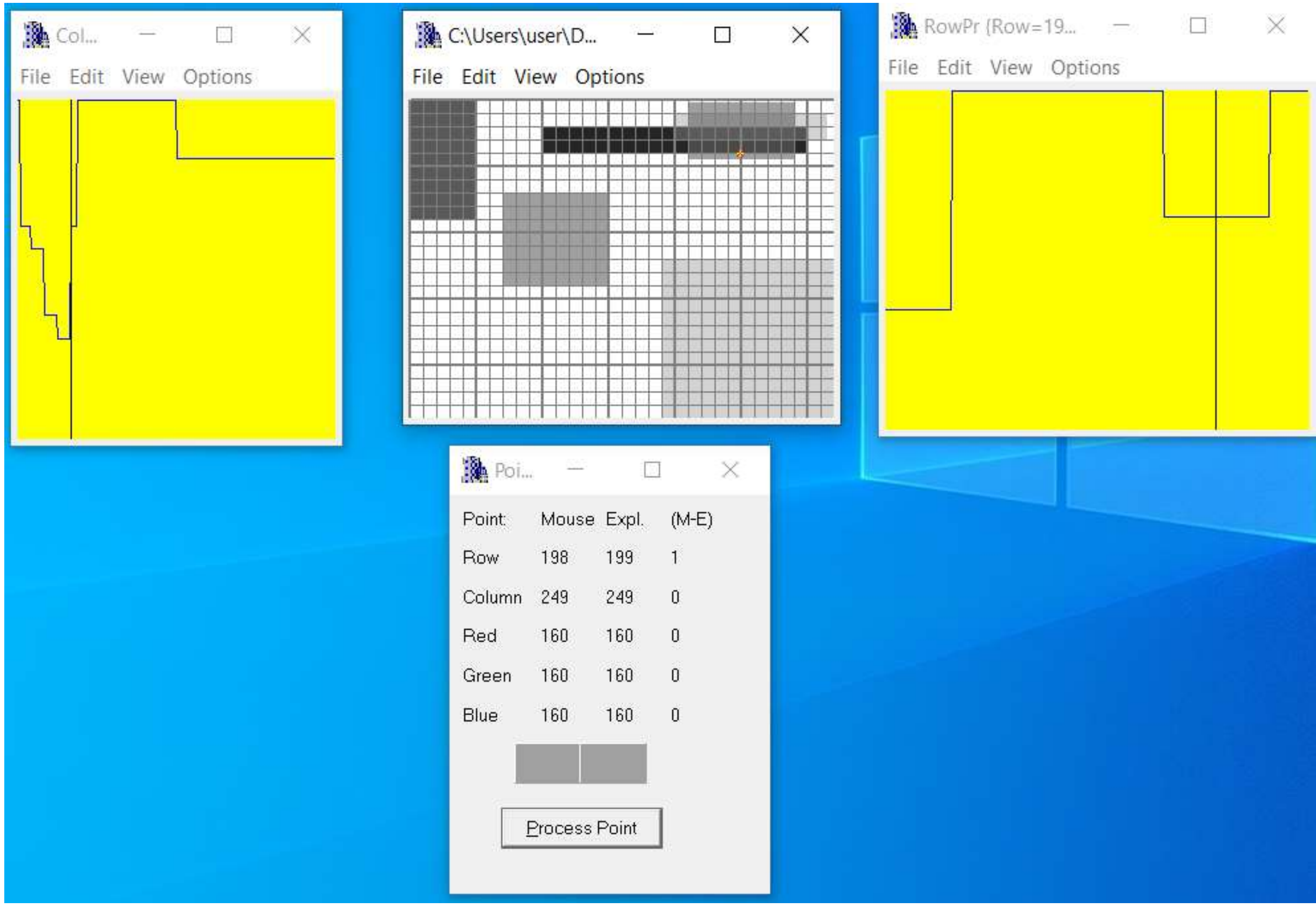


```
Point:   Mouse  Expl.  (M-E)
Row      215    215    0
Column   210    210    0
Red      93     93     0
Green    93     93     0
Blue     93     93     0
```

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 14

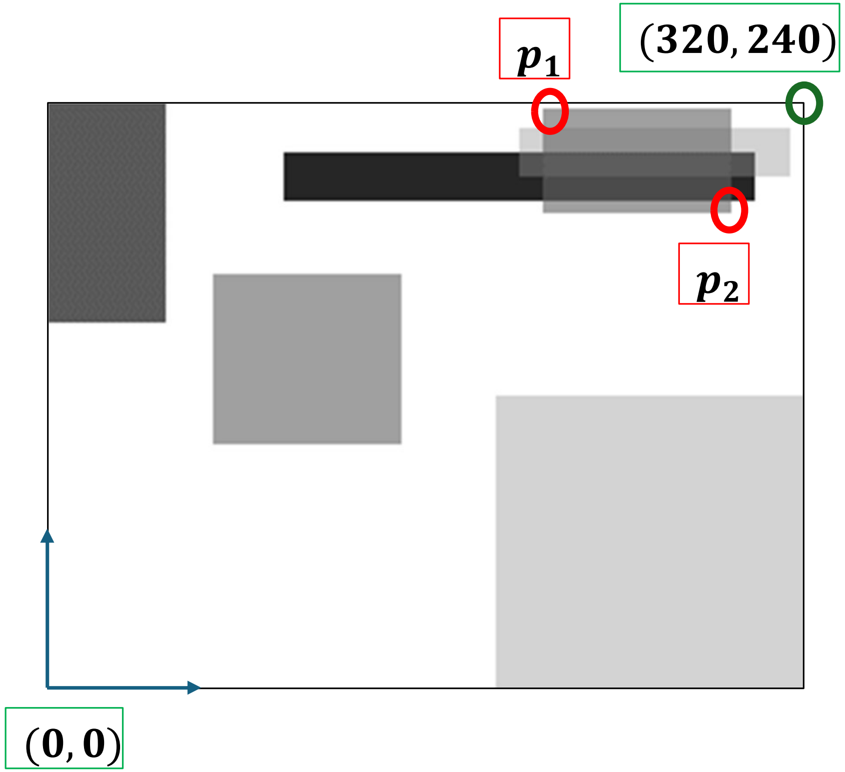


$$p_1 = (210,238)$$
$$p_2 = (290,195)$$

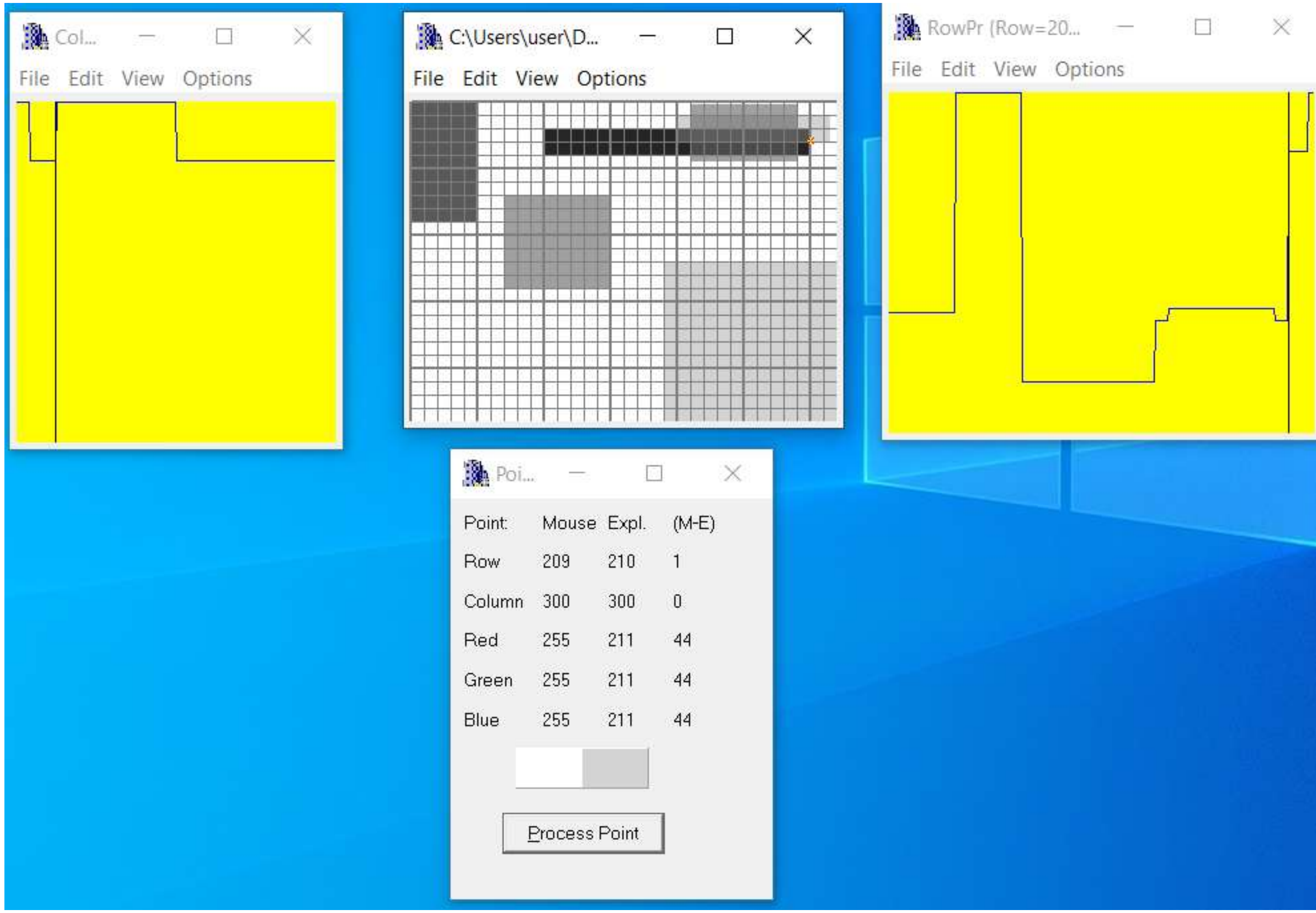


Point:	Mouse	Expl.	(M-E)
Row	198	199	1
Column	249	249	0
Red	160	160	0
Green	160	160	0
Blue	160	160	0

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 15

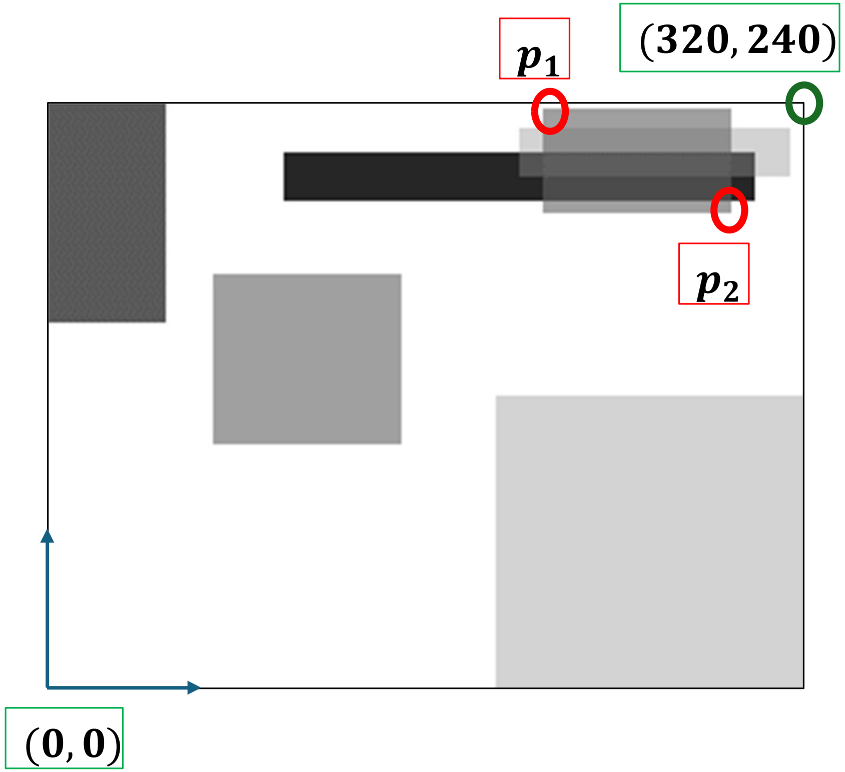


$$p_1 = (210,238)$$
$$p_2 = (290,195)$$

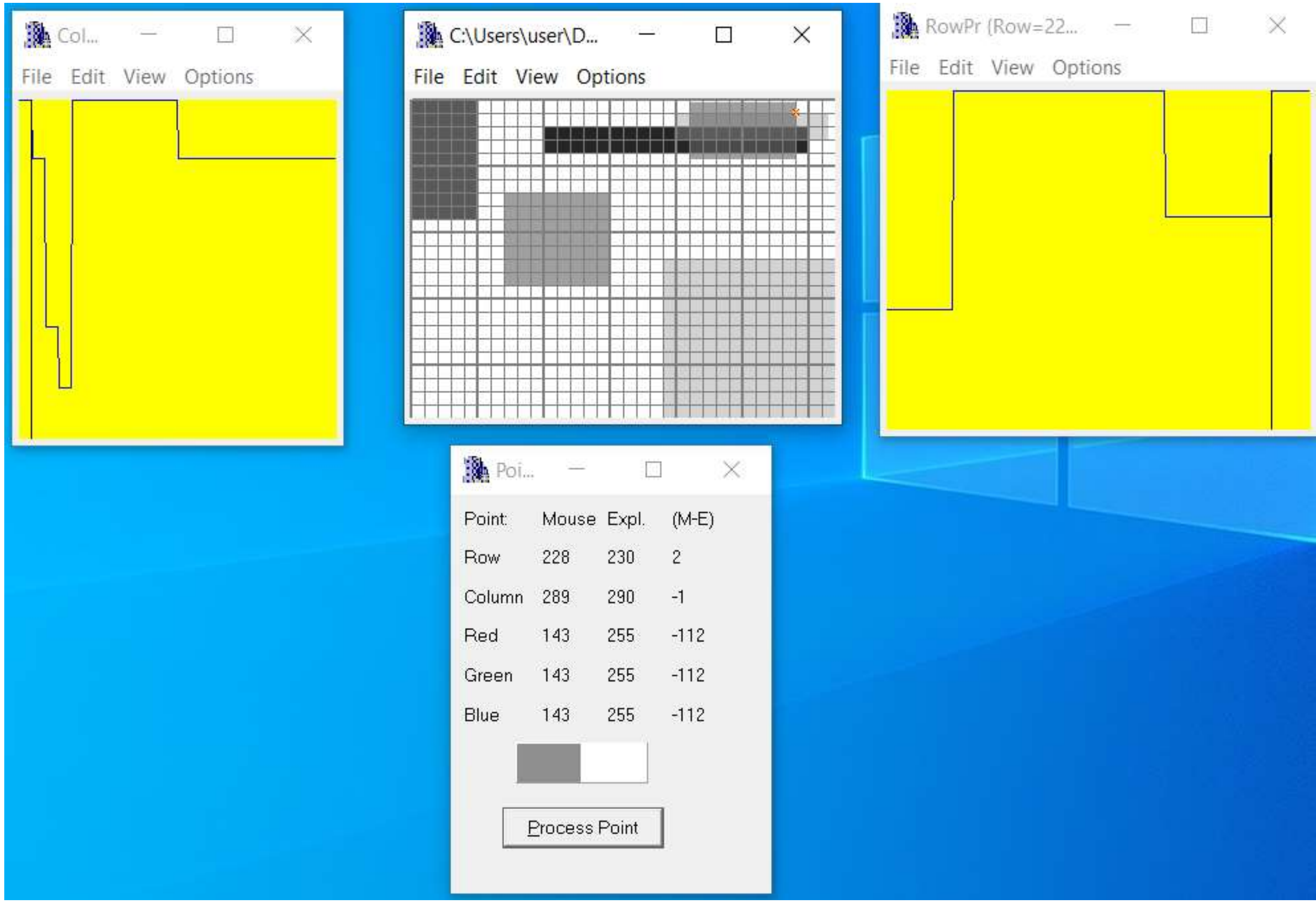


```
Point:   Mouse  Expl.  (M-E)
Row      209    210    1
Column   300    300    0
Red      255    211    44
Green    255    211    44
Blue     255    211    44
```

11.3 For the specific test images (see 11.2), put relevant profiles and refer relevant lines of code to prove that specific test having specific rectangle were created as required – part 16

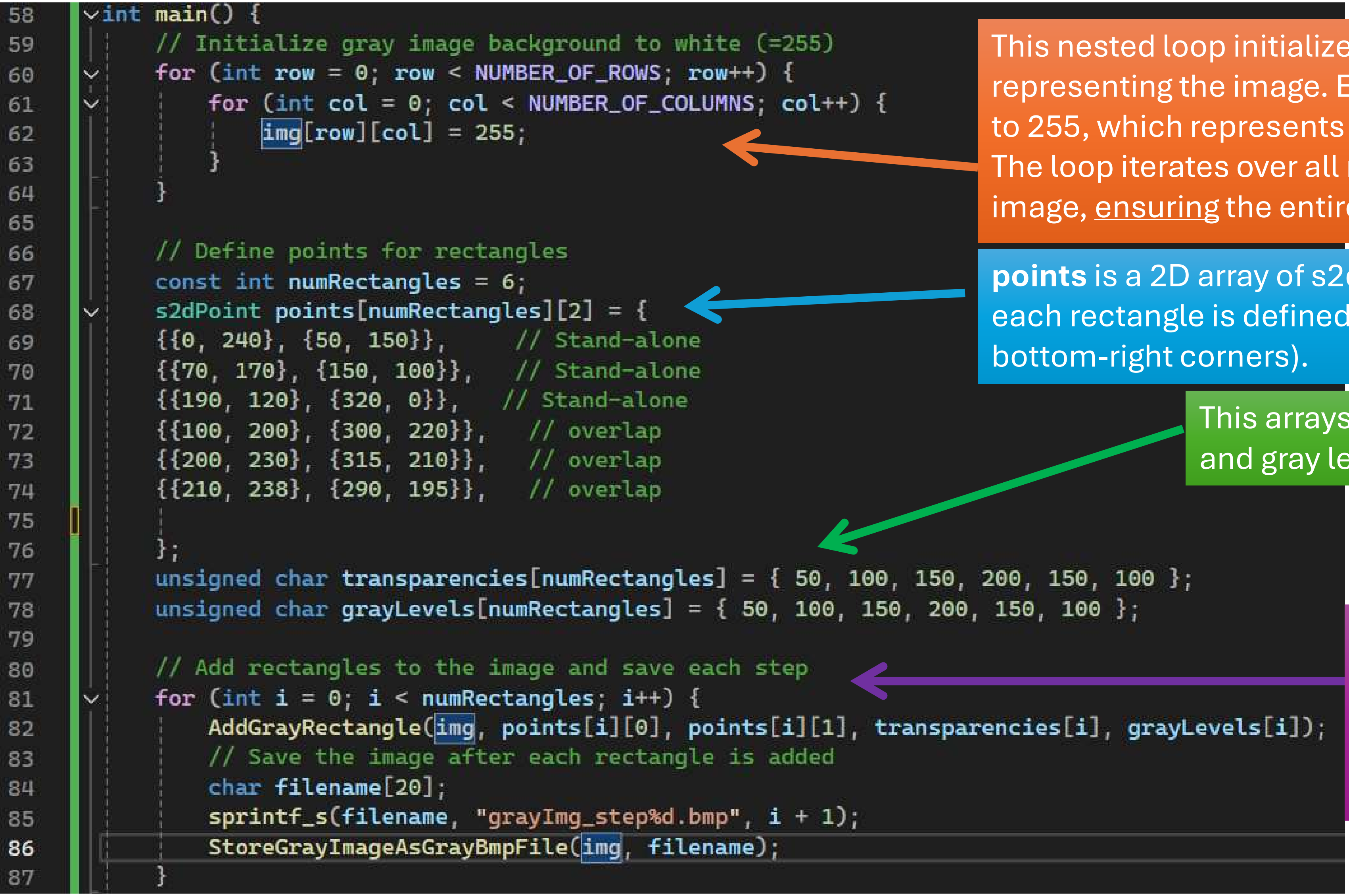


$p_1 = (210{,}238)$
$p_2 = (290{,}195)$



Point	Mouse	Expl.	(M-E)
Row	228	230	2
Column	289	290	-1
Red	143	255	-112
Green	143	255	-112
Blue	143	255	-112

11.4 Code of the “main” function and set of intermediate images-part 1



```
int main() {
    // Initialize gray image background to white (=255)
    for (int row = 0; row < NUMBER_OF_ROWS; row++) {
        for (int col = 0; col < NUMBER_OF_COLUMNS; col++) {
            img[row][col] = 255;
        }
    }

    // Define points for rectangles
    const int numRectangles = 6;
    s2dPoint points[numRectangles][2] = {
    {{0, 240}, {50, 150}},      // Stand-alone
    {{70, 170}, {150, 100}},    // Stand-alone
    {{190, 120}, {320, 0}},    // Stand-alone
    {{100, 200}, {300, 220}},    // overlap
    {{200, 230}, {315, 210}},    // overlap
    {{210, 238}, {290, 195}},    // overlap

    };
    unsigned char transparencies[numRectangles] = { 50, 100, 150, 200, 150, 100 };
    unsigned char grayLevels[numRectangles] = { 50, 100, 150, 200, 150, 100 };

    // Add rectangles to the image and save each step
    for (int i = 0; i < numRectangles; i++) {
        AddGrayRectangle(img, points[i][0], points[i][1], transparencies[i], grayLevels[i]);
        // Save the image after each rectangle is added
        char filename[20];
        sprintf_s(filename, "grayImg_step%d.bmp", i + 1);
        StoreGrayImageAsGrayBmpFile(img, filename);
    }
```

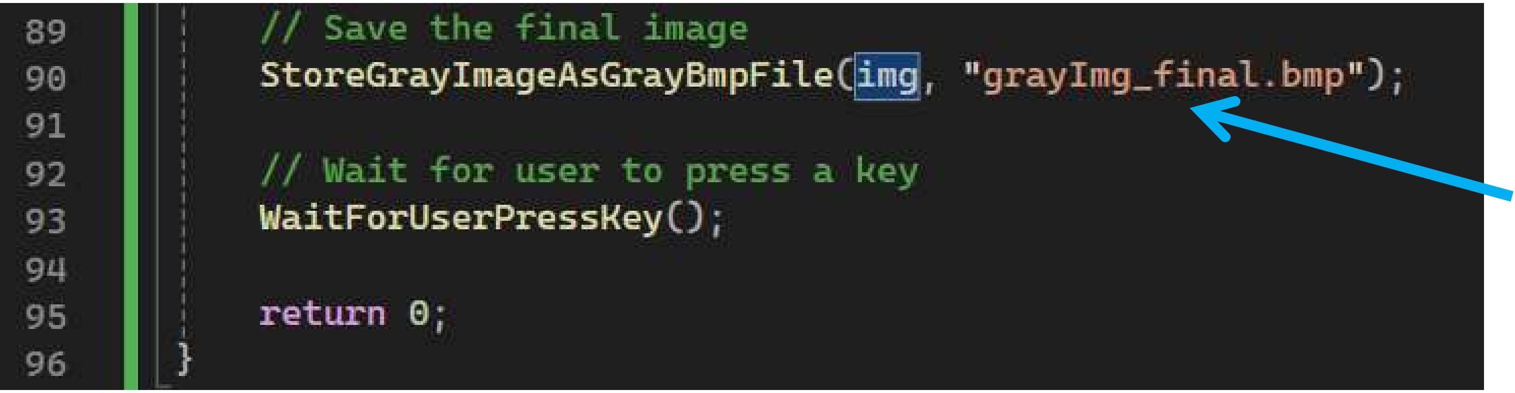
This nested loop initializes a 2D array **img** representing the image. Each pixel in the image is set to 255, which represents **white** in a grayscale image. The loop iterates over all rows and columns of the image, ensuring the entire background is white.

points is a 2D array of s2dPoint structures, where each rectangle is defined by two points (top-left and bottom-right corners).

This arrays stores the transparencies and gray levels for each rectangle.

This for loop save an image in the folder of the project after every iteration – in order to show a step-by-step creation of the rectangles

11.4 Code of the “main” function and set of intermediate images-part 2

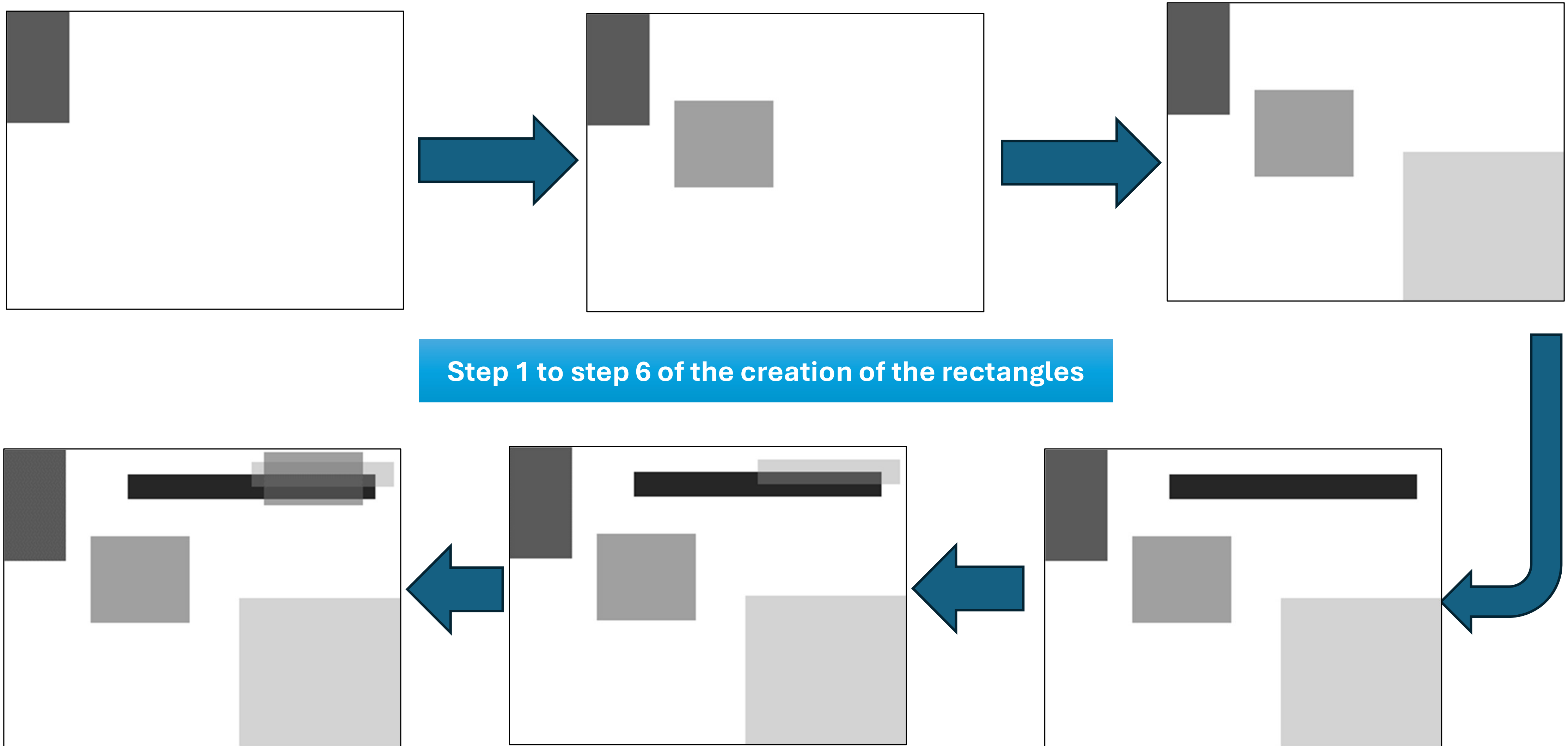


```
    // Save the final image
    StoreGrayImageAsGrayBmpFile(img, "grayImg_final.bmp");

    // Wait for user to press a key
    WaitForUserPressKey();

    return 0;
}
```

Calls the function StoreGrayImageAsGrayBmpFile to save the image img as a BMP file named "grayImg11.bmp".

11.4 Code of the “main” function and set of intermediate images-part 3

series of intermediate images created in the main step by step with proper explanations



Step 1 to step 6 of the creation of the rectangles

11.5 what did we learned?

We learned how to create a base image.

We learned how to draw on an existing image.

We learned about the use of transparency and gray level of an image

We learned to find the coordinate (0,0) on the screen

We learned to create a rectangle of gray levels

We learned to draw several rectangles in one drawing in different places in the picture