Picture lab | Peyton Duncan

Section 1:

1. 8 bits

2. 3 bytes for RGB

3. 307,200

Section 2:

1. 255 0 255

2. 255 255 0

3. 77 0 77

4. 255 255 255

5. 125 125 125

Section 3:

1. 0

2. 0

3. 640

4. 480

5. Top to Bottom

6. Left to Right

Sec4 1: public int getCount(int value) { int count = 0; for(int i = 0; i < matrix.length; i++){ for(int j = 0; j < matrix[i].length; j++){ if(value==matrix[i][j]){ count++; } } } return count; }

Sec4 2: public int getLargest(){ int largest = 0; for(int i = 0; i < matrix.length; i++){ for(int j = 0; j < matrix[i].length; j++){ if(matrix[i][j]>largest){ largest = matrix[i][j]; } } } return largest; }

Sec4 3: public getColTotal(int columnVal){ int result = 0; for(int i = 0; i < matrix[0].length; i++){ result+= matrix[i][columnVal]; } return result; }

Sec5: 1. Nope 2. Yup 3. Nah Bruh 4. Yup 5. Yup 6. Yup 7. Nope

Exer:

3: public static void keepOnlyBlue() { Pixel[][] pixels = this.getPixels2D(); for (Pixel[] rowArray : pixels) { for (Pixel pixelObj : rowArray) { pixelObj.setRed(0); pixelObj.setGreen(0); } } }

4: public static void negate() { Pixel[][] pixels = this.getPixels2D(); for (Pixel[] rowArray : pixels) { for (Pixel pixelObj : rowArray) { pixelObj.setRed(255 - pixelObj.getRed()); pixelObj.setGreen(255 - pixelObj.getGreen()); pixelObj.setBlue(255 - pixelObj.getBlue()); } } }

5: public static void grayscale() { Pixel[][] pixels = this.getPixels2D(); int avg = 0; for (Pixel[] rowArray : pixels) { for (Pixel pixelObj : rowArray) { avg = (pixelObj.getRed() + pixelObj.getGreen() + pixelObj .getBlue()) / 3; pixelObj.setRed(avg); pixelObj.setGreen(avg); pixelObj.setBlue(avg);

}

}

}

6: public static void fixUnderWater() { Pixel[][] pixels = this.getPixels2D(); int avg = 0; for (Pixel[] rowArray : pixels) { for (Pixel pixelObj : rowArray) { pixelObj.setRed(20); pixelObj.setGreen(20); pixelObj.setBlue(200);

}

}

}

Sec6:

6:

public static void fixUnderWater() {

- Pixel[][] pixels = this.getPixels2D();

- int avg = 0;

- for (Pixel[] rowArray : pixels) {

- for (Pixel pixelObj : rowArray) {

- pixelObj.setRed(20);

- pixelObj.setGreen(20);

- pixelObj.setBlue(200);

+ Pixel[][] pixels = this.getPixels2D();

+ int avg = 0;

+ for (Pixel[] rowArray : pixels) {

+ for (Pixel pixelObj : rowArray) {

+ pixelObj.setRed(20);

+ pixelObj.setGreen(20);

+ pixelObj.setBlue(200);

- }

}

}

+}

Sec6:

+1.

+public static void mirrorVerticalRightToLeft(){

+ Pixel[][] pixels = this.getPixels2D();

+ Pixel leftPixel = null;

+ Pixel rightPixel = null;

+ int width = pixels[0].length;

+ for(int i = 0; i < pixels.length; i++){

+ for(int j = width-1; j > width/2; j--){

+ leftPixel = pixels[i][j];

+ rightPixel = pixels[i][width-1-j];

+ rightPixel.setColor(leftPixel.getColor());

+ }

+ }

+}

+

+2.

+public static void mirrorHorizontal(){

+ Pixel[][] pixels = this.getPixels2D();

+ Pixel topPixel = null;

+ Pixel botPixel = null;

+ int height = pixels.length;

+ for(int i = 0; i < height / 2; i++){

+ for(int j = 0; j < pixels.length; j++){

+ topPixel = pixels[i][j];

+ botPixel = pixels[i][height-1-j];

+ botPixel.setColor(leftPixel.getColor());

+ }

+ }

+}

+

+3.

+public static void mirrorHorizontalBotToTop(){

+ Pixel[][] pixels = this.getPixels2D();

+ Pixel topPixel = null;

+ Pixel botPixel = null;

+ int height = pixels.length;

+ for(int i = height-1; i > height/2; i--){

+ for(int j = 0; j < pixels.length; j++){

+ topPixel = pixels[i][j];

+ botPixel = pixels[i][height-1-j];

+ botPixel.setColor(leftPixel.getColor());

+ }

+ }

+}

+

+4.

+public static void mirrorDiagonal(){

+ Pixel[][] pixels = this.getPixels2D();

+ Pixel leftPixel = null;

+ Pixel rightPixel = null;

+ int diagLength = Math.sqrt((pixels[0].length\*pixels[0].length)+(pixels.length\*pixels.length));

+ for(int row = 0; row < pixels.length; row++){

+ for(int j = 0; j < i; j++){

+ leftPixel = pixels[row][j];

+ rightPixel = pixels[row][height-1-j];

+ rightPixel.setColor(leftPixel.getColor());

+ }

+ }

+}

Sec 7

1. 90
2. 112

Exercises:

Section 7 2:

public static void mirrorArms() {

Pixel topPixel = null;

Pixel botPixel = null;

Pixel[][] pixels = new Picture("snowman.jpg").getPixels2D();

for (int row = 155; row < 191; row++) {

for (int col = 98; col < 169; col++) {

topPixel = pixels[row][col];

botPixel = pixels[191 - row + 191][col];

botPixel.setColor(topPixel.getColor());

}

}

for (int row = 155; row < 191; row++) {

for (int col = 238; col < 296; col++) {

topPixel = pixels[row][col];

botPixel = pixels[191 - row + 191][col];

botPixel.setColor(topPixel.getColor());

}

}

}

Section 7 3:

public static void mirrorGull() {

int mirrorPoint = 350;

Pixel leftPixel = null;

Pixel rightPixel = null;

Pixel[][] pixels = new Picture("seagull.jpg").getPixels2D();

for (int row = 225; row < 332; row++)

{

for (int col = 219; col < mirrorPoint; col++)

{

leftPixel = pixels[row][col];

rightPixel = pixels[row][mirrorPoint - col +

mirrorPoint];

rightPixel.setColor(leftPixel.getColor());

}

}

}

Section 8 1:

public static void copy(Picture fromPic,int startRow, int startCol, int endRow, int endCol) {

Pixel fromPixel = null;

Pixel toPixel = null;

Pixel[][] toPixels = this.getPixels2D();

Pixel[][] fromPixels = fromPic.getPixels2D();

for (int fromRow = 0, toRow = startRow; fromRow < endRow

&& toRow < toPixels.length; fromRow++, toRow++) {

for (int fromCol = 0, toCol = startCol; fromCol < endCol

&& toCol < toPixels[0].length; fromCol++, toCol++) {

fromPixel = fromPixels[fromRow][fromCol];

toPixel = toPixels[toRow][toCol];

toPixel.setColor(fromPixel.getColor());

}

}

}

Section 8 2:

public static collage(){

picture pic = new Picture(“snowman.jpg”);

for(int i = 0; I < 100; i++){

copy(pic,i,0);}

pic.explore;}

Section 9 1:

public void edgeDetection(int edgeDist){

Picture copy = new Picture(this);

Pixel leftPixel = null;

Pixel rightPixel = null;

Pixel[][] pixels = this.getPixels2D();

Color rightColor = null;

for (int row = 0; row < pixels.length; row++)

{

for (int col = 0; col < pixels[0].length - 1; col++)

{

leftPixel = pixels[row][col];

rightPixel = pixels[row][col + 1];

rightColor = rightPixel.getColor();

if (leftPixel.colorDistance(rightColor) > edgeDist) {

leftPixel.setColor(Color.BLACK);

}

else{

leftPixel.setColor(Color.WHITE);

}

}

}

Pixel[][] copyPixels = copy.getPixels2D();

Pixel bottomPixel = null;

Pixel upperPixel = null;

Color upperColr = null;

for (int row = copyPixels.length - 1; row > 0 ; row--)

{

for (int col = 0; col < copyPixels[0].length; col++)

{

upperPixel = copyPixels[row + 1][col];

bottomPixel = copyPixels[row][col];

upperColr = upperPixel.getColor();

if (bottomPixel.colorDistance(upperColr) > edgeDist)

{

pixels[row][col].setColor(Color.BLACK);

}

}

}

}

Section 9 part 2:

public static void edgeDetect(double thresh,Picture pic){

Color white = new Color(1,1,1);

Color black = new Color(0,0,0);

Pixel top = null;

Pixel bottom = null;

double topInt;

double botInt;

for(int i = 0; i < pic.getHeight()-1; i++){

for(int j = 0; j < pic.getWidth(); j ++){

top = pic.getPixel(i,j);

bottom = pic.getPixel(i,j+1);

topInt = (top.getRed() + top.getGreen() + top.getBlue())/3;

botInt = (bottom.getRed() + bottom.getGreen() + bottom.getBlue())/3;

if(Math.*abs*(topInt-botInt) < thresh){

top.setColor(white);

}

else{

top.setColor(black);

}

}

}

}