Relatório Atividade 1

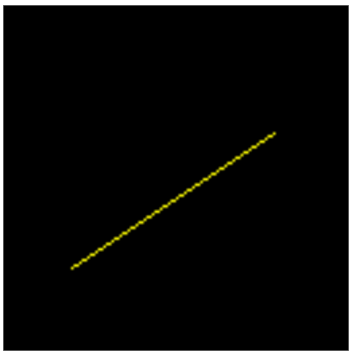
Samantha Dantas Medeiros **20190145598**

Desenvolvimento da Atividade

A atividade foi divida em duas partes, sendo a primeira voltada para a implementação de duas funções que são capazes de desenhar linhas por todo o plano permitindo que as linhas possam funcionar em todos os octantes. Para que fosse desenvolvido, foi disponibilizado um repositório do CodePen no qual contém uma classe **Canva** que é responsável pela coloração dos pixels acesos e as funções `MidPointLineAlgorithm()` e `DrawTriangle()` a serem implementadas. Enquanto que a segunda parte, após implementar as funções anteriores, exigiu a criação de uma imagem de tema livre utilizando as funções implementadas.

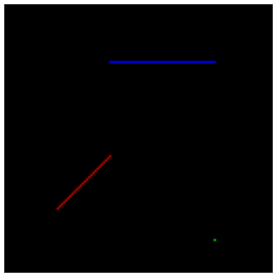
Parte 1: Implementação de funções

Para a função `MidPointLineAlgorithm()` utilizei o algoritmo de *Bresenham* visto nas aulas 5 e 6 da disciplina, cujam tratavam da rasterização de linhas em um plano. O algoritmo ascendia pixels na tela conforme o incremento da variável de decisão, esta podia permitir a movimentação para Leste ou Nordeste do plano, a cada incremento um pixel seria aceso. Com a escrita dessa parte do algoritmo feita, realizei um teste com as coordenadas fornecidas, o resultado pode ser observado a seguir (IMG1), uma linha foi traçada e colorida em amarelo (sem gradientes, por enquanto.



IMG1 - Implementação `MidPointLineAlgorithm(25, 30, 100, 80, [255,255,0,255], [255,0,0])`.
Em cor sólida, sem gradiente.

Em seguida decidi testar a função que desenha triângulos. Sua implementação consiste em apenas chamar a função anterior 3 vezes, de forma que na primeira chamada fosse traçada uma linha entre os pontos P0(25,30) e P1(50,100), depois P1(50,100) e P2(100,15) e por fim P2(100,15) e P0(25,30). Os resultados obtido foi desencontro de linhas(IMG2), ou seja, não foi formado um triângulo. Em outras palavras o algoritmo de ponto médio precisa de mais implementação.



IMG2 - testando o algoritmo de ponto médio com os pontos do triângulo

```
function DrawTriangle(x0, y0, x1, y1, x2, y2, color_0, color_
1, color_2) {
  MidPointLineAlgorithm(x0, y0, x1, y1, color_0, color_1);
  MidPointLineAlgorithm(x1, y1, x2, y2, color_1, color_2);
  MidPointLineAlgorithm(x2, y2, x0, y0, color_2, color_0);

}
DrawTriangle(25, 30, 50, 100, 100, 15, [255,0,0,255], [0,0,25
5,255], [0,255,0,255]);
```

Para melhorar o algoritmo tive de me atentar a alguns detalhes:

- A todas as direções possíveis que as linhas podiam seguir, não apenas Leste e Nordeste;
- Às variáveis Δx e Δy do algoritmo de *Bresenham* que correspondem ao número de iterações para ascender um pixel em um eixo
- E como eu poderia lidar com o incremento das variáveis `x` e `y` caso Δx e Δy fossem negativos.

Para o meu 1º e 2º ponto, eu precisei comparar os valores de Δx e Δy, se $\Delta x > \Delta y$ ou se $\Delta x < \Delta y$ e dentro dessas estruturas de decisão eu inclui um laço de repetição em cada um no qual seria responsável pela interação e pela coloração dos pixels em tela. Para a condição de `dx > dy` eu deixei as instruções similares ao algoritmo inicial de *Bresenham* visto em aula(e disponibilizado como material de leitura), sendo assim, no meu `else` eu só fiz inverter um pouco o cálculo da variável de decisão `d` e quem iria ser incrementado(neste caso, o y sempre seria incrementado).

```
if (dx > dy) {
  d = 2 * dy - dx; // decisão para o 1º pixel

  for (i = 0; i < dx; i++){
    // pinta o pixel de coordenadas (x,y)
    color_buffer.putPixel(x,y,color_0);
    if(d < 0){
      d += 2 * dy; // caminha para leste
    } else {
      y += iy;
      d += 2 * (dy - dx); // caminha para nordeste
    }
    x+=ix; // sempre incrementa o x
  }
}

else {
  d = 2 * dx - dy; // decisão para o 1º pixel
```

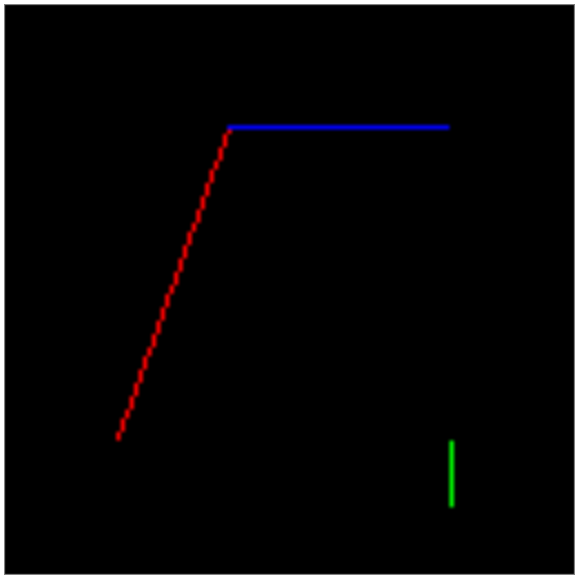
```
  for(i = 0; i < dy; i++){
    // pinta o pixel de coordenadas (x,y)
    color_buffer.putPixel(x,y,color_0);
    if(d < 0) {
      d += 2 * dx; // caminha para leste
    } else {
      x += ix;
      d += 2 * (dx - dy); // caminha para nordeste
    }
    y+=iy; // sempre incrementa o y
  }
}
```

Em relação ao meu 3º ponto, criei duas variáveis auxiliares `ix` e `iy` que servirão para incrementar as variáveis `x` e `y` a depender do valor das variáveis `dx` e `dy`.

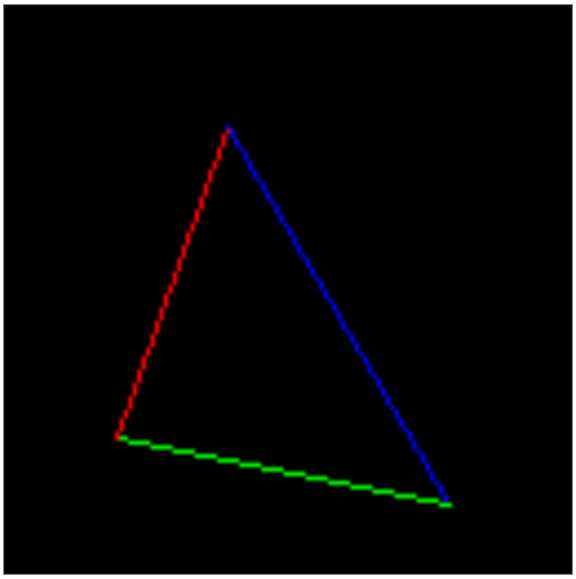
```
// incremento, auxiliar
var ix = 1;
var iy = 1;

// verificando o resultado da variação das coordenadas x e y
/*
Os valores de x e y serão incrementados ou decrementados de acordo com o valor
das variáveis de distância do eixo x(dx) e eixo y(dy)
*/
if (dx < 0) {
  ix = -ix;
}
if (dy < 0) {
  iy = -iy;
}
```

Em seguida testei o meu algoritmo mais uma vez e já obtive alguma melhora no resultado como pode ser observado na imagem 3, duas linhas conseguiram se encontrar. Para resolver isso, apenas obtive os valores absolutos das variáveis `dx` e `dy` utilizando a função `Math.abs()` que resultou na normalização das linhas desenhadas. Testando mais uma vez, consegui fazer com que as demais linhas se encontrassem e finalmente formassem o triângulo esperado(IMG4).

```
dx = Math.abs(dx);
dy = Math.abs(dy);
```



IMG3 - sem valor Absoluto.



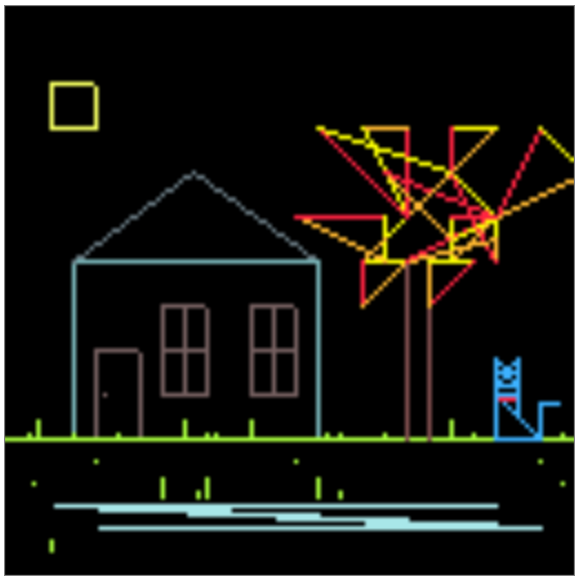
IMG4 - com valor absoluto.

Por fim ficou apenas faltando a melhoria na pintura dos pixels acesos pelo algoritmo.

Parte 2: Figura de tema livre

Para a imagem de tema livre utilizei: `color_buffer.putPixel()`, `MidPointLineAlgorithm()` e `DrawTriangle()` para gerar o seguinte cenário que pode ser observado na imagem abaixo.

Para gerar apenas linhas retas e quadriláteros fiz uso da função do Algoritmo de Ponto Médio, enquanto que a função desenha triângulos foi utilizada no telhado da casa, na copa bagunçada da árvore e orelhas e corpo do gato. A função que apenas ascende pixels foi utilizada para ascender algumas gramas e adicionar uma maçaneta na porta da casa.



IMG5 - Imagem de tema livre.

IMG6 - gato azul.

Para o rosto do gato desenhei um quadrado assim como fiz para o sol que está localizado no superior esquerdo da tela. Ao adicionar dois triângulos retângulos

sobrepostos no quadrilátero resultou em um rosto engraçado.

Considerações finais da atividade

As dificuldades encontradas na atividade foram de modo geral em como **implementar o algoritmo**. Pois no começo eu não estava conseguindo enxergar como poderia implementar após notar que as linhas que eu conseguia traçar cresciam positivamente para a direita, elas não conseguiam ser traçadas para outras direções e por isso ficavam como linhas retas, sem qualquer deslocamento. Mas quando consegui abrir meus olhos para notar esses pequenos detalhes, como a distância entre um ponto e outro, consegui prosseguir.

Gostaria de ter conseguindo desenvolver a parte da **coloração do pixels em gradiente**, mas acabei me ocupando com outras atividades do período e por isso não consegui completar essa questão, mas acredito que seja utilizando algum incremento nos valores das cores RGBA, talvez colocando um limite de incremento até o valor de 255 ou até o valor da próxima cor.

Por fim a dificuldade na 2ª parte da atividade foi pensar em uma **imagem de tema livre**, mas no geral foi algo divertido na atividade em relação a uma questão similar da atividade 0.

Referências

- Aula 5: https://www.youtube.com/watch?v=x6zbuyA-0S4
- Aula 6: https://www.youtube.com/watch?v=tygja6rr62M
- Algoritmo de Bresenham: https://en.wikipedia.org/wiki/Bresenham's_line_algorithm

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/65a73b53-a046-40c4-b028-2017a59ba60d/Bresenhams_line_algorithm_-_Wikipedia.pdf