基礎プログラミングおよび演習 レポート ＃06

1920031, 山川竜太郎 (ペア: 1920003:伊東隼人,1720031:倉橋和孝)

2019/12/01

1　構想・計画・設計

(どのような構想で絵を生成したか、具体的にどのように計画し、プログラムはどう設計したか)

絵を作成するならドット絵の要領でｘとｙ軸を指定するだけで良い。しかしそれだとコンピューターを使用して計算して絵を出力するという講義の内容から外れてしまうので、自動で絵を生成してランダム性を内包した絵を作成することを目標にした。計画としては、手書きでゴーストはどのようなパーツで構成されているのか分解して分解して使用するメソッドを考えた。プログラムの設計は、最初に日本語で何を実装するのか書き出した。具体的には、パーツ単位でメソッドを作成するようにした。それを一つの中心的なメソッドで動かす。メソッドは数値などをなるべくハードコードしないように、変数によって挙動を変えるようにした。

2　プログラムコード

packman.rb という名前で作成した。

```
Pixel = Struct.new(:r, :g, :b)
$img = Array.new(200) do
  Array.new(300) do
    Pixel.new(255, 255, 255)
  end
end

def pset(x, y, r = 0, g = 0, b = 0, a = 0.0)
  if x < 0 || x >= 300 || y < 0 || y >= 200 then
    return
```

```
  end
  $img[y][x].r = ($img[y][x].r * a + r * (1.0 - a)).to_i
  $img[y][x].g = ($img[y][x].g * a + g * (1.0 - a)).to_i
  $img[y][x].b = ($img[y][x].b * a + b * (1.0 - a)).to_i
end

def writeimage(name)
  open(name, "wb") do |f|
    f.puts("P6\n300 200\n255")
    $img.each do |a|
      a.each do |p|
        f.write(p.to_a.pack("ccc"))
      end
    end
  end
  return true
end

def fillcircle(x, y, rad, r = 0, g = 0, b = 0, a = 0.0)
  j0 = (y - rad).to_i
  j1 = (y + rad).to_i
  i0 = (x - rad).to_i
  i1 = (x + rad).to_i

  j0.step(j1) do |j|
    i0.step(i1) do |i|
      if (i - x) ** 2 + (j - y) ** 2 < rad ** 2
        if block_given? then
          yield(i, j)
        else
          pset(i, j, r, g, b, a)
        end
      end
    end
  end
end

# rectangle の略
# w:width の略、幅という意味
```

```
# h:height の略、高さという意味
def fillrect(x, y, w, h, r = 0, g = 0, b = 0, a = 0.0)
  j0 = (y - 0.5 * h).to_i
  j1 = (y + 0.5 * h).to_i
  i0 = (x - 0.5 * w).to_i
  i1 = (x + 0.5 * w).to_i

  j0.step(j1) do |j|
    i0.step(i1) do |i|
      if block_given? then
        yield(i, j)
      else
        pset(i, j, r, g, b, a)
      end
    end
  end
end

# 楕円
def fillellipse(x, y, rx, ry, r = 0, g = 0, b = 0, a = 0.0)
  j0 = (y - ry).to_i
  j1 = (y + ry).to_i
  i0 = (x - rx).to_i
  i1 = (x + rx).to_i

  j0.step(j1) do |j|
    i0.step(i1) do |i|
      if ((i - x).to_f / rx) ** 2 + ((j - y).to_f / ry) ** 2 < 1.0
        if block_given? then
          yield(i, j)
        else
          pset(i, j, r, g, b, a)
        end
      end
    end
  end
end

# --- ここからパックマンの敵、ゴーストを作るためのメソッド集
```

```
# 凸型に塗りつぶすメソッド
# 正方形を4つ並べる
# x,yは3辺が正方形に面している正方形の対角線の軸を指定する
#   □
#  □■□ この黒い正方形のこと
def fill_convex(x, y, h, r = 0, g = 0, b = 0, a = 0.0)
  fillrect(x, y, h, h, r, g, b, a)
  # 上
  fillrect(x, y - h, h, h, r, g, b, a)
  # 左
  fillrect(x - h, y, h, h, r, g, b, a)
  # 右
  fillrect(x + h, y, h, h, r, g, b, a)
end

def two_convex(x, y, w, r = 0, g = 0, b = 0, a = 0.0)
  # 足の太さは4px
  2.times do |i|
    # wを二等分して、その中の1/3の部分にxを打つ
    fill_convex(x - w / 3 + w / 3 * 2 * i, y, 4, r, g, b, a)
  end
end

# nはゴーストの数
def ghosts(n = 1)
  w = 60 # 胴体部分に当たる長方形の横の長さ
  h = 30 # 胴体部分に当たる長方形の縦の長さ
  s = 4 # ゴーストの両足に当たる凸を構成する正方形4つのうちの1辺の長さ
  rx = 6
  ry = 8

  # 赤、ピンク、水色、オレンジ
  red = [255, 0, 0] # 赤
  pink = [255, 182, 193] # ピンク
  light_blue = [0, 255, 255] # 水色
  orange = [255, 165, 0] # オレンジ

  n.times do
```

```
    x = rand((w / 2)..(300 - w / 2)) # 胴体部分に当たる長方形の x 軸
    y = rand((10 + h / 2)..(200 - h / 2)) # 胴体部分に当たる長方形の y
軸
    color = [red, pink, light_blue, orange].sample

    # まずゴーストの胴体である長方形を配置する
    fillrect(x, y, w, h, color[0], color[1], color[2], 0.0)
    # ゴーストの頭となるような楕円を配置する
    fillellipse(x, y - h / 2, w / 2, 10, color[0], color[1], color[2], 0.0)
    # 足を切り抜く
    two_convex(x, y + h / 2 - s / 2, w, 255, 255, 255, 0.0)
    fillrect(x, y + h / 2 - s, s * 2, s * 2, 255, 255, 255, 0.0)
    # ゴーストの目となるような楕円を配置する
    # 左目
    fillellipse(x - w / 4, y - h / 4, rx, ry, 255, 255, 255, 0.0)
    # 目玉
    fillcircle(x - w / 4 - rx / 2, y - h / 4 + ry / 3, 3, 0, 0, 0, 0.0)
    # 右目
    fillellipse(x + w / 4, y - h / 4, rx, ry, 255, 255, 255, 0.0)
    # 目玉
    fillcircle(x + w / 4 - rx / 2, y - h / 4 + ry / 3, 3, 0, 0, 0, 0.0)
  end
  writeimage(__FILE__.match(%{(^.*).rb})[1] + ".ppm")
end

ghosts 10
```

ruby packman.rb で実行する。

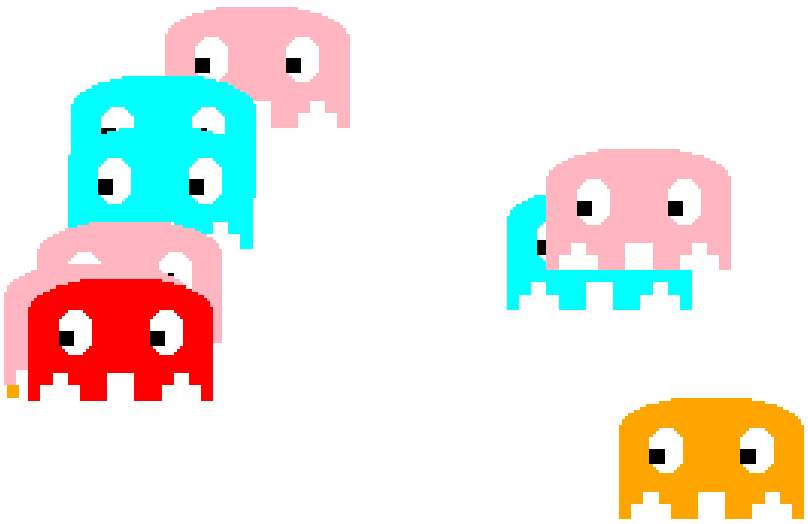
3 プログラムの説明

ghosts メソッドは、引数として与えられた数のゴーストをキャンパスのどこかに配置する。ゴーストは長方形、楕円、真円、正方形、凸の集合で構成されており、それぞれ対応するメソッドを呼び出してゴーストを構成する。長方形、楕円、真円、正方形についての生成は講義で行なった通りの生成方法である。凸に関しては正方形の集合体と考え fill_convex メソッドを新たに作成して凸を生み出す

ようにした。どこにゴーストが出現するかは、ゴーストが見切れない範囲でランダムに出現するようになったいる。またゴーストの色についても 4 色のうちからランダムに選ばれるようになっている。

4　生成された絵

パックマンの敵であるゴーストというキャラクターが色と座標がランダムで 10 体出力される。ゴーストの構成は長方形、楕円、正方形の組み合わせで出力される。



5　考察

事前準備として今回は 3 人で共同で開発するということで、ソースコードの共有の方法を GitHub を用いて共有することに決めた。その結果、ソースコードの連携が滞りなく完了した。共同開発なのでプログラムの理解力が均一になるようにコメントを残すようにした。3 人の共同開発で、別の絵を生成しなくてはいけないので、実行ごとに違う絵が出るようにするのが良いと思った。そのためゴーストを表示するときにゴーストの中心部分の x,y 座標をキャンパスの範囲内でランダムに選ぶようにした。そうすると全員が全く違う絵が出るようになった。ランダムに x,y を選ぶと開発途中にゴーストが範囲外に出る事象が発生した。見切れるのは見栄えが悪いため、範囲内に入るように x,y をランダムで選ぶときにゴーストの一部が範囲外にかからないよう範囲を計算した。具体的な計算式は以下の通りである。

```
x = rand((w / 2)..(300 - w / 2)) # 胴体部分に当たる長方形の x 軸
y = rand((10 + h / 2)..(200 - h / 2)) # 胴体部分に当たる長方形の y 軸
```

rand メソッドに Range オブジェクトを渡すを渡してその中で x,y を選択してもらった方が効率的だと思いそのような実装にした。色をゴースト一体ごとに変更する必要があり、実装に悩んだ。実装したのは各色の RGB 値を、それぞれ持っていく。そしてそれを 2 次元配列の形にしておき Array#sample メソッドで一つの色を選ぶようにした。こうすることによって各メソッドに引数として RGB 値を渡すときに共同のコードで実行することができるようになる。具体的には下記のような変数 color にランダムに色の配列が入るので、color[0] が R の値、color[1] が G の値というような書き方にすればどの色であってもメソッドに渡す RGB 値は変更が必要ない。結果として余分なコードを書く必要がなくなった。

```
  # 赤、ピンク、水色、オレンジ
  red = [255, 0, 0] # 赤
  pink = [255, 182, 193] # ピンク
  light_blue = [0, 255, 255] # 水色
  orange = [255, 165, 0] # オレンジ
# 省略
    color = [red, pink, light_blue, orange].sample
# 省略
    fillrect(x, y, w, h, color[0], color[1], color[2], 0.0)
```

ゴーストの生成は図形の集合である。ここで問題になるのはどの図形をどのように組み合わせるかである。胴体、頭、左右の足、真ん中の足、両目というパーツに分割して考えることによってゴーストを表現できると考えた。どの座標が基準になってもいいように、x 軸と y 軸さえ変数として与えてあげれば、形を保ったまま移動できるようにすることを優先として実装するようにした。生成方法は胴体は長方形で作成することにした。この長方形の対角線の交点はゴーストの基準軸として考えることにする。長方形の上に楕円を重ねることにより頭とすることにした。楕円の中心点 (x,y) は、x 軸は同一、y 軸は胴体の長方形の上辺の高さと考えることで胴体の (x,y) と高ささえわかれば自由に移動できるようになる。ゴーストは 3 つの足をもち、凸でくり抜かれた間に正方形がある。凸を生成するため、演習で出てきた正方形を作成するメソッドを応用して fill_convex メソッドを作成した。生成方法については、サイコロの展開図を想像して作成した。凸は三片が同一の正方形に隣接した正方形 4 つの集合体と考える。真ん中の正方形に対して、(x,y) と 1 辺の長さをを与えてあげてあげれば、そこから計算して凸を作成することができた。

```
def fill_convex(x, y, h, r = 0, g = 0, b = 0, a = 0.0)
```

この凸を左右の等間隔に並べなくてはならないため、fill_convex メソッドをラップした two_convex メソッドを定義した。two_convex メソッドのコメントに

記述していると通り胴体の横の長さを 6 等分して、1/6 の部分と 5/6 の部分が x 軸の点になるようにした。これも動的にゴーストを動かすために必要なことであった。両目に対しては、胴体の高さの高さの 3/4 に白目の中心になるようにした。以上で胴体の長方形の基準点である (x,y) 座標がわかり、なおかつ長方形の横と縦の長ささえ変数で与えてあげればおおよそ全てのパーツを求めることができた。前述したように胴体の (x,y) をランダムに選択しているようになるのでゴーストはキャンパス内を自由に移動できるようになった。ゴーストの数についても変動できるよう ghosts メソッドに数字を引数として与えることにより、その数の回数ゴーストの生成を繰り返すようにして、人によってゴーストの出力数を変更できるよう工夫した。

6 アンケート

6.1 Q1：画像が自由に生成できるようになりましたか。

円の生成をすることが難しかったです。

6.2 Q2：画像をうまく生成する「コツ」は何だと思いましたか。

まず自然言語でどのように絵を作成するのか言語化することだと思います。

6.3 Q3：リフレクション (今回の課題で分かったこと)・感想・要望をどうぞ。

画像の作り方がわかりました。