Języki skryptowe

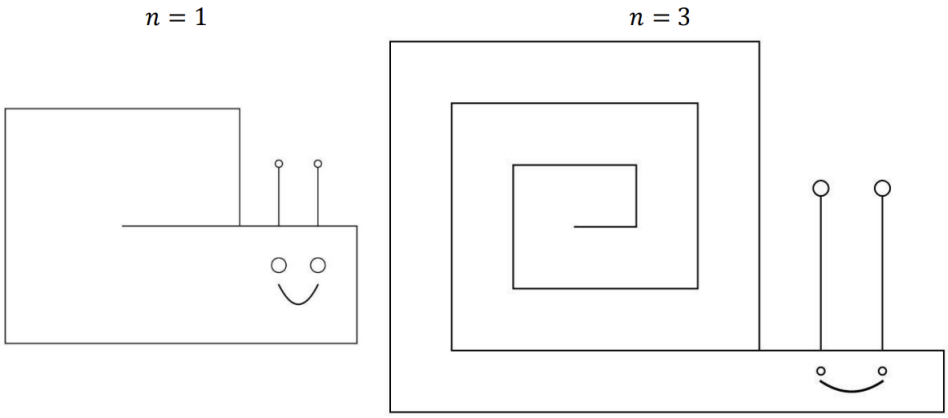
Tomasz Tomaszek, grupa 1/2

3 stycznia 2024

Część I

Opis programu

Należało napisać program, który dla zadanej liczby naturalnej n rysować będzie ślimaka. Liczba pętli muszli ślimaka wyznaczana jest przez wartość n, długość głowy ślimaka (część na prawo od muszli) jest równa połowie długości muszli (mierząc tę muszlę na górze). Czułki ślimaka mają wysokość równą połowie muszli licząc od górnej części głowy ślimaka. Oczy na czułkach są okręgami o promieniu proporcjonalnym do n (proporcjonalność tę można było dobrać według własnego uznania). Oczy na głowie są okręgami, których środki wyznaczone są przez położenie czułek (współrzędne n) oraz $\frac{1}{3}$ wysokości głowy i mają promień stały (niezależny od n). Uśmiech ślimaka jest fragmentem paraboli, której wierzchołek leży po środku oczu (współrzędna n) i w $\frac{1}{3}$ wysokości głowy licząc od dołu tej głowy (współrzędna n), a współczynnik n tej paraboli dobrany jest tak, że uśmiech kończy się w połowie głowy (współrzędna n) natomiast ślimak uśmiecha się „od oka do oka" (współrzędna n). Na rysunkach niezachowane są proporcje, ogólnie, dla $n = 1$ muszla ślimaka (mierząc ją na górze) ma długość 2 (rysunek lewy), a dla $n = 3$ muszla ślimaka (mierząc ją na górze) ma długość 6 (rysunek prawy).



Rysunek 1: Przykładowe ślimaki

Część II

Opis działania

Program wykorzystuje bibliotekę PIL, która umożliwia tworzenie obrazu. W skład struktury programu wchodzi klasa Snail oraz metody rysujące poszczególne części ślimaka. Program po podaniu przez użytkownika liczby całkowitej rysuje w oknie o rozdzielczości 1920x1080 ślimaka o wymaganych parametrach. Tło jest białe a kolor rysunku czarny. Metody rysujące tworzą ślimaka z linii, okręgów oraz paraboli (parabolą jest w zasadzie zbiór punktów należących do wykresu funkcji kwadratowej). Jako długość początkowego segmentu obrałem 50px. Poniżej przedstawiam algorytmy wykorzystane przy rysowaniu ślimaka.

drawShell()

Metoda rysująca muszelkę ślimaka. Współrzędne początkowe obliczane są ze wzorów:

$$x = 50 + n \cdot 50, y = 50 + n \cdot 50$$

Współrzędne "środka"muszli zależą od liczby n.

Data: Współrzędne początkowe i końcowe: $prev_x, prev_y, next_x, next_y$
Liczba całkowita n
Długość segmentu s
for n **do**
  $next_x += s$
  rysuj linię
  $prev_x = next_x$
  $next_y -= s$
  rysuj linię
  $s += 50px$
  $next_x -= s$
  $prev_y = next_y$
  rysuj linię
  $prev_x = next_x$
  $next_y += s$
  rysuj linię
  $prev_y = next_y$ $s += 50px$
end
$next_x = s - 50px$
rysuj linię

drawHead()

Metoda rysująca głowę ślimaka. Współrzędne początkowe są współrzędnymi końcowymi z metody drawShell().

Data: Współrzędne początkowe i końcowe: $prev_x, prev_y, next_x, next_y$
Długość segmentu s
rysuj linię
$next_y -= 50$
rysuj linię
$prev_y = next_y$
$next_x -= \frac{s}{2}$
rysuj linię

drawEyes()

Metoda rysująca oczy ślimaka. Współrzędne środka są obliczane ze wzorów:

$$center_x = prev_x - \frac{s}{6}, center_y = prev_y + \frac{50}{3}$$

Data: Współrzędne środka okręgu: $center_x, center_y$
Długość segmentu s
rysuj okrąg
$center_x -= \frac{s}{6}$
rysuj okrąg

drawAnthenas()

Metoda rysująca czułki ślimaka. Współrzędne początkowe są obliczane ze wzorów:

$$x = prev_x - \frac{s}{6}, y = prev_y$$

Promień czułek jest obliczany ze wzoru:

$$(2 + n)px$$

Data: Współrzędne początkowe, końcowe i promień czułek:
$prev_x, prev_y, next_x, next_y, r$
Długość segmentu s
$next_y = prev_y - \frac{s-100}{2}$
rysuj linię
$next_y - = r$
rysuj okrąg
$prev_x - = \frac{s}{6}$
$next_y + = r$
$next_x = prev_x$
rysuj linię
$next_y - = r$
rysuj okrąg

drawSmile()

Metoda rysująca uśmiech ślimaka. Parabola jest w rzeczywistości zbiorem punktów należących do funkcji $y = ax^2$. Współrzędne wierzchołka paraboli obliczane są ze wzoru:

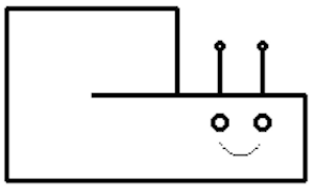
$$x = prev_x - \frac{s}{4}$$

$$y = prev_y + 35$$

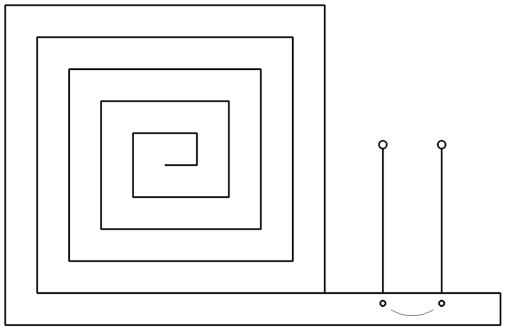
Współczynnik a został dobrany rozwiązując poniższy układ równań. Wartości 0.05 oraz 0.001 były odpowiednie dla $n = 1$ oraz $n = 10$. Dobranie odpowiedniego wzoru funkcji było zdecydowanie najtrudniejszą cześcią projektu.

$$\begin{cases} 0.05 = ab \\ 0.001 = ab^{10} \end{cases}$$

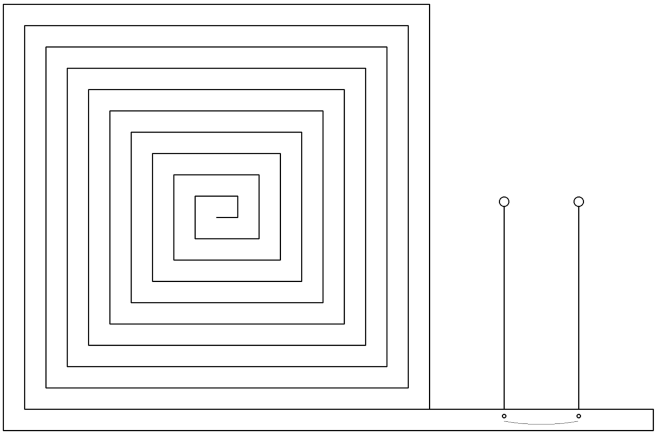
Program w zakresie od $-\frac{s}{12}$ do $\frac{s}{12}$ wyznacza punkty, które tworzą parabolę ax^2. Współczynnik $a = 0.077222605 \cdot 0.647478803^n$. Dla każdej wielkości ślimaka współczynnik a będzie przybierał inną wartość.



Rysunek 2: Dla $n = 1$



Rysunek 3: Dla $n = 5$



Rysunek 4: Dla $n = 10$

Pełen kod aplikacji

```
# Biblioteka PIL umozliwia tworzenie rysunkow

from PIL import Image, ImageDraw

# Klasa snail

class Snail:

    # Konstruktor obiektu

    def __init__(self, n):
        self.n = n # Atrybut n
        self.segment_length = 50 # Jednostka dlugosci n
        self.eye_radius = 5
        self.anthenas_radius = 2 + self.n
        self.image = Image.new("RGB", (1920, 1080), "white") # Utworzenie rysunku
        self.draw = ImageDraw.Draw(self.image) # Metoda rysujaca
        self.position_x = (50 + self.n * 50) # Aktualna pozycja "rysowania"
        self.position_y = (50 + self.n * 50)

    # Metoda rysujaca muszle

    def drawShell(self):
        prev_x = self.position_x # Zmienne pozycji poczatkowej linii
        prev_y = self.position_y
        next_x = prev_x # Zmienne pozycji koncowej linii
        next_y = prev_y
        for _ in range(self.n): # Petla rysujaca muszle
            next_x += self.segment_length # Do koncowej wspolrzednej x dodajemy dlugosc segmentu
            self.draw.line([prev_x, prev_y, next_x, prev_y], fill = "black", width = 3) # Rysowanie linii
            prev_x = next_x # Aktualizacja wspolrzednej x
            next_y -= self.segment_length # Aktualizacja wspolrzednej y
            self.draw.line([prev_x, prev_y, next_x, next_y], fill = "black", width = 3) # Rysowanie linii
            self.segment_length += 50
            next_x -= self.segment_length
            prev_y = next_y
            self.draw.line([prev_x, prev_y, next_x, next_y], fill = "black", width = 3)
            prev_x = next_x
            next_y += self.segment_length
            self.draw.line([prev_x, prev_y, next_x, next_y], fill = "black", width = 3)
            prev_y = next_y
            self.segment_length += 50
        next_x += self.segment_length - 50
        self.draw.line([prev_x, prev_y, next_x, next_y], fill = "black", width = 3)
```

```
45          prev_x = next_x
46          self.position_x = prev_x
47          self.position_y = prev_y
48
49      # Metoda rysujaca glowe
50
51      def drawHead(self):
52          prev_x = self.position_x
53          prev_y = self.position_y
54          next_x = prev_x + self.segment_length / 2
55          next_y = prev_y
56          self.draw.line([prev_x, prev_y, next_x, next_y], fill = "black",
                width = 3)
57          prev_x = next_x
58          next_y -= 50
59          self.draw.line([prev_x, prev_y, next_x, next_y], fill = "black",
                width = 3)
60          prev_y = next_y
61          next_x -= self.segment_length / 2
62          self.draw.line([prev_x, prev_y, next_x, next_y], fill = "black",
                width = 3)
63          self.position_x = prev_x
64          self.position_y = prev_y
65
66      # Metoda rysujaca oczy
67
68      def drawEyes(self):
69          center_x = self.position_x - self.segment_length / 6
70          center_y = self.position_y + 50 / 3
71          self.draw.ellipse((center_x - self.eye_radius, center_y - self.
               eye_radius, center_x + self.eye_radius, center_y + self.
               eye_radius), outline="black", width=3)
72          center_x -= self.segment_length / 6
73          self.draw.ellipse((center_x - self.eye_radius, center_y - self.
               eye_radius, center_x + self.eye_radius, center_y + self.
               eye_radius), outline="black", width=3)
74
75      def drawAnthenas(self):
76          prev_x = self.position_x - self.segment_length / 6
77          prev_y = self.position_y
78          next_x = prev_x
79          next_y = prev_y - ((self.segment_length - 100) / 2)
80          self.draw.line([prev_x, prev_y, next_x, next_y], fill = "black",
                width = 3)
81          next_y -= self.anthenas_radius
82          self.draw.ellipse((next_x - self.anthenas_radius, next_y - self.
               anthenas_radius, next_x + self.anthenas_radius, next_y + self
               .anthenas_radius), outline="black", width=3)
83          prev_x -= self.segment_length / 6
84          next_y += self.anthenas_radius
85          next_x = prev_x
86          self.draw.line([prev_x, prev_y, next_x, next_y], fill = "black",
                width = 3)
87          next_y -= self.anthenas_radius
88          self.draw.ellipse((next_x - self.anthenas_radius, next_y - self.
```

```
            anthenas_radius, next_x + self.anthenas_radius, next_y + self
            .anthenas_radius), outline="black", width=3)

    def drawSmile(self):
        center_x, center_y = self.position_x - self.segment_length / 4,
            self.position_y + 35
        for x in range(-int((self.segment_length / 12)), int(self.
            segment_length / 12)):
            a = 0.077222605 * (0.647478803 ** self.n)
            y = int(a * x ** 2)
            if y < 10:
                self.draw.point((center_x + x, center_y - y), fill = "
                    black")

    def drawSnail(self):
        self.drawShell()
        self.drawHead()
        self.drawEyes()
        self.drawSmile()
        self.drawAnthenas()
        self.image.show()

while True:
    try:
        n = int(input("Podaj liczbe calkowita n: "))
        break
    except ValueError:
        print("To nie jest poprawna liczba calkowita. Sprobuj ponownie."
            )
snaily = Snail(n)
snaily.drawSnail()
```