|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

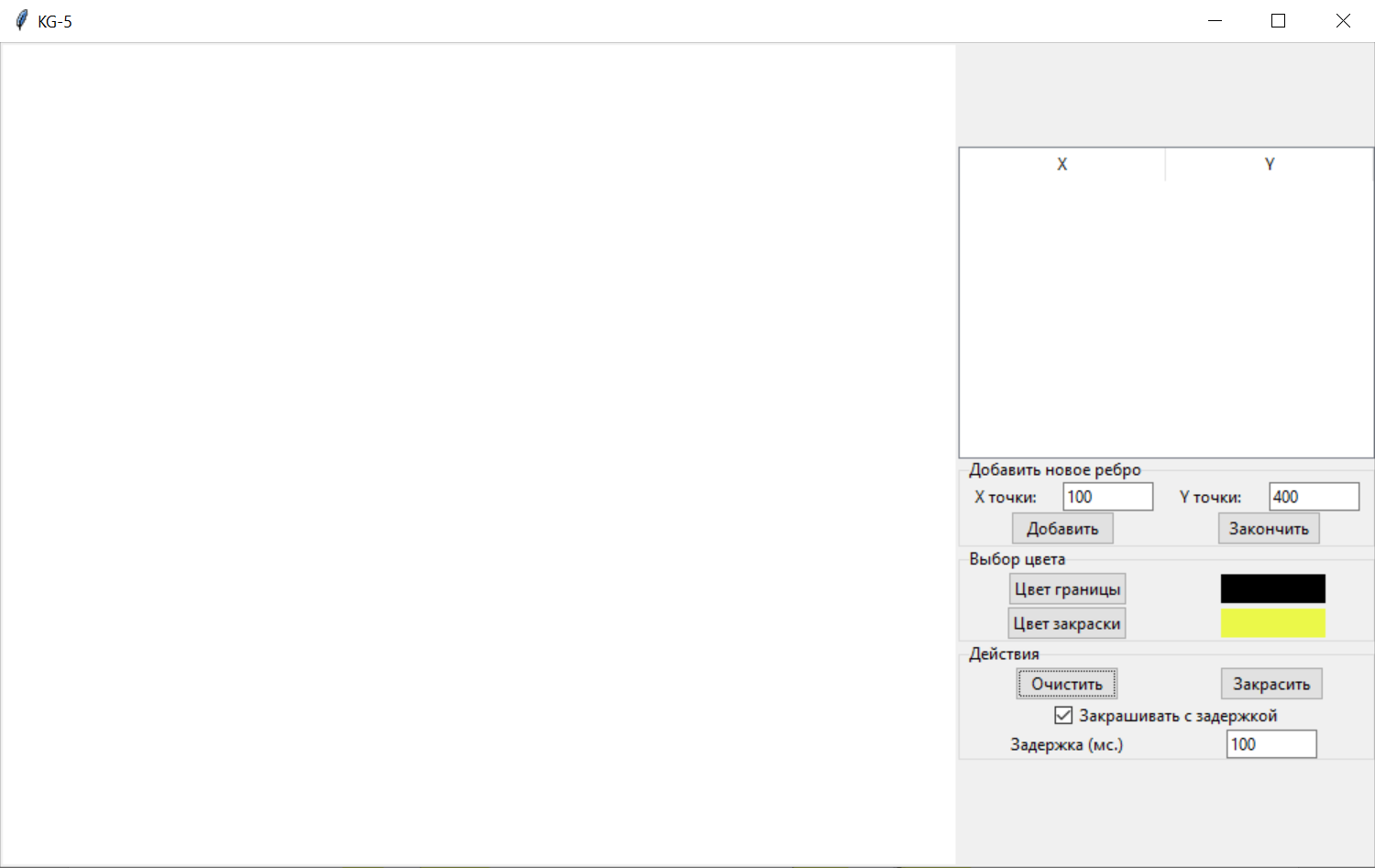
**Лабораторная работа № 5**

|  |  |
| --- | --- |
| **Тема: Реализация одного из растровых алгоритмов заполнения**  **Студент: Нгуен Ань Тхы**  **Группа: ИУ7-46Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель: Куров.А.В** |  |

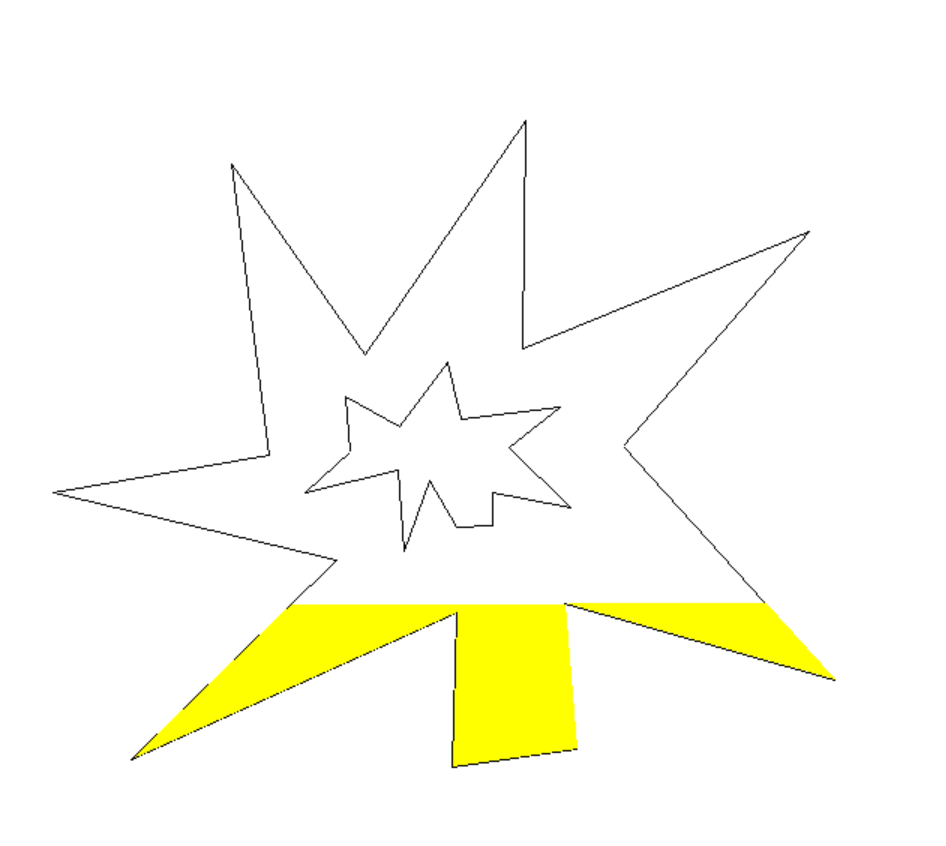
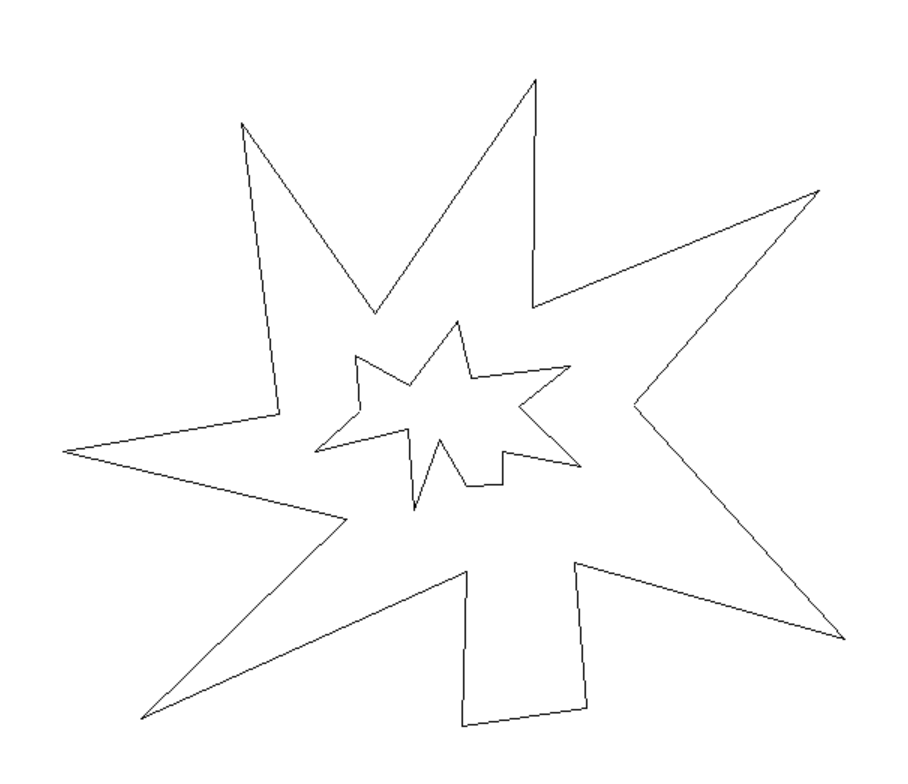
Москва.

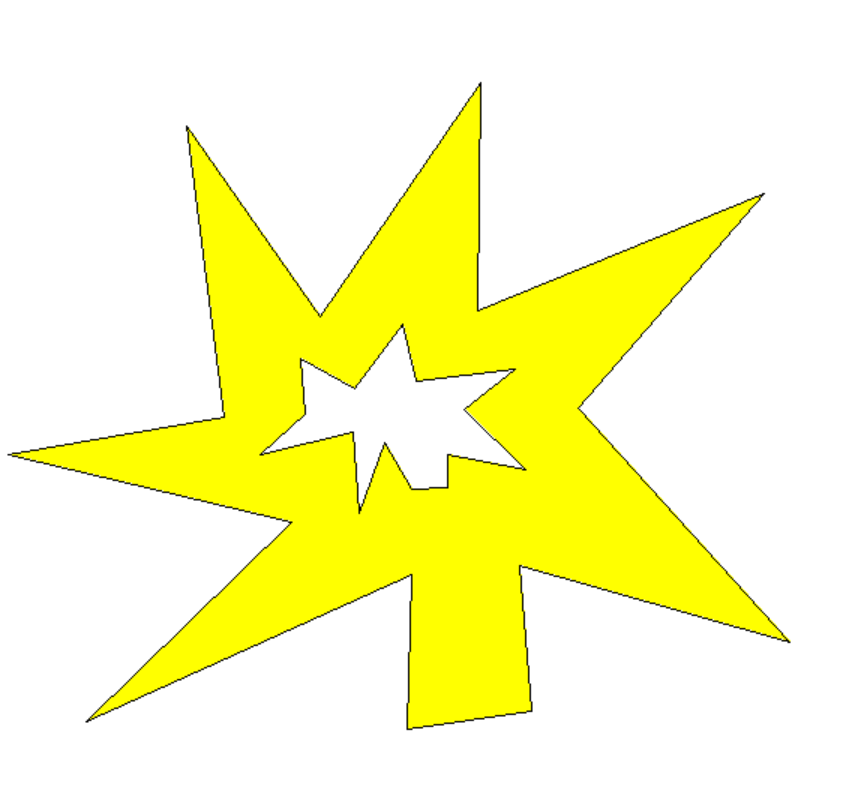
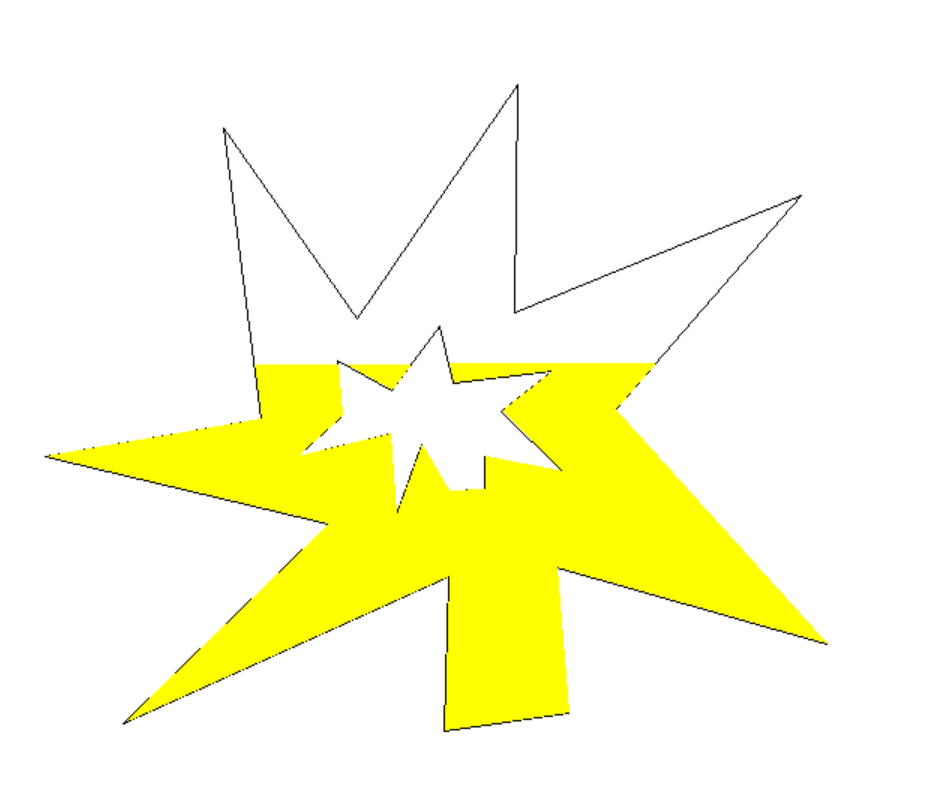
2020 г.

1. **Интерфей:**



1. **Пример выпольнения программы:**





1. **Алгоритм:**

Алгоритм с упорядоченным списком:

А) Простой вариант алгоритма:

(x1, y1)...(xn, yn)

1.Нахождение точек пересечения

2. Сортировка точек по убыванию Y

3. Сортировка по возрастанию X

4. Образование пар точек в отсортированном массиве

5. Закрашивание пикселей

Б) Болле эффективный вариант алгоритма с упорядоченном списком ребер:

1. Нахождение точек пересечения сканирующих строк с ребрами многоугольника с одновременым занесением их в соответствующую Y группу

2. Сортировка элементов каждой Y группы по возрастанию координат X

3.Образование пар пикселей в отсортированным массиве

4. Закрашивание пикселей в интервалах, организованных каждой пар пикселей

**IV. Код программы:**

#Main

import tkinter as tk

import tkinter.ttk as ttk

from tkinter import messagebox

from tkinter.colorchooser import askcolor

from operator import itemgetter

from funcs import \*

# Main window class

class Kg5App(tk.Tk):

def \_\_init\_\_(self, \*args, \*\*kwargs):

tk.Tk.\_\_init\_\_(self)

def click\_add(event):

if self.ctrl\_pressed == 0:

add(self, event.x, event.y)

else:

if len(self.figs) > self.fig\_n:

x\_prev = self.figs[self.fig\_n][-1][0]

y\_prev = self.figs[self.fig\_n][-1][1]

x = event.x - x\_prev

y = event.y - y\_prev

if abs(y) >= abs(x):

add(self, x\_prev, event.y)

else:

add(self, event.x, y\_prev)

else:

add(self, event.x, event.y)

def click\_end(event):

if len(self.figs) > self.fig\_n:

if len(self.figs[self.fig\_n]) <= 2:

clear(self, "tag" + str(self.fig\_n))

else:

end(self)

def press\_key(event):

if event.keysym == "Control\_L":

if self.ctrl\_pressed == 0:

self.ctrl\_pressed = 1

def release\_key(event):

if event.keysym == "Control\_L":

if self.ctrl\_pressed == 1:

self.ctrl\_pressed = 0

def moving\_line(event):

self.in\_canvas = 1

if self.drawing == 1:

if self.ctrl\_pressed == 0:

cur\_pos = (event.x, event.y)

self.canvas.delete("new")

self.canvas.create\_line(self.figs[self.fig\_n][-1], cur\_pos, fill=self.bd\_color, tag="new")

else:

x\_prev = self.figs[self.fig\_n][-1][0]

y\_prev = self.figs[self.fig\_n][-1][1]

x = event.x - x\_prev

y = event.y - y\_prev

if abs(y) >= abs(x):

cur\_pos = (x\_prev, event.y)

else:

cur\_pos = (event.x, y\_prev)

self.canvas.delete("new")

self.canvas.create\_line(self.figs[self.fig\_n][-1], cur\_pos, fill=self.bd\_color, tag="new")

def in\_window(event):

if self.in\_canvas == 0:

self.canvas.delete("new")

self.in\_canvas = 0

options\_size = 300 # const

can\_x = args[0] - options\_size

can\_y = args[1]

tk.Tk.title(self, "KG-5")

tk.Tk.geometry(self, str(can\_x + options\_size) + "x" + str(can\_y))

self.fill\_color = "#42f4d1"

self.bg\_color = "#fff"

self.bd\_color = "#000000"

self.fig\_n = 0 # Количество многоугольников

self.figs = [] # Хранит многоугольники

self.ctrl\_pressed = 0 # Нажат ли ctrl

self.drawing = 0 # Отображать отрезок до курсора

self.in\_canvas = 0 # Курсор находитсся в canvas

self.delay = 2 # Задержка

self.edges = [] # Массив ребер

self.process = None

self.table\_items = [] # Элементы таблицы

self.work = ttk.Frame(self)

self.table = ttk.Treeview(self.work, columns="X", height=10)

self.table.heading("#0", text="X")

self.table.heading("#1", text="Y")

self.table.column("#0", width=150)

self.table.column("#1", width=150)

self.table.grid(row=0, column=0, sticky="ns")

self.add = ttk.LabelFrame(self.work, text="Добавить новое ребро")

self.var\_x = tk.StringVar()

self.var\_y = tk.StringVar()

self.label\_x = ttk.Label(self.add, text="X точки:")

self.label\_x.grid(row=1, column=0)

self.entry\_x = ttk.Entry(self.add, textvariable = self.var\_x, width=10)

self.entry\_x.grid(row=1, column=1)

self.label\_y = ttk.Label(self.add, text="Y точки:")

self.label\_y.grid(row=1, column=2)

self.entry\_y = ttk.Entry(self.add, textvariable=self.var\_y, width=10)

self.entry\_y.grid(row=1, column=3)

self.button\_add = ttk.Button(self.add, text="Добавить", command=lambda: get\_add(self))

self.button\_add.grid(row=2, column=0, columnspan=2)

self.button\_end = ttk.Button(self.add, text="Закончить", command=lambda: click\_end(self))

self.button\_end.grid(row=2, column=2, columnspan=2)

self.add.grid(row=1, column=0, sticky="nsew")

self.add.grid\_columnconfigure(0, weight=1)

self.add.grid\_columnconfigure(1, weight=1)

self.add.grid\_columnconfigure(2, weight=1)

self.add.grid\_columnconfigure(3, weight=1)

self.color = ttk.LabelFrame(self.work, text="Выбор цвета")

self.button\_border = ttk.Button(self.color, text="Цвет границы",

command=lambda: pick\_color(self, "bd"))

self.button\_border.grid(row=0, column=0)

self.label\_bd = tk.Label(self.color, bg=self.bd\_color, width=10)

self.label\_bd.grid(row=0, column=1)

self.button\_fill = ttk.Button(self.color, text="Цвет закраски",

command=lambda: pick\_color(self, "fill"))

self.button\_fill.grid(row=1, column=0)

self.label\_fill = tk.Label(self.color, bg=self.fill\_color, width=10)

self.label\_fill.grid(row=1, column=1)

self.color.grid(row=2, column=0, sticky="nsew")

self.color.grid\_columnconfigure(0, weight=1)

self.color.grid\_columnconfigure(1, weight=1)

self.options = ttk.LabelFrame(self.work, text="Действия")

self.button\_clear = ttk.Button(self.options, text="Очистить", command=lambda: clear(self, "all"))

self.button\_clear.grid(row=0, column=0)

self.button\_fill = ttk.Button(self.options, text="Закрасить", command=lambda: fill(self))

self.button\_fill.grid(row=0, column=1)

self.delay\_var = tk.IntVar()

self.delay\_var.set(0)

self.delay\_cb = ttk.Checkbutton(self.options, text="Закрашивать с задержкой", variable=self.delay\_var)

self.delay\_cb.grid(row=1, column=0, columnspan=2)

self.delay\_label = ttk.Label(self.options, text="Задержка (мс.)")

self.delay\_label.grid(row=2, column=0)

self.delay\_str = tk.StringVar()

self.delay\_str.set(str(self.delay))

self.delay\_entry = ttk.Entry(self.options, width=10, textvariable=self.delay\_str)

self.delay\_entry.grid(row=2, column=1)

self.options.grid(row=3, column=0, sticky="nsew")

self.options.grid\_columnconfigure(0, weight=1)

self.options.grid\_columnconfigure(1, weight=1)

self.work.pack(side=tk.RIGHT)

self.canvas = tk.Canvas(self, bg=self.bg\_color)

tk.Tk.bind(self, "<KeyPress>", press\_key)

tk.Tk.bind(self, "<KeyRelease>", release\_key)

tk.Tk.bind(self, "<Motion>", in\_window)

self.canvas.bind("<Button-1>", click\_add)

self.canvas.bind("<Button-3>", click\_end)

self.canvas.bind("<Motion>", moving\_line)

self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Calls functions for filling polygon

def fill(self):

self.edges = get\_edges(self.figs)

if len(self.edges) < 3:

mes("Недостаточно ребер")

return -1

intersections = get\_intersections(self.edges)

#Sorting

intersections.sort(key=itemgetter(1, 0))

if int(self.delay\_var.get()) == 1:

try:

self.delay = int(self.delay\_str.get())

except ValueError:

mes("Неверная задержка.")

return -2

fill\_delay(self, intersections)

else:

fill\_figure(self, intersections)

draw\_edges(self)

# Picks color

def pick\_color(self, name):

color = askcolor()[1]

print(color)

if name == "bd":

self.bd\_color = color

self.label\_bd.configure(bg=color)

elif name == "fill":

self.fill\_color = color

self.label\_fill.configure(bg=color)

# Creates window with warning

def mes(text):

messagebox.showinfo("Внимание", text)

# Runs add with manual input

def get\_add(self):

try:

x = int(self.var\_x.get())

y = int(self.var\_y.get())

except ValueError:

mes("Неверные данные!")

return -1

add(self, x, y)

# Adds new dot and connects with previous

def add(self, x, y):

if len(self.figs) <= self.fig\_n:

self.figs.append([])

self.table\_items.append([])

self.figs[self.fig\_n].append((x, y))

self.table\_items[self.fig\_n].append(self.table.insert("", "end", text=str(x), values=(str(y))))

if self.drawing == 0:

self.drawing = 1

if len(self.figs[self.fig\_n]) > 1:

self.canvas.create\_line(self.figs[self.fig\_n][-1],

self.figs[self.fig\_n][-2], fill=self.bd\_color, tag="tag"+str(self.fig\_n))

# Connects last point with first

def end(self):

if len(self.figs[self.fig\_n]) > 2:

self.canvas.create\_line(self.figs[self.fig\_n][-1],

self.figs[self.fig\_n][0], fill=self.bd\_color, tag="tag"+str(self.fig\_n))

self.table\_items[self.fig\_n].append(self.table.insert("", "end", text="==========", values="=========="))

self.fig\_n += 1

self.drawing = 0

self.canvas.delete("new")

# Clears obj and resets variables

def clear(self, obj):

self.drawing = 0

self.canvas.delete(obj)

self.canvas.delete("new")

if self.process is not None:

self.canvas.after\_cancel(self.process)

self.pix\_map = []

self.edges = []

if obj == "all":

self.figs = []

self.fig\_n = 0

self.table\_items = []

self.table.delete(\*self.table.get\_children())

else:

self.figs.pop()

for item in self.table\_items.pop():

self.table.delete(item)

if \_\_name\_\_ == '\_\_main\_\_':

app = Kg5App(1000, 600)

app.mainloop()

#Algorithms

# Returns edges from list of dots

def get\_edges(dots\_mas):

edges = []

for dots in dots\_mas:

for i in range(len(dots)):

if i + 1 > len(dots) - 1:

edges.append([dots[i], dots[0]])

else:

edges.append([dots[i], dots[i + 1]])

return edges

# Returns intersections from list of edges

def get\_intersections(edges):

intersections = []

for i in range(len(edges)):

x1 = edges[i][0][0]

y1 = edges[i][0][1]

x2 = edges[i][1][0]

y2 = edges[i][1][1]

len\_x = abs(int(x2) - int(x1))

len\_y = abs(int(y2) - int(y1))

if len\_y != 0:

dx = ((x2 > x1) - (x2 < x1)) \* len\_x / len\_y

dy = ((y2 > y1) - (y2 < y1))

x1 += dx / 2

y1 += dy / 2

for j in range(len\_y):

intersections.append((x1, y1))

x1 += dx

y1 += dy

return intersections

# Fills figure instantly

def fill\_figure(self, inter):

for i in range(0, len(inter), 2):

draw\_couple(self, inter[i], inter[i + 1])

# Fills figure with delay

def fill\_delay(self, inter):

cross1 = inter.pop()

cross2 = inter.pop()

draw\_couple(self, cross1, cross2)

if len(inter) > 0:

self.process = self.canvas.after(self.delay, lambda:fill\_delay(self, inter))

else:

draw\_edges(self)

# Draws line between two dots

def draw\_couple(self, dot1, dot2):

x\_beg = int(dot1[0] + 0.5)

x\_end = int(dot2[0] - 0.5) + 1

y = int(dot1[1])

self.canvas.create\_line(x\_beg, y, x\_end, y, fill=self.fill\_color)

# Draws edges

def draw\_edges(self):

for i in range(len(self.edges)):

self.canvas.create\_line(self.edges[i][0], self.edges[i][1], fill=self.bd\_color)