Министерство высшего образования и науки РФ

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа киберфизических систем и управления

**Отчет по лабораторной работе**

по дисциплине "Кроссплатформенное программирование"

на тему: "Графический редактор"

Выполнил

студенты гр. 3530902/70201

\_\_\_\_\_\_\_\_\_\_\_ Ю. Ю. Мансуров

Проверил

Доцент, к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_ С. В. Хлопин

дата, подпись

Санкт-Петербург

2020

Содержание

Введение 4

1 Архитектура 5

2 Пользовательский интерфейс 6

3 Реализация 10

3.1 Кисти 10

3.2 Алгоритмы рисования 11

3.2.1 Pen 11

3.2.2 Line 11

3.2.3 Rect 12

3.2.4 Ellipse 12

3.2.5 Fill 12

3.3 Кроссплатформенность 12

4 Инструкция по использованию приложения 13

5 Примеры работы 14

Исходный код 17

Заключение 32

Приложение 1. Алгоритм Брезенхема для линии 33

Приложение 2. Модифицированный алгоритм Брезенхема для толстой линии 34

Приложение 3. Алгоритм построения прямоугольника 35

Приложение 4. Модифицированный алгоритм Брезенхема для круга 36

Приложение 5. Модифицированный алгоритм Брезенхема для эллипса 37

Приложение 6. Модифицированный алгоритм Брезенхема для эллипса на основе кисти 39

Приложение 7. Рекурсивный алгоритм заливки 41

# Введение

В ходе данной работы требуется реализовать графический редактор, позволяющий выбирать цвет и размер кисти для рисования. Должна присутствовать возможность рисовать некоторые примитивы (линия, круг и т.д.). Также должна быть возможность изменение размера и перемещение некоторой области.

В качестве языка программирования был использован Python 3.8.1.

## Архитектура

Программа состоит из главного класса, представляющего главное окно и содержащего реализацию функций, и классов, улучшающих графический интерфейс. Главный класс содержит алгоритм создания графического интерфейса и методы, реализующие функциональности редактора.

## Пользовательский интерфейс

Приложение имеет главное окно, имеющее следующий вид (Рис. 1), а также использует несколько диалоговых окон: окно задания размера (Рис. 2), стандартное окно выбора места сохранения (Рис. 3), стандартное окно выбора цвета (Рис. 4) и стандартное окно вывода информации (Рис. 5). Изображения стандартных диалоговых окон приведены для ОС Windows 10 и могут отличаться в других операционных системах.

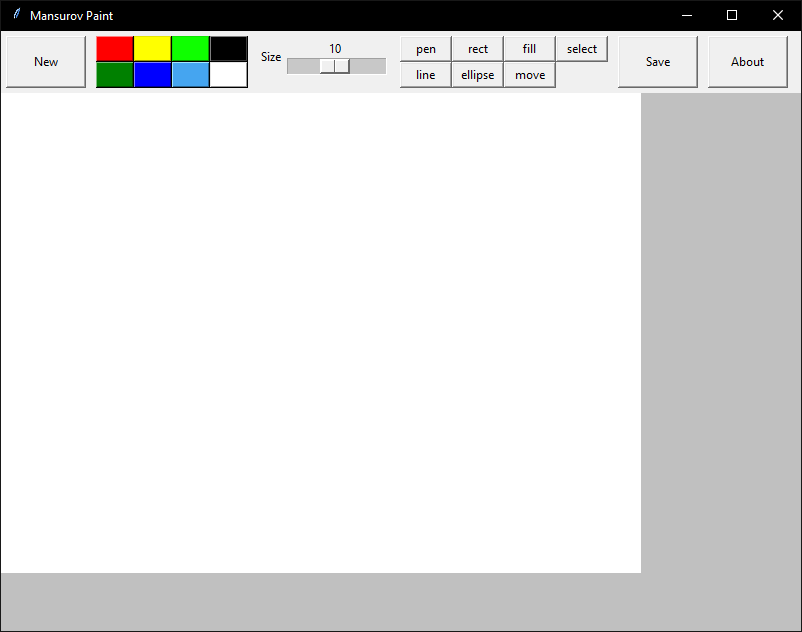


Рис. 1 – главное окно

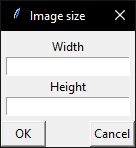


Рис. 2 – диалоговое окно задания размера

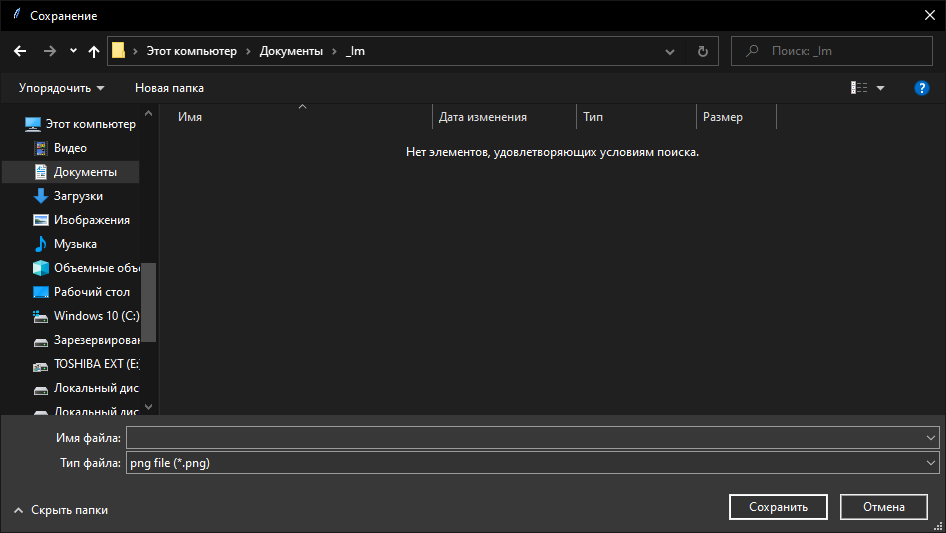


Рис. 3 – диалоговое окно выбора места сохранения

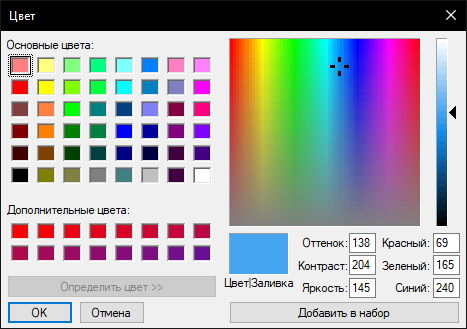


Рис. 4 – диалоговое окно выбора цвета

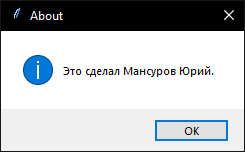


Рис. 5 – диалоговое окно информации

На Рис. 6 выделены следующие элементы интерфейса:

1. Кнопка «New» – позволяет создать новое чистое поле с заданными размерами
2. Область выбора цвета – нажатие левой клавишей мыши позволяет выбрать соответствующий цвет для рисования; нажатие правой клавишей мыши позволяет заменить цвет в ячейке на другой (через диалоговое окно выбора цвета) для последующего использования
3. Область выбора размера кисти – перемещение ползунка позволяет выбрать размер (диаметр) кисти от 1 до 20 пикселей
4. Область инструментов – позволяет выбрать один из следующих инструментов:
   1. pen – рисование кистью
   2. line – рисование прямой линии
   3. rect – рисование прямоугольника (при зажатом «Shift» - квадрата)
   4. ellipse – рисование эллипса (при зажатом «Shift» - круга)
   5. fill – заливка одноцветной области выбранным цветом
   6. move – перемещение изображения в окне просмотра (если изображение не влезает в окно просмотра)
   7. select – выделение области на изображении, а также перемещение и изменение размера выделенной области
5. Кнопка «Save» – сохранение изображения (выбор пути и название файла происходит через диалоговое окно сохранения файла)
6. Кнопка «About» – вывод информации о создателе приложения
7. Область рисования (изображение) – область к которой можно применять инструменты
8. Область просмотра – область в которой возможен просмотр и изменение изображения

Для удобства работы с изображением колесиком мыши можно изменять масштаб области рисования.

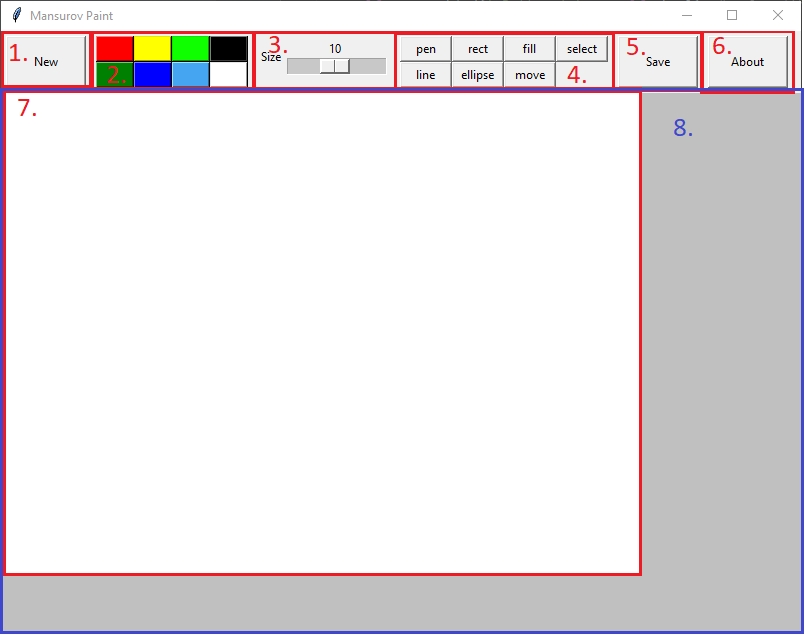


Рис. 6 – элементы интерфейса

## Реализация

Приложение было написано на языке Python 3 с использованием следующих модулей:

tkinter – модуль, содержащий классы для создания кроссплатформенного графического интерфейса

PIL – модуль, содержащий классы для работы с растровыми изображениями

Графический интерфейс реализован на основе модуля tkinter.

Инструменты рисования (pen, line, rect, ellipse, fill) реализованы с помощью алгоритмов пиксельного рисования. Инструмент перемещения изображения (move) взят из модуля tkinter. Инструмент выделения и изменения области (select) реализован с частичным использованием методов модуля PIL.

Для хранения и попиксельного изменения изображения используется класс Image модуля PIL.

### Кисти

В приложении используются кисти различных размеров. Все доступные кисти имеют форму круга, хотя возможна доработка приложения для создания кистей любых симметричных относительно вертикальной и горизонтальной осей форм за счет изменения кода построения кисти.

При изменении кисти старая кисть удаляется из памяти, а новая – создается. Это позволяет неограниченно расширять ассортимент кистей без влияния на используемую приложением память.

Для создания кисти применяется модифицированный алгоритм Брезенхема для построения окружности (Приложение 4). Для ускорения работы алгоритмов рисования используется дополнительная информация о кисти, содержащаяся в объекте кисти. Такой информацией являются пиксели кисти, которые изменяются в изображении при движении кисти вправо, вниз и по диагонали вправо вниз. В дальнейшем, при описании алгоритмов, эти пиксели будут называться границами кисти. В классе кисти форма хранится в свойстве form, правая граница – в свойстве R, нижняя граница – в свойстве B, диагональная граница – в свойстве RBAll.

Форма и границы кисти хранятся в виде массива кортежей (x,y), где каждый кортеж задает координаты точки, принадлежащей кисти или границе. Координаты задаются относительно центра кисти. В случае, если центр кисти находится между пикселями (диаметр кисти является четным числом пикселей), центром считается правый нижний пиксель среди ближайших к центру.

### Алгоритмы рисования

#### Pen

Данный инструмент подразумевает рисование линии вслед за движениями мыши. При быстром движении мышью вместо линии получается набор точек. Причиной этого является задержка между последовательными вызовами обработчика событий мыши. Данное поведение исправляется рисованием прямой линии между двумя последними позициями мыши вместо рисования одной точки в последней позиции мыши.

#### Line

Данный инструмент рисует линию между двумя точками. Для определения пикселей, находящихся на данной линии, используется алгоритм Брезенхема рисования линии (Приложение 1). Этот алгоритм позволяет построить прямую линию толщиной 1 пиксель (тонкую линию) между двумя точками (пикселями). Чтобы нарисовать линию более одного пикселя шириной, требуется нарисовать кисть в каждом пикселе тонкой линии. Для того, чтобы уменьшить количество бесполезной работы по перекрашиванию пикселей, в каждом пикселе линии кроме первого рисуется не вся кисть, а только соответствующая направлению рисования граница кисти (Приложение 2).

#### Rect

Данный инструмент рисует прямоугольник между двумя точками, определяющими противоположные углы. Форма прямоугольника не должна зависеть от формы кисти, поэтому используется алгоритм, зависящий от размера, но не от формы кисти (Приложение 3). Рисуется две горизонтальных и две вертикальных линии необходимой толщины, что не представляет трудности ввиду ориентированности линий.

#### Ellipse

Данный инструмент рисует эллипс, вписанный в прямоугольник, с заданными противоположными углами. Для рисования эллипса применяется модифицированный алгоритм Брезенхема для рисования окружностей (Приложение 5). Из-за необходимости рисовать эллипс заданной кистью применяется модификация алгоритма из приложения 5 (Приложение 6).

#### Fill

Данный алгоритм перекрашивает связную одноцветную область другим цветом. Для выполнения данной операции применяется рекурсивный алгоритм заливки (Приложение 7).

### Кроссплатформенность

Язык программирования Python 3 является кроссплатформенным языком программирования, поскольку имеет интерпретаторы, работающие на различных операционных системах. Модули, использованные при создании приложения, также являются кроссплатформенными.

## Инструкция по использованию приложения

Для работы приложения требуется интерпретатор языка Python 3 со следующими установленными библиотеками (модулями):

1. tkinter

В Windows 10 не требует отдельной установки

В Linux требуется установить специальный пакет. В Ubuntu и некоторых других Linux системах следует выполнить в командной строке команду

sudo apt-get install python3-tk

для установки этого пакета.

1. Pillow

Для установки требуется выполнить следующую команду в командной строке

pip install pillow

## Примеры работы

На следующих рисунках приведены скриншоты окна приложения после использования определенных инструментов.

На Рис. 7 показаны примеры использования различных инструментов рисования с различным размером кисти и цветом.Рис. 1 Инструменты, использованные для рисования фигур, слева направо: pen, line, rect, rect (с зажатым Shift), ellipse, ellipse (с зажатым Shift).

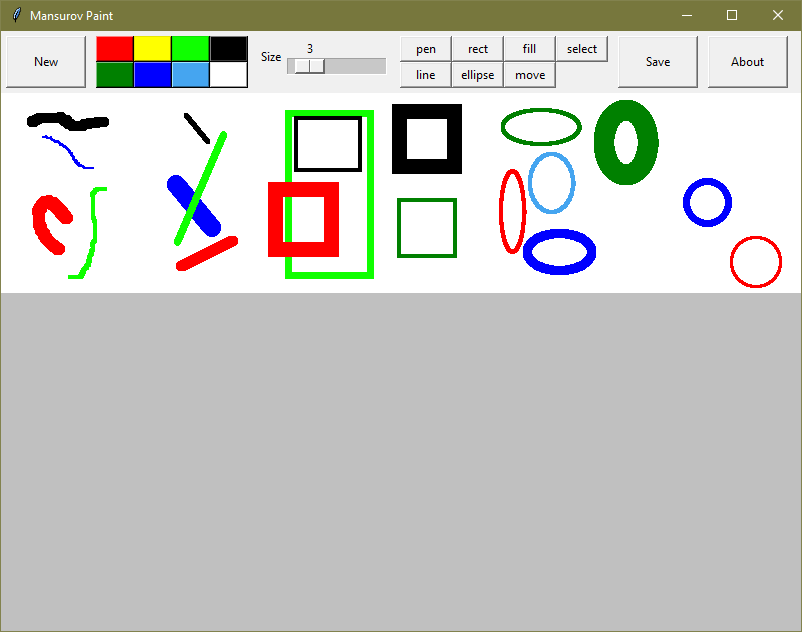


Рис. 7 – результат использования инструментов рисования

На Рис. 8 показан пример работы инструмента заливки (fill). На левом рисунке показано изображение до применения инструмента заливки. На правом – изображение полученное после заливки.

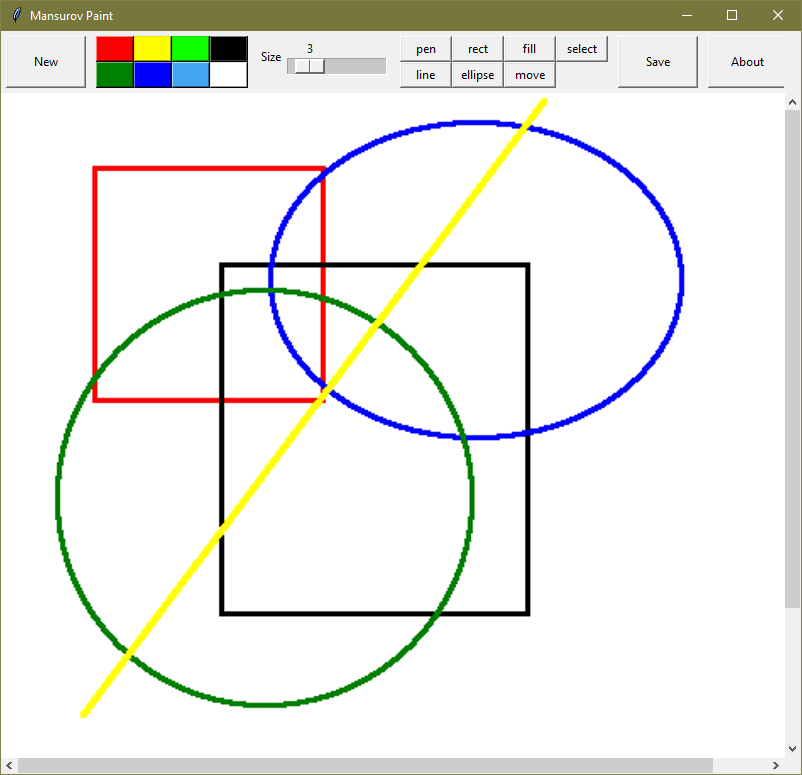
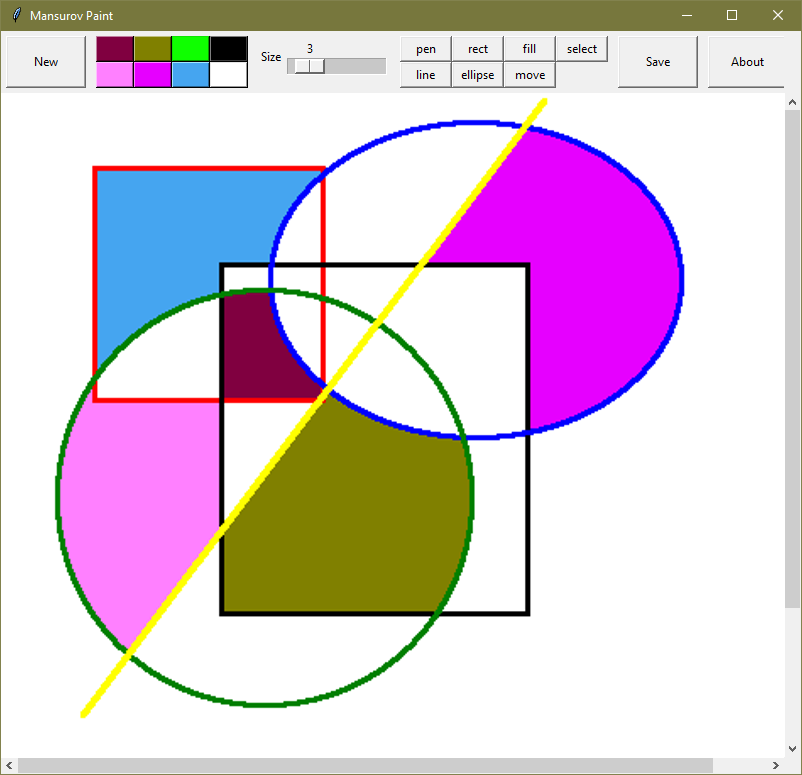
 

Рис. 8, 9 – пример работы инструмента fill

На следующих рисунках показан пример работы инструмента выделения select. На первом рисунке показан изначальное изображение. На следующих рисунках показано изображение после последовательного выполнения следующих действий: выделение области, перемещение выделенной области, изменение размера выделенной области, снятие выделения. Выделенная область обведена черно-белой рамкой.

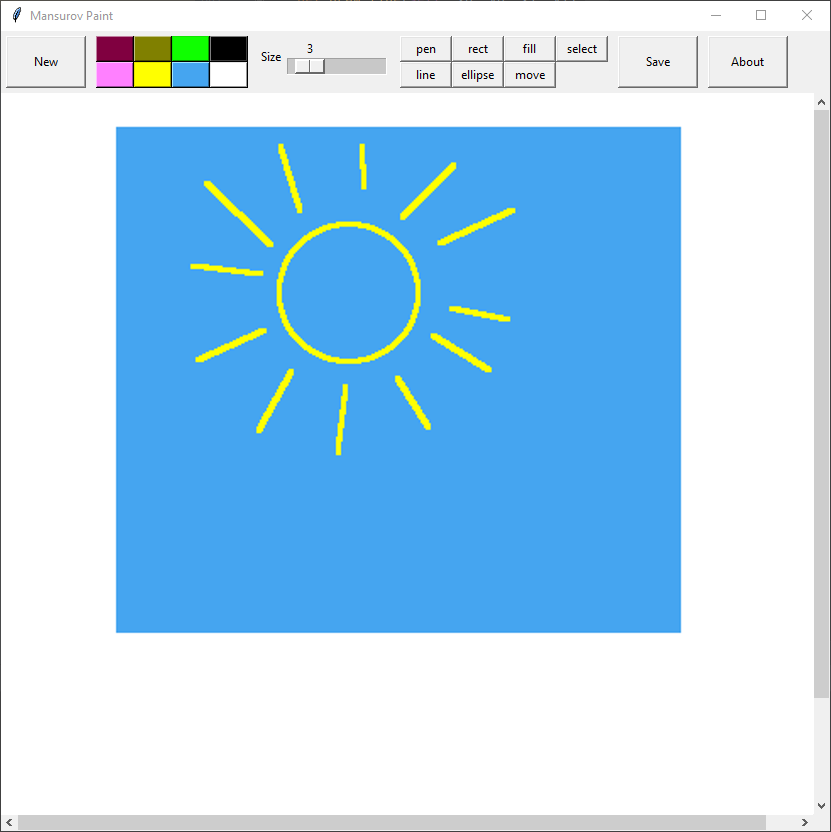
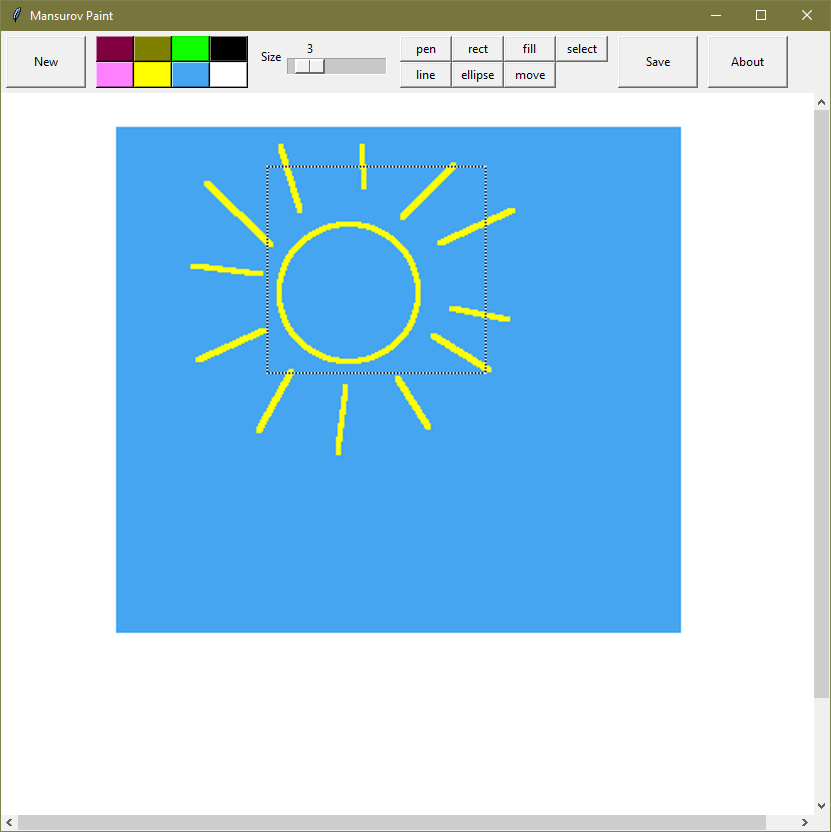
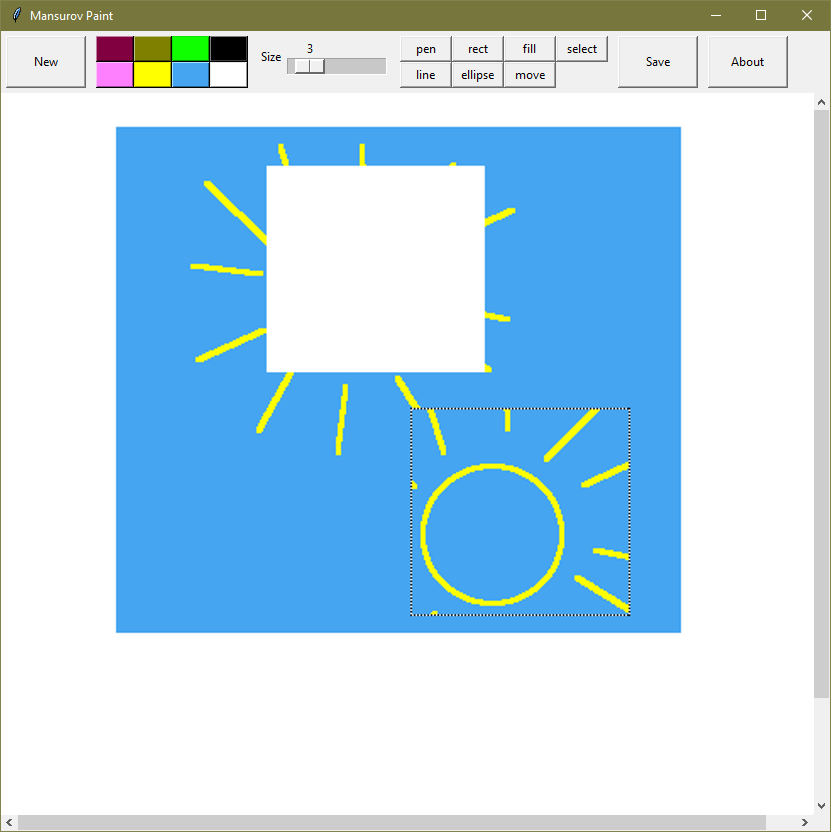
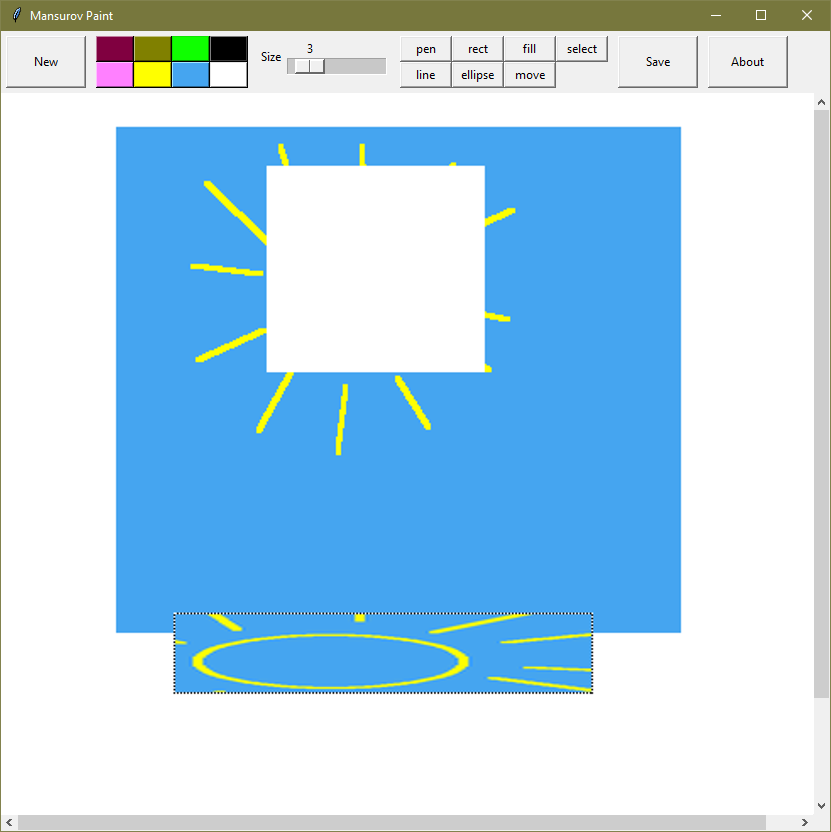
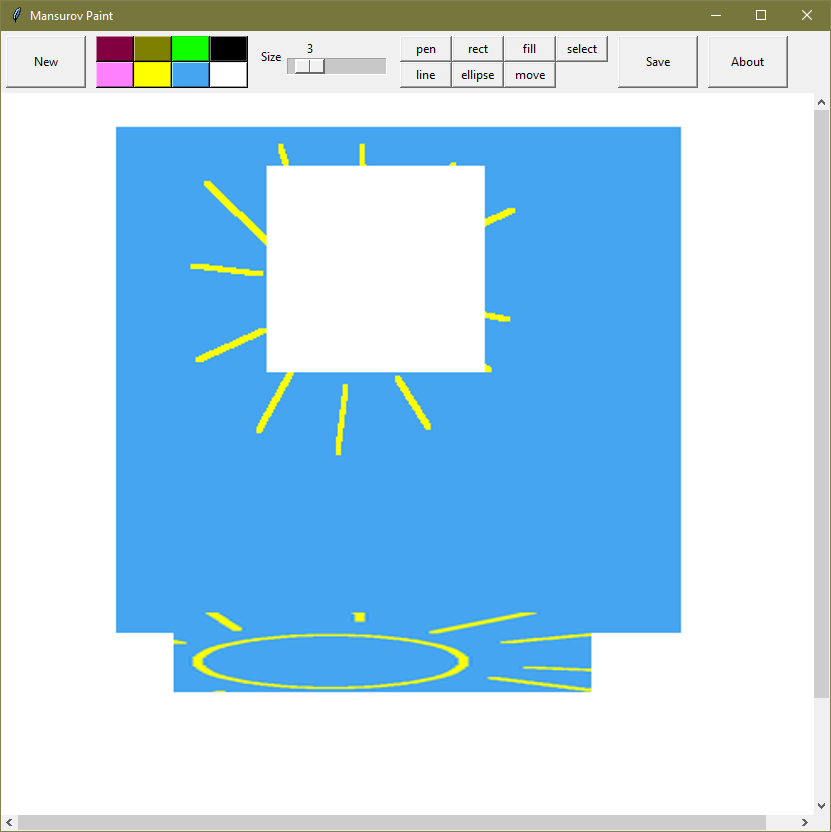
   
  

Рис. 10, 11, 12, 13, 14 – пример работы инструмента select

Все приведенные примеры показывают вид приложения в Windows 10. В других ОС приложение может выглядеть иначе, однако это не будет сказываться на функционале приложения.

# Исходный код

Исходный код приложения доступен в Github репозитории по ссылке:

* <https://github.com/kiarro/PythonGraphicEditor>

Исходный код программы:

**import** tkinter **as** tk  
**from** tkinter **import** filedialog, colorchooser  
**from** tkinter **import** simpledialog, messagebox  
**import** tkinter.ttk **as** ttk  
**from** PIL **import** Image, ImageTk, ImageColor  
  
*# simple dialog for input sizes of new image***class** DialogSize(tk.simpledialog.Dialog):  
 **def** \_\_init\_\_(self, parent):  
 self.w = **None** self.h=**None** super().\_\_init\_\_(parent, **"Image size"**)  
  
 **def** body(self, frame):  
 self.lbl = tk.Label(frame, text=**'Width'**)  
 self.lbl.pack()  
  
 self.width = tk.Entry(frame)  
 self.width.pack()  
  
 self.lbl = tk.Label(frame, text=**'Height'**)  
 self.lbl.pack()  
  
 self.height = tk.Entry(frame)  
 self.height.pack()  
 **return** frame  
  
 **def** buttonbox(self):  
 self.ok\_button = tk.Button(self, text=**'OK'**, width=5, command=self.ok\_pressed)  
 self.ok\_button.pack(side=**"left"**)  
 cancel\_button = tk.Button(self, text=**'Cancel'**, width=5, command=self.cancel\_pressed)  
 cancel\_button.pack(side=**"right"**)  
 self.bind(**"<Return>"**, **lambda** event: self.ok\_pressed())  
 self.bind(**"<Escape>"**, **lambda** event: self.cancel\_pressed())  
  
 **def** ok\_pressed(self):  
 *# print("ok")* self.w = int(self.width.get())  
 self.h = int(self.height.get())  
 self.destroy()  
  
 **def** cancel\_pressed(self):  
 *# print("cancel")* self.destroy()  
  
*# scrollbar that visibility depends on sizes of image and canvas***class** AutoScrollbar(ttk.Scrollbar):  
 *''' A scrollbar that hides itself if it's not needed.  
 Works only if you use the grid geometry manager '''* **def** set(self, lo, hi):  
 **if** float(lo) <= 0.0 **and** float(hi) >= 1.0:  
 self.grid\_remove()  
 **else**:  
 self.grid()  
 ttk.Scrollbar.set(self, lo, hi)  
  
 **def** pack(self, \*\*kw):  
 **raise** tk.TclError(**'Cannot use pack with this widget'**)  
  
 **def** place(self, \*\*kw):  
 **raise** tk.TclError(**'Cannot use place with this widget'**)  
  
*# class of main window***class** Paint(tk.Frame):  
 *# contains form of brush and borders of brush  
 # brush has form of round* **class** BrushForm:  
 **def** \_\_init\_\_(self, diam):  
 r, v = divmod(diam - 1, 2)  
 *# form of brush* form = list()  
 *# new dots when move to right* self.R = list()  
 *# new dots when move to bottom* self.B = list()  
 *# intersection of sets R and B* self.RB = list()  
 *# new dots when move to right-bottom* self.RBAll = list()  
 *# create round* x = 0  
 y = r  
 d = 1 - 2 \* r  
 *# last move -1 - horiz, 0 - diag, 1 - vert* l = 0  
 **while** y >= 0:  
 **if** d < 0 **and** 2 \* (d + y) - 1 < 0:  
 l = -1  
 x += 1  
 d += 2 \* x + 1  
 **continue  
 if** d > 0 **and** 2 \* (d - x) - 1 > 0:  
 self.R.append((x, y))  
 self.B.append((y, x))  
 self.R.append((x, -y - v))  
 self.B.append((-y - v, x))  
 self.RBAll.append((x, y))  
 self.RBAll.append((y, x))  
 self.RBAll.append((x, -y - v))  
 self.RBAll.append((-y - v, x))  
 *# if prev move - diag or horiz* **if** l <= 0: self.RB.append((x, y))  
 **for** xx **in** range(-x - v, +x + 1):  
 form.append((xx, +y))  
 form.append((xx, -y - v))  
 l = 1  
 y -= 1  
 d += -2 \* y + 1  
 **continue** self.R.append((x, y))  
 self.B.append((y, x))  
 self.R.append((x, -y - v))  
 self.B.append((-y - v, x))  
 self.RBAll.append((x, y))  
 self.RBAll.append((y, x))  
 self.RBAll.append((x, -y - v))  
 self.RBAll.append((-y - v, x))  
 self.RBAll.append((x, y-1))  
 self.RBAll.append((y-1, x))  
 self.RBAll.append((x, -y+1 - v))  
 self.RBAll.append((-y+1 - v, x))  
 **if** (l <= 0): self.RB.append((x, y))  
 **for** xx **in** range(-x - v, +x + 1):  
 form.append((xx, +y))  
 form.append((xx, -y - v))  
 l = 0  
 x += 1  
 y -= 1  
 d += 2 \* (x - y) + 2  
  
 self.form = form  
  
 **def** \_\_init\_\_(self, parent):  
 tk.Frame.\_\_init\_\_(self, parent)  
 self.parent = parent  
  
 *# start sizes of image* self.fig\_width = 640  
 self.fig\_height = 480  
  
 *# current image* self.image = Image.new(**'RGB'**, (self.fig\_width, self.fig\_height), color=(255, 255, 255))  
 *# copy of image (necessary for tools usage)* self.imgcopy = self.image.copy()  
 *# coordinates* self.x = -1  
 self.y = -1  
  
 self.selection = **None** self.selection\_move = **False** self.selection\_resize = **False** self.selection\_resize\_border = 5  
 self.selection\_image = **None** self.brush\_size = 10  
 self.brush\_color = (0,0,0)  
 self.brush = self.BrushForm(self.brush\_size)  
  
 *# functions that calling at ( every time for recalculate coordinates,  
 # on left mouse button down,  
 # on left mouse button move,  
 # on left mouse button up)* self.modes = {**"pen"**: (self.PointToImage, self.PrepareDraw, self.pen, self.null),  
 **"line"**: (self.PointToImage, self.PrepareDraw, self.line, self.null),  
 **"rect"**: (self.PointToImage, self.PrepareDraw, self.rect, self.null),  
 **"ellipse"**: (self.PointToImage, self.PrepareDraw, self.round, self.null),  
 **"fill"**: (self.PointToImage, self.fill, self.null, self.null),  
 **"move"**: ((**lambda** x,y: (x,y)), (**lambda** x,y: self.canvas.scan\_mark(x, y)), self.move, self.null),  
 **"select"**: (self.PointToImage, self.select\_start, self.select, self.select\_end),  
 }  
 self.mode = self.modes[**"pen"**]  
  
  
 self.initUI()  
 *# bind handling methods* self.canvas.bind(**'<Configure>'**, self.show\_image) *# canvas is resized* self.canvas.bind(**"<1>"**, self.LBdown) *# left mouse button down* self.canvas.bind(**"<B1-Motion>"**, self.LBmove) *# left mouse button move* self.canvas.bind(**"<ButtonRelease-1>"**, self.LBup) *# left mouse button up* self.canvas.bind(**'<MouseWheel>'**, self.wheel) *# wheel for Windows and MacOS, but not Linux* self.canvas.bind(**'<Button-5>'**, self.wheel) *# only with Linux, wheel scroll down* self.canvas.bind(**'<Button-4>'**, self.wheel) *# only with Linux, wheel scroll up  
  
  
 # h = 21  
 # r, \_ = divmod(h - 1, 2)  
 # xx0 = r + \_  
 # yy0 = r  
 # for i in range(1, 22):  
 # im = Image.new('RGB', (h, h), color=(255, 255, 255))  
 # # im.putpixel((0,0), (255))  
 # # im.putpixel((1,1), (255))  
 # f = self.BrushForm(i)  
 # for x, y in f.form:  
 # im.putpixel((xx0 + x, yy0 + y), (0, 0, 0))  
 # for x, y in f.R:  
 # im.putpixel((xx0 + x, yy0 + y), (255, 0, 0))  
 # for x, y in f.B:  
 # im.putpixel((xx0 + x, yy0 + y), (0, 255, 0))  
 # for x, y in f.RB:  
 # im.putpixel((xx0 + x, yy0 + y), (0, 0, 255))  
 # for x, y in f.RBAll:  
 # im.putpixel((xx0 + x, yy0 + y), (0, 255, 255))  
 #  
 # im.save(f'brush\_{i}.png')* print(**"done"**)  
  
 **def** initUI(self):  
  
 self.parent.title(**"Mansurov Paint"**)  
 self.pack(fill=tk.BOTH, expand=1)  
  
 self.columnconfigure(0, weight=1)  
 self.rowconfigure(1, weight=1)  
  
 self.top\_panel = tk.Frame(self)  
 self.top\_panel.grid(column=0, row=0, sticky=**'nswe'**)  
  
 vbar = AutoScrollbar(self, orient=**'vertical'**)  
 hbar = AutoScrollbar(self, orient=**'horizontal'**)  
 vbar.grid(row=1, column=1, sticky=**'ns'**)  
 hbar.grid(row=2, column=0, sticky=**'we'**)  
 self.canvas = tk.Canvas(self, highlightthickness=0,  
 *# width=self.fig\_width - 2, height=self.fig\_height - 2,* bg=**"#C0C0C0"**,  
 xscrollcommand=hbar.set, yscrollcommand=vbar.set) *# Создаем поле для рисования, устанавливаем белый фон* self.canvas.grid(column=0, row=1, sticky=**'nswe'**)  
 self.canvas.update()  
 *# self.canvas.create\_image((0, 0), image=ImageTk.PhotoImage(self.image), anchor='nw')* vbar.configure(command=self.scroll\_y) *# bind scrollbars to the canvas* hbar.configure(command=self.scroll\_x)  
  
 self.width, self.height = self.image.size  
 *# scale for the canvaas image* self.imscale = 1.0  
 *# zoom magnitude* self.delta = 1.3  
 self.container = self.canvas.create\_rectangle(0, 0, self.width, self.height, width=0)  
 self.show\_image()  
  
 btn = tk.Button(self.top\_panel, text=**"New"**, width=10, command=self.new\_image)  
 btn.grid(column=0, row=0, sticky=**'nswe'**, padx=5, pady=5)  
  
 color\_bar = tk.Frame(self.top\_panel)  
 color\_bar.grid(column=1, row=0, sticky=**'nswe'**, padx=5, pady=5)  
  
 colors\_list = [**"red"**, **"green"**, **"yellow"**, **"blue"**, **"#0FFF00"**, **"#45A5F0"**, **"black"**, **"white"**]  
 self.color\_buttons = list()  
 **for** i **in** range(len(colors\_list)):  
 btn = tk.Button(color\_bar, bg=colors\_list[i], width=4, command=**lambda** c=ImageColor.getcolor(colors\_list[i], **'RGB'**): self.set\_color(c))  
 btn.bind(**"<3>"**, **lambda** e, b=btn: self.ChangeColorTile(b))  
 d, m = divmod(i, 2)  
 btn.grid(column=d + 1, row=m)  
 self.color\_buttons.append(btn)  
  
 bar = tk.Frame(self.top\_panel)  
 lbl = tk.Label(bar, text=**"Size"**)  
 lbl.grid(column=0, row=0, sticky=**'nswe'**, padx=0, pady=0)  
 scale = tk.Scale(bar, from\_=1, to\_=20, variable=self.brush\_size, command=self.set\_size, length=100,  
 orient=tk.HORIZONTAL)  
 scale.set(self.brush\_size)  
 scale.grid(column=2, row=0, sticky=**'nswe'**, padx=0, pady=0)  
 bar.grid(column=2, row=0, sticky=**'nswe'**, padx=5, pady=5)  
  
 bar = tk.Frame(self.top\_panel)  
 i = 0  
 **for** m **in** self.modes.keys():  
 btn = tk.Button(bar, text=m, width=6, command=(**lambda** c=m: self.set\_mode(c)))  
 d, m = divmod(i, 2)  
 btn.grid(column=d + 1, row=m)  
 i += 1  
 bar.grid(column=3, row=0, sticky=**'nswe'**, padx=5, pady=5)  
  
 btn = tk.Button(self.top\_panel, text=**"Save"**, width=10, command=self.save)  
 btn.grid(column=4, row=0, sticky=**'nswe'**, padx=5, pady=5)  
  
 btn = tk.Button(self.top\_panel, text=**"About"**, width=10, command=self.about)  
 btn.grid(column=5, row=0, sticky=**'nswe'**, padx=5, pady=5)  
  
  
  
 **def** set\_mode(self, m):  
 **if** self.selection:  
 self.image = self.imgcopy.copy()  
 self.image.paste(self.selection\_image,  
 box=(self.selection[0], self.selection[1], self.selection[2], self.selection[3]))  
 self.imgcopy = self.image.copy()  
 self.selection = **None** self.show\_image()  
 self.mode = self.modes[m]  
 print(m)  
  
 **def** ChangeColorTile(self, btn):  
 col = ImageColor.getcolor(btn[**"bg"**], **'RGB'**)  
 color = colorchooser.askcolor(initialcolor=col, parent=self)  
 btn.configure(bg = color[1])  
 btn[**"command"**] = **lambda** c=color[0]: self.set\_color(c)  
  
 **def** scroll\_y(self, \*args, \*\*kwargs):  
 *''' Scroll canvas vertically and redraw the image '''* self.canvas.yview(\*args, \*\*kwargs) *# scroll vertically* self.show\_image() *# redraw the image* **def** scroll\_x(self, \*args, \*\*kwargs):  
 *''' Scroll canvas horizontally and redraw the image '''* self.canvas.xview(\*args, \*\*kwargs) *# scroll horizontally* self.show\_image() *# redraw the image* **def** wheel(self, event):  
 *''' Zoom with mouse wheel '''* x = self.canvas.canvasx(event.x)  
 y = self.canvas.canvasy(event.y)  
 bbox = self.canvas.bbox(self.container) *# get image area* **if** bbox[0] < x < bbox[2] **and** bbox[1] < y < bbox[3]: **pass** *# Ok! Inside the image* **else**: **return** *# zoom only inside image area* scale = 1.0  
 *# Respond to Linux (event.num) or Windows (event.delta) wheel event* **if** event.num == 5 **or** event.delta == -120: *# scroll down* i = min(self.width, self.height)  
 **if** int(i \* self.imscale) < 30: **return** *# image is less than 30 pixels* self.imscale /= self.delta  
 scale /= self.delta  
 **if** event.num == 4 **or** event.delta == 120: *# scroll up* i = min(self.canvas.winfo\_width(), self.canvas.winfo\_height())  
 **if** i < self.imscale: **return** *# 1 pixel is bigger than the visible area* self.imscale \*= self.delta  
 scale \*= self.delta  
 self.canvas.scale(**'all'**, x, y, scale, scale) *# rescale all canvas objects* self.show\_image()  
  
 **def** show\_image(self, event=**None**, img=**None**):  
 *''' Show image on the Canvas '''* **if** img == **None**:  
 img = self.image  
 bbox1 = self.canvas.bbox(self.container) *# get image area  
 # Remove 1 pixel shift at the sides of the bbox1* bbox1 = (bbox1[0] + 1, bbox1[1] + 1, bbox1[2] - 1, bbox1[3] - 1)  
 bbox2 = (self.canvas.canvasx(0), *# get visible area of the canvas* self.canvas.canvasy(0),  
 self.canvas.canvasx(self.canvas.winfo\_width()),  
 self.canvas.canvasy(self.canvas.winfo\_height()))  
 bbox = [min(bbox1[0], bbox2[0]), min(bbox1[1], bbox2[1]), *# get scroll region box* max(bbox1[2], bbox2[2]), max(bbox1[3], bbox2[3])]  
 **if** bbox[0] == bbox2[0] **and** bbox[2] == bbox2[2]: *# whole image in the visible area* bbox[0] = bbox1[0]  
 bbox[2] = bbox1[2]  
 **if** bbox[1] == bbox2[1] **and** bbox[3] == bbox2[3]: *# whole image in the visible area* bbox[1] = bbox1[1]  
 bbox[3] = bbox1[3]  
 self.canvas.configure(scrollregion=bbox) *# set scroll region* x1 = max(bbox2[0] - bbox1[0], 0) *# get coordinates (x1,y1,x2,y2) of the image tile* y1 = max(bbox2[1] - bbox1[1], 0)  
 x2 = min(bbox2[2], bbox1[2]) - bbox1[0]  
 y2 = min(bbox2[3], bbox1[3]) - bbox1[1]  
 **if** int(x2 - x1) > 0 **and** int(y2 - y1) > 0: *# show image if it in the visible area* x = min(int(x2 / self.imscale), self.width) *# sometimes it is larger on 1 pixel...* y = min(int(y2 / self.imscale), self.height) *# ...and sometimes not* image = img.crop((int(x1 / self.imscale), int(y1 / self.imscale), x, y))  
 imagetk = ImageTk.PhotoImage(image.resize((int(x2 - x1), int(y2 - y1))))  
 imageid = self.canvas.create\_image(max(bbox2[0], bbox1[0]), max(bbox2[1], bbox1[1]),  
 anchor=**'nw'**, image=imagetk)  
 self.canvas.lower(imageid) *# set image into background* self.canvas.imagetk = imagetk *# keep an extra reference to prevent garbage-collection* **def** putpixel(self, x, y):  
 **if** x >= 0 **and** y >= 0 **and** x < self.width **and** y < self.height:  
 *# self.image.put(self.brush\_color, (x, y))* self.image.putpixel((x, y), self.brush\_color)  
  
 **def** pen(self, cx, cy, \*args):  
 self.draw\_line(self.x, self.y, cx, cy)  
 self.x, self.y = cx, cy  
  
 **def** fill(self, cx, cy, \*args):  
 source = self.image.getpixel((cx, cy))  
 color = self.brush\_color  
 toFill = set()  
 toFill.add((cx,cy))  
 **while** toFill:  
 (x,y) = toFill.pop()  
 c = self.image.getpixel((x,y))  
 **if not** c == source:  
 **continue** self.image.putpixel((x,y), color)  
 **if** x-1>=0: toFill.add((x-1,y))  
 **if** x+1<self.width: toFill.add((x+1,y))  
 **if** y-1>=0: toFill.add((x,y-1))  
 **if** y+1<self.height: toFill.add((x,y+1))  
  
 **def** line(self, cx, cy, \*args):  
 self.image = self.imgcopy.copy()  
 self.show\_image()  
 self.draw\_line(self.x, self.y, cx, cy)  
  
 **def** rect(self, cx, cy, e):  
 self.image = self.imgcopy.copy()  
 self.show\_image()  
 **if** (e.state & 0x1): *# if shift pressed* h = min(abs(cx-self.x), abs(cy-self.y))  
 cx = self.x + (h **if** cx>self.x **else** -h)  
 cy = self.y + (h **if** cy>self.y **else** -h)  
 self.draw\_rectangle(self.x, self.y, cx, cy)  
  
 **def** round(self, cx, cy, e):  
 self.image = self.imgcopy.copy()  
 self.show\_image()  
 **if** (e.state & 0x1): *# if shift pressed* h = min(abs(cx-self.x), abs(cy-self.y))  
 cx = self.x + (h **if** cx>self.x **else** -h)  
 cy = self.y + (h **if** cy>self.y **else** -h)  
 self.draw\_ellipse(self.x, self.y, cx, cy)  
  
  
 **def** move(self, cx, cy, \*args):  
 self.canvas.scan\_dragto(cx, cy, gain=1)  
 self.show\_image() *# redraw the image* **def** select\_start(self, cx, cy, \*args):  
 *# if not selected yet => prepare for drawing* **if** self.selection == **None**:  
  
 self.x, self.y = cx, cy  
 self.imgcopy = self.image.copy()  
 **else**:  
 *# if pointer inside selection (with borders for resizing) => prepare for dragging* b = self.selection\_resize\_border  
 **if** cx>self.selection[0]+b **and** cx<self.selection[2]-b **and** cy>self.selection[1]+b **and** cy<self.selection[3]-b:  
 self.selection\_move = **True** *# save coords of selection relative to pointer* self.x, self.y = cx - self.selection[0], cy - self.selection[1]  
 **return** *# if pointer at border => prepare for resize  
 # if right border* **if** cx>self.selection[2]-b **and** cx<self.selection[2]+b **and** cy>self.selection[1]+b **and** cy<self.selection[3]-b:  
 self.selection\_resize = **"r"** *# save pointer coords* self.x, self.y = cx, cy  
 **return** *# if left border* **if** cx>self.selection[0]-b **and** cx<self.selection[0]+b **and** cy>self.selection[1]+b **and** cy<self.selection[3]-b:  
 self.selection\_resize = **"l"** *# save pointer coords* self.x, self.y = cx, cy  
 **return** *# if top border* **if** cx>self.selection[0]+b **and** cx<self.selection[2]-b **and** cy>self.selection[1]-b **and** cy<self.selection[1]+b:  
 self.selection\_resize = **"t"** *# save pointer coords* self.x, self.y = cx, cy  
 **return** *# if bottom border* **if** cx>self.selection[0]+b **and** cx<self.selection[2]-b **and** cy>self.selection[3]-b **and** cy<self.selection[3]+b:  
 self.selection\_resize = **"b"** *# save pointer coords* self.x, self.y = cx, cy  
 **return** *# if right top corner* **if** cx>self.selection[2]-b **and** cx<self.selection[2]+b **and** cy>self.selection[1]-b **and** cy<self.selection[1]+b:  
 self.selection\_resize = **"rt"** *# save pointer coords* self.x, self.y = cx, cy  
 **return** *# if right bottom corner* **if** cx>self.selection[2]-b **and** cx<self.selection[2]+b **and** cy>self.selection[3]-b **and** cy<self.selection[3]+b:  
 self.selection\_resize = **"rb"** *# save pointer coords* self.x, self.y = cx, cy  
 **return** *# if left top corner* **if** cx>self.selection[0]-b **and** cx<self.selection[0]+b **and** cy>self.selection[1]-b **and** cy<self.selection[1]+b:  
 self.selection\_resize = **"lt"** *# save pointer coords* self.x, self.y = cx, cy  
 **return** *# if left bottom corner* **if** cx>self.selection[0]-b **and** cx<self.selection[0]+b **and** cy>self.selection[3]-b **and** cy<self.selection[3]+b:  
 self.selection\_resize = **"lb"** *# save pointer coords* self.x, self.y = cx, cy  
 **return** *# else reset selection* self.image = self.imgcopy.copy()  
 self.image.paste(self.selection\_image,  
 box=(self.selection[0], self.selection[1], self.selection[2], self.selection[3]))  
 self.imgcopy = self.image.copy()  
 self.selection = **None** self.x, self.y = cx, cy  
 self.image = self.imgcopy.copy()  
  
 **def** select(self, cx, cy, \*args):  
 **if** self.selection == **None**:  
 **if** self.x<0 **or** self.y<0 **or** self.x>=self.width **or** self.y>=self.height:  
 **return** self.image = self.imgcopy.copy()  
 self.show\_image()  
 **if** cx<0:  
 cx=0  
 **if** cy<0:  
 cy=0  
 **if** cx>=self.width:  
 cx=self.width-1  
 **if** cy>=self.height:  
 cy=self.height-1  
 self.draw\_select(self.x, self.y, cx, cy)  
 **return  
 else**:  
 **if** self.selection\_move:  
 self.image = self.imgcopy.copy()  
 self.image.paste(self.selection\_image,  
 box=(cx-self.x, cy-self.y, cx-self.x+self.selection[2]-self.selection[0], cy-self.y+self.selection[3]-self.selection[1]))  
 **return  
 if** self.selection\_resize:  
 self.image = self.imgcopy.copy()  
 img = self.selection\_image.copy()  
 w, h = img.size  
 **if** self.selection\_resize == **"r"**:  
 img = img.resize((w+cx-self.x,h), Image.ANTIALIAS)  
 self.image.paste(img,  
 box=(self.selection[0], self.selection[1], self.selection[0]+w+cx-self.x, self.selection[1]+h))  
 **if** self.selection\_resize == **"l"**:  
 img = img.resize((w-cx+self.x,h), Image.ANTIALIAS)  
 self.image.paste(img,  
 box=(self.selection[2]-w+cx-self.x, self.selection[1], self.selection[2], self.selection[1]+h))  
 **if** self.selection\_resize == **"t"**:  
 img = img.resize((w,h-cy+self.y), Image.ANTIALIAS)  
 self.image.paste(img,  
 box=(self.selection[0], self.selection[3]-h+cy-self.y, self.selection[0]+w, self.selection[3]))  
 **if** self.selection\_resize == **"b"**:  
 img = img.resize((w,h+cy-self.y), Image.ANTIALIAS)  
 self.image.paste(img,  
 box=(self.selection[0], self.selection[1], self.selection[0]+w, self.selection[3]+cy-self.y))  
 **if** self.selection\_resize == **"rt"**:  
 img = img.resize((w+cx-self.x,h-cy+self.y), Image.ANTIALIAS)  
 self.image.paste(img,  
 box=(self.selection[0], self.selection[3]-h+cy-self.y, self.selection[0]+w+cx-self.x, self.selection[3]))  
 **if** self.selection\_resize == **"rb"**:  
 img = img.resize((w+cx-self.x,h+cy-self.y), Image.ANTIALIAS)  
 self.image.paste(img,  
 box=(self.selection[0], self.selection[1], self.selection[0]+w+cx-self.x, self.selection[3]+cy-self.y))  
 **if** self.selection\_resize == **"lt"**:  
 img = img.resize((w-cx+self.x,h-cy+self.y), Image.ANTIALIAS)  
 self.image.paste(img,  
 box=(self.selection[2]-w+cx-self.x, self.selection[3]-h+cy-self.y, self.selection[2], self.selection[3]))  
 **if** self.selection\_resize == **"lb"**:  
 img = img.resize((w-cx+self.x,h+cy-self.y), Image.ANTIALIAS)  
 self.image.paste(img,  
 box=(self.selection[2]-w+cx-self.x, self.selection[1], self.selection[2], self.selection[3]+cy-self.y))  
  
 **def** select\_end(self, cx, cy, \*args):  
 **if** (self.selection\_move):  
 *# self.imgcopy = self.image.copy()* self.selection = (cx-self.x, cy-self.y, cx-self.x+self.selection[2]-self.selection[0], cy-self.y+self.selection[3]-self.selection[1])  
 self.draw\_select(self.selection[0], self.selection[1], self.selection[2], self.selection[3])  
 self.selection\_move = **False  
 return** *# self.selection = None* **if** (self.selection\_resize):  
 w, h = self.selection[2]-self.selection[0], self.selection[3]-self.selection[1]  
 **if** self.selection\_resize == **"r"**:  
 self.selection = (self.selection[0], self.selection[1], self.selection[0]+w+cx-self.x, self.selection[1]+h)  
 self.selection\_image = self.selection\_image.resize((w+cx-self.x,h), Image.ANTIALIAS)  
 **if** self.selection\_resize == **"l"**:  
 self.selection = (self.selection[2]-w+cx-self.x, self.selection[1], self.selection[2], self.selection[1]+h)  
 self.selection\_image = self.selection\_image.resize((w-cx+self.x,h), Image.ANTIALIAS)  
 **if** self.selection\_resize == **"t"**:  
 self.selection = (self.selection[0], self.selection[3]-h+cy-self.y, self.selection[0]+w, self.selection[3])  
 self.selection\_image = self.selection\_image.resize((w,h-cy+self.y), Image.ANTIALIAS)  
 **if** self.selection\_resize == **"b"**:  
 self.selection = (self.selection[0], self.selection[1], self.selection[0]+w, self.selection[3]+cy-self.y)  
 self.selection\_image = self.selection\_image.resize((w,h+cy-self.y), Image.ANTIALIAS)  
 **if** self.selection\_resize == **"rt"**:  
 self.selection = (self.selection[0], self.selection[3]-h+cy-self.y, self.selection[0]+w+cx-self.x, self.selection[3])  
 self.selection\_image = self.selection\_image.resize((w+cx-self.x,h-cy+self.y), Image.ANTIALIAS)  
 **if** self.selection\_resize == **"rb"**:  
 self.selection = (self.selection[0], self.selection[1], self.selection[0]+w+cx-self.x, self.selection[3]+cy-self.y)  
 self.selection\_image = self.selection\_image.resize((w+cx-self.x,h+cy-self.y), Image.ANTIALIAS)  
 **if** self.selection\_resize == **"lt"**:  
 self.selection = (self.selection[2]-w+cx-self.x, self.selection[3]-h+cy-self.y, self.selection[2], self.selection[3])  
 self.selection\_image = self.selection\_image.resize((w-cx+self.x,h-cy+self.y), Image.ANTIALIAS)  
 **if** self.selection\_resize == **"lb"**:  
 self.selection = (self.selection[2]-w+cx-self.x, self.selection[1], self.selection[2], self.selection[3]+cy-self.y)  
 self.selection\_image = self.selection\_image.resize((w-cx+self.x,h+cy-self.y), Image.ANTIALIAS)  
  
 self.draw\_select(self.selection[0], self.selection[1], self.selection[2], self.selection[3])  
 self.selection\_resize = **False  
 else**:  
 **if** self.x<0 **or** self.y<0 **or** self.x>=self.width **or** self.y>=self.height:  
 **return  
 if** cx<0:  
 cx=0  
 **if** cy<0:  
 cy=0  
 **if** cx>=self.width:  
 cx=self.width-1  
 **if** cy>=self.height:  
 cy=self.height-1  
 x0, x1 = ((self.x, cx) **if** cx>self.x **else** (cx, self.x))  
 y0, y1 = ((self.y, cy) **if** cy>self.y **else** (cy, self.y))  
 self.selection = (x0, y0, x1, y1)  
 w = self.selection[2]-self.selection[0]  
 h = self.selection[3]-self.selection[1]  
 self.selection\_image = self.imgcopy.crop(self.selection)  
 img = Image.new(**'RGB'**, (w, h), color=(255,255,255))  
 self.imgcopy.paste(img, (self.selection[0],self.selection[1], self.selection[2], self.selection[3]))  
  
 **def** set\_color(self, new\_color):  
 print(new\_color)  
 self.brush\_color = new\_color  
  
 **def** PointToImage(self, xd, yd):  
 bbox1 = self.canvas.bbox(self.container) *# get image area  
 # Remove 1 pixel shift at the sides of the bbox1* bbox1 = (bbox1[0] + 1, bbox1[1] + 1, bbox1[2] - 1, bbox1[3] - 1)  
 bbox2 = (self.canvas.canvasx(0), *# get visible area of the canvas* self.canvas.canvasy(0),  
 self.canvas.canvasx(self.canvas.winfo\_width()),  
 self.canvas.canvasy(self.canvas.winfo\_height()))  
 bbox = (bbox1[0]-bbox2[0],bbox1[1]-bbox2[1],bbox1[2]-bbox2[2],bbox1[3]-bbox2[3])  
 x, y = int((xd-bbox[0])/self.imscale), int((yd-bbox[1])/self.imscale)  
 **return** (x, y)  
  
 **def** PrepareDraw(self, x, y):  
 self.x, self.y = x, y  
 self.imgcopy = self.image.copy()  
  
 **def** LBdown(self, e):  
 x, y = e.x, e.y  
 x, y = self.mode[0](x,y)  
 self.mode[1](x,y)  
 self.LBmove(e)  
  
 **def** LBmove(self, e):  
 x, y = e.x, e.y  
 cx, cy = self.mode[0](x,y)  
 self.mode[2](cx, cy, e)  
 self.show\_image()  
  
 **def** LBup(self, e):  
 cx, cy = self.mode[0](e.x, e.y)  
 self.mode[3](cx, cy)  
 self.show\_image()  
  
 **def** draw\_line(self, x0, y0, x1, y1):  
 dx = abs(x1 - x0)  
 dy = abs(y1 - y0)  
 dirx = 1 **if** x1 - x0 > 0 **else** -1  
 diry = 1 **if** y1 - y0 > 0 **else** -1  
 brush = self.brush  
 e = 0  
  
 *# draw start point* **for** x, y **in** brush.form:  
 self.putpixel(x0 + x \* dirx, y0 + y \* diry)  
  
 **if** dx > dy:  
 de = dy + 1  
 y = y0  
 **for** x **in** range(x0, x1 + dirx, dirx):  
 **for** xx, yy **in** brush.R:  
 self.putpixel(x + xx \* dirx, y + yy \* diry)  
 e += de  
 **if** e > dx + 1:  
 y += diry  
 **for** xx, yy **in** brush.B:  
 self.putpixel(x + xx \* dirx, y + yy \* diry)  
 e -= (dx + 1)  
  
 **else**:  
 de = dx + 1  
 x = x0  
 **for** y **in** range(y0, y1 + diry, diry):  
 **for** xx, yy **in** brush.B:  
 self.putpixel(x + xx \* dirx, y + yy \* diry)  
 e += de  
 **if** e > dy + 1:  
 x += dirx  
 **for** xx, yy **in** brush.R:  
 self.putpixel(x + xx \* dirx, y + yy \* diry)  
 e -= (dy + 1)  
  
 **def** draw\_select(self, x0, y0, x1, y1):  
 color = ((0,0,0), (255,255,255))  
 **if** x0>x1: x0, x1 = x1, x0  
 **if** y0>y1: y0, y1 = y1, y0  
 **for** x **in** range(x0, x1+1):  
 self.image.putpixel((x, y0), color[x%2])  
 self.image.putpixel((x, y1), color[x%2])  
 **for** y **in** range(y0, y1+1):  
 self.image.putpixel((x0, y), color[y%2])  
 self.image.putpixel((x1, y), color[y%2])  
  
 **def** draw\_rectangle(self, x0, y0, x1, y1):  
 h = self.brush\_size  
 h1, h2 = divmod(h, 2)  
 h1, h2 = h1, h1+h2  
 **if** x0>x1: x0, x1 = x1, x0  
 **if** y0>y1: y0, y1 = y1, y0  
 **for** x **in** range(x0-h1, x1+h2):  
 **for** yy **in** range(-h1, h2):  
 self.putpixel(x, y0+yy)  
 self.putpixel(x, y1+yy)  
 **for** y **in** range(y0+h1, y1-h2+1):  
 **for** xx **in** range(-h1, h2):  
 self.putpixel(x0+xx, y)  
 self.putpixel(x1+xx, y)  
  
 **def** draw\_ellipse(self, x0, y0, x1, y1):  
 diamx = abs(x1 - x0)  
 diamy = abs(y1 - y0)  
 *# rx* a, \_a = divmod(diamx - 1, 2)  
 *# ry* b, \_b = divmod(diamy - 1, 2)  
  
 xc = x0 + (a + \_a **if** x1 > x0 **else** -a)  
 yc = y0 + (b + \_b **if** y1 > y0 **else** -b)  
 xcm = xc - \_a  
 ycm = yc - \_b  
 a2 = a \* a  
 b2 = b \* b  
 x = 0  
 y = b  
 d = b2 - 2 \* b \* a2 + a2  
 e = 0  
  
 *# draw start point* **for** xx, yy **in** self.brush.form:  
 self.putpixel(xc + xx, yc+b - yy)  
 self.putpixel(xc + xx, ycm-b + yy)  
 self.putpixel(xc+a - xx, yc - yy)  
 self.putpixel(xcm-a - xx, yc + yy)  
 self.putpixel(xc+a - xx, ycm - yy)  
 self.putpixel(xcm-a - xx, ycm + yy)  
  
 **while** y >= 0:  
 **if** y == 0:  
 **for** xf **in** range(x, a + 1):  
 **for** xx, yy **in** self.brush.R:  
 self.putpixel(xc+xf + xx, yc + yy)  
 self.putpixel(xcm-xf - xx, yc + yy)  
 y -= 1  
 **else**:  
 **for** xx, yy **in** self.brush.RBAll:  
 self.putpixel(xc+x + xx, yc+y - yy)  
 self.putpixel(xcm-x - xx, yc+y - yy)  
 self.putpixel(xc+x + xx, ycm-y + yy)  
 self.putpixel(xcm-x - xx, ycm-y + yy)  
 *# horizontal or diag* **if** d < 0:  
 e = 2 \* d + 2 \* y \* a2 - a2  
 *# horizontal* **if** e <= 0:  
 x += 1  
 d += 2 \* x \* b2 + b2  
 *# diag* **else**:  
 x += 1  
 y -= 1  
 d += (2 \* x \* b2 + b2 - 2 \* y \* a2 + a2)  
 *# vertical or diag* **else**:  
 e = 2 \* d - 2 \* x \* b2 - b2  
 *# vertical* **if** e > 0:  
 y -= 1  
 d += (-2 \* y \* a2 + a2)  
 *# diag* **else**:  
 x += 1  
 y -= 1  
 d += (2 \* x \* b2 + b2 - 2 \* y \* a2 + a2)  
  
 **def** set\_size(self, val):  
 self.brush\_size = int(val)  
 self.brush = self.BrushForm(int(val))  
  
 **def** new\_image(self):  
 d = DialogSize(self)  
 **if** (d.h == **None**):  
 **return** w, h = d.w, d.h  
  
 self.canvas.scale(**'all'**, 0, 0, 1/self.imscale, 1/self.imscale) *# rescale all canvas objects* self.image = Image.new(**'RGB'**, (w, h), color=(255, 255, 255))  
 self.imgcopy = self.image.copy()  
 self.width, self.height = self.image.size  
 *# scale for the canvaas image* self.imscale = 1.0  
 *# zoom magnitude* self.delta = 1.3  
 self.container = self.canvas.create\_rectangle(0, 0, self.width, self.height, width=0)  
 self.show\_image()  
  
  
 **def** save(self):  
 path = filedialog.asksaveasfilename(defaultextension=**".png"**, filetypes=((**"png file"**, **"\*.png"**),(**"All Files"**, **"\*.\*"**) ))  
 **if** path != **""**:  
 *# self.image.write(path)* self.image.save(path)  
  
 **def** about(self):  
 messagebox.showinfo(**'About'**, **'Это сделал Мансуров Юрий.'**, parent=self)  
  
 **def** save\_img(self):  
 self.imgcopy = self.image.copy()  
  
 **def** null(self, \*args):  
 **return  
  
  
  
def** main():  
 root = tk.Tk()  
 root.geometry(**"800x600+300+300"**)  
 app = Paint(root)  
 root.mainloop()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

# Заключение

В ходе работы было разработано приложение, позволяющее создавать и изменять растровое изображение с помощью инструментов. Для создания приложения был использован пакет кроссплатформенной графики для языка Python 3. Также были исследованы, доработаны и реализованы алгоритмы рисования примитивов.

# Приложение 1. Алгоритм Брезенхема для линии

На вход алгоритму подаются две точки (x0, y0) и (x1, y1), между которыми строится линия толщиной 1 пиксель.

dx = abs(x1 - x0)  
dy = abs(y1 - y0)  
dirx = 1 **if** x1 - x0 > 0 **else** -1  
diry = 1 **if** y1 - y0 > 0 **else** -1  
brush = self.brush  
e = 0  
  
**if** dx > dy:  
 de = dy + 1  
 y = y0  
 **for** x **in** range(x0, x1 + dirx, dirx):  
 self.putpixel(x, y)  
 e += de  
 **if** e > dx + 1:  
 y += diry  
 e -= (dx + 1)

**else**:  
 de = dx + 1  
 x = x0  
 **for** y **in** range(y0, y1 + diry, diry):  
 self.putpixel(x, y)  
 e += de  
 **if** e > dy + 1:  
 x += dirx  
 e -= (dy + 1)

# Приложение 2. Модифицированный алгоритм Брезенхема для толстой линии

На вход алгоритму подаются две точки (x0, y0) и (x1, y1), между которыми строится линия. Толщина линии, а также форма на ее концах определяются кистью brush.

dx = abs(x1 - x0)  
dy = abs(y1 - y0)  
dirx = 1 **if** x1 - x0 > 0 **else** -1  
diry = 1 **if** y1 - y0 > 0 **else** -1  
e = 0  
  
*# draw start point***for** x, y **in** brush.form:  
 self.putpixel(x0 + x \* dirx, y0 + y \* diry)  
  
**if** dx > dy:  
 de = dy + 1  
 y = y0  
 **for** x **in** range(x0, x1 + dirx, dirx):  
 **for** xx, yy **in** brush.R:  
 self.putpixel(x + xx \* dirx, y + yy \* diry)  
 e += de  
 **if** e > dx + 1:  
 y += diry  
 **for** xx, yy **in** brush.B:  
 self.putpixel(x + xx \* dirx, y + yy \* diry)  
 e -= (dx + 1)  
  
**else**:  
 de = dx + 1  
 x = x0  
 **for** y **in** range(y0, y1 + diry, diry):  
 **for** xx, yy **in** brush.B:  
 self.putpixel(x + xx \* dirx, y + yy \* diry)  
 e += de  
 **if** e > dy + 1:  
 x += dirx  
 **for** xx, yy **in** brush.R:  
 self.putpixel(x + xx \* dirx, y + yy \* diry)  
 e -= (dy + 1)

# Приложение 3. Алгоритм построения прямоугольника

На вход алгоритму подаются две точки (x0, y0) и (x1, y1), являющиеся противоположными углами прямоугольника. Толщина линии задается параметром h.

h1, h2 = divmod(h, 2)  
h1, h2 = h1, h1+h2  
**if** x0>x1: x0, x1 = x1, x0  
**if** y0>y1: y0, y1 = y1, y0  
**for** x **in** range(x0-h1, x1+h2):  
 **for** yy **in** range(-h1, h2):  
 self.putpixel(x, y0+yy)  
 self.putpixel(x, y1+yy)  
**for** y **in** range(y0+h1, y1-h2+1):  
 **for** xx **in** range(-h1, h2):  
 self.putpixel(x0+xx, y)  
 self.putpixel(x1+xx, y)

# Приложение 4. Модифицированный алгоритм Брезенхема для круга

На вход алгоритму подается диаметр круга в пикселях. Точки, принадлежащие кругу, сохраняются в списке form в виде координат (x, y), относительно центра круга (ближайшего правого нижнего пикселя к центру). Алгоритм может строить круг с диаметром равным четному числу пикселей.

form = list()

r, v = divmod(diam - 1, 2)  
x = 0  
y = r  
d = 1 - 2 \* r  
**while** y >= 0:  
 **if** d < 0 **and** 2 \* (d + y) - 1 < 0:  
 x += 1  
 d += 2 \* x + 1  
 **continue  
 if** d > 0 **and** 2 \* (d - x) - 1 > 0:  
 **for** xx **in** range(-x - v, +x + 1):  
 form.append((xx, +y))  
 form.append((xx, -y - v))  
 y -= 1  
 d += -2 \* y + 1  
 **continue  
  
 for** xx **in** range(-x - v, +x + 1):  
 form.append((xx, +y))  
 form.append((xx, -y - v))  
 x += 1  
 y -= 1  
 d += 2 \* (x - y) + 2

# Приложение 5. Модифицированный алгоритм Брезенхема для эллипса

На вход алгоритму подаются две точки (x0, y0) и (x1, y1), являющиеся противоположными углами прямоугольника, в который требуется вписать эллипс. Строится эллипс толщиной в 1 пиксель. Алгоритм может строить эллипс с осями равными четному числу пикселей.

diamx = abs(x1 - x0)  
diamy = abs(y1 - y0)  
*# rx*a, \_a = divmod(diamx - 1, 2)  
*# ry*b, \_b = divmod(diamy - 1, 2)  
  
xc = x0 + (a + \_a **if** x1 > x0 **else** -a)  
yc = y0 + (b + \_b **if** y1 > y0 **else** -b)  
xcm = xc - \_a  
ycm = yc - \_b  
a2 = a \* a  
b2 = b \* b  
x = 0  
y = b  
d = b2 - 2 \* b \* a2 + a2  
e = 0  
  
**while** y >= 0:  
 **if** y == 0:  
 **for** xf **in** range(x, a + 1):  
 self.putpixel(xc+xf, yc)  
 self.putpixel(xcm-xf, yc)  
 y -= 1  
 **else**:  
 self.putpixel(xc+x , yc+y)  
 self.putpixel(xcm-x, yc+y)  
 self.putpixel(xc+x , ycm-y)  
 self.putpixel(xcm-x, ycm-y)  
 *# horizontal or diag* **if** d < 0:  
 e = 2 \* d + 2 \* y \* a2 - a2  
 *# horizontal* **if** e <= 0:  
 x += 1  
 d += 2 \* x \* b2 + b2  
 *# diag* **else**:  
 x += 1  
 y -= 1  
 d += (2 \* x \* b2 + b2 - 2 \* y \* a2 + a2)  
 *# vertical or diag* **else**:  
 e = 2 \* d - 2 \* x \* b2 - b2  
 *# vertical* **if** e > 0:  
 y -= 1  
 d += (-2 \* y \* a2 + a2)  
 *# diag* **else**:  
 x += 1  
 y -= 1  
 d += (2 \* x \* b2 + b2 - 2 \* y \* a2 + a2)

# Приложение 6. Модифицированный алгоритм Брезенхема для эллипса на основе кисти

На вход алгоритму подаются две точки (x0, y0) и (x1, y1), являющиеся противоположными углами прямоугольника, в который требуется вписать эллипс. Строится эллипс, толщина которого зависит от размера и формы кисти. Алгоритм может строить эллипс с осями равными четному числу пикселей. Кисть задается параметром brush.

diamx = abs(x1 - x0)  
diamy = abs(y1 - y0)  
*# rx*a, \_a = divmod(diamx - 1, 2)  
*# ry*b, \_b = divmod(diamy - 1, 2)  
  
xc = x0 + (a + \_a **if** x1 > x0 **else** -a)  
yc = y0 + (b + \_b **if** y1 > y0 **else** -b)  
xcm = xc - \_a  
ycm = yc - \_b  
a2 = a \* a  
b2 = b \* b  
x = 0  
y = b  
d = b2 - 2 \* b \* a2 + a2  
e = 0  
  
*# draw start point***for** xx, yy **in** brush.form:  
 self.putpixel(xc + xx, yc+b - yy)  
 self.putpixel(xc + xx, ycm-b + yy)  
 self.putpixel(xc+a - xx, yc - yy)  
 self.putpixel(xcm-a - xx, yc + yy)  
 self.putpixel(xc+a - xx, ycm - yy)  
 self.putpixel(xcm-a - xx, ycm + yy)  
  
**while** y >= 0:  
 **if** y == 0:  
 **for** xf **in** range(x, a + 1):  
 **for** xx, yy **in** brush.R:  
 self.putpixel(xc+xf + xx, yc + yy)  
 self.putpixel(xcm-xf - xx, yc + yy)  
 y -= 1  
 **else**:  
 **for** xx, yy **in** brush.RBAll:  
 self.putpixel(xc+x + xx, yc+y - yy)  
 self.putpixel(xcm-x - xx, yc+y - yy)  
 self.putpixel(xc+x + xx, ycm-y + yy)  
 self.putpixel(xcm-x - xx, ycm-y + yy)  
 *# horizontal or diag* **if** d < 0:  
 e = 2 \* d + 2 \* y \* a2 - a2  
 *# horizontal* **if** e <= 0:  
 x += 1  
 d += 2 \* x \* b2 + b2  
 *# diag* **else**:  
 x += 1  
 y -= 1  
 d += (2 \* x \* b2 + b2 - 2 \* y \* a2 + a2)  
 *# vertical or diag* **else**:  
 e = 2 \* d - 2 \* x \* b2 - b2  
 *# vertical* **if** e > 0:  
 y -= 1  
 d += (-2 \* y \* a2 + a2)  
 *# diag* **else**:  
 x += 1  
 y -= 1  
 d += (2 \* x \* b2 + b2 - 2 \* y \* a2 + a2)

# Приложение 7. Рекурсивный алгоритм заливки

На вход алгоритму подается точка (cx, cy), начиная с которой требуется начать заливку области.

source = self.image.getpixel((cx, cy))  
color = self.brush\_color  
toFill = set()  
toFill.add((cx,cy))  
**while** toFill:  
 (x,y) = toFill.pop()  
 c = self.image.getpixel((x,y))  
 **if not** c == source:  
 **continue** self.image.putpixel((x,y), color)  
 **if** x-1>=0: toFill.add((x-1,y))  
 **if** x+1<self.width: toFill.add((x+1,y))  
 **if** y-1>=0: toFill.add((x,y-1))  
 **if** y+1<self.height: toFill.add((x,y+1))