|  |  |
| --- | --- |
| 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 № 6**

|  |  |
| --- | --- |
| **Тема:** Реализация и исследование алгоритма построчного затравочного заполнения  **Студент** ВоякинА. Я.  **Группа ИУ7-44Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:**

Реализация и исследование алгоритма построчного затравочного заполнения

**Техническое задание:**

1. Необходимо обеспечить ввод произвольной многоугольной области, содержащей произвольное количество отверстий. Ввод (вершин многоугольника) производить с помощью мыши, при этом для удобства пользователя должны отображаться ребра, соединяющие вводимые вершины. Предусмотреть ввод горизонтальных и вертикальных ребер. Должен быть предусмотрен ввод затравочной точки.
2. Пользователь должен иметь возможность задания цвета заполнения.
3. Работа программы должна предусматривать два режима – с задержкой и без задержки. Режим с задержкой должен позволить проследить выполняемую последовательность действий. (Задержку целесообразно выполнять после обработки очередной строки).
4. Обеспечить замер времени выполнения алгоритма (без задержки, с выводом на экран только окончательного результата).

**Теория:**

В построчном алгоритме затравочного заполнения мы помещаем в стек один затравочный пиксель для непрерывного интервала пикселей.

Алгоритм можно описать следующим образом:

1. Ввод исходных данных (информация о границах заполняемой области, затравочный пиксель, цвет границы, цвет заполнения).
2. Занесение затравочного пикселя в стек.
3. Пока стек не пуст, выполнять следующие действия:
   1. Извлечение пикселя из стека (x, y).
   2. Закраска пикселей текущей строки (y) влево и вправо от затравочного до встречи с граничным пикселем.

Пока цвет(x, y) < > цвет границы:

цвет(x, y) = цвет закраски.

x = x – 1, если движемся влево (x = x + 1, если движемся вправо).

* 1. Если цвет(x, y) = цвет границы, то xл = x + 1 (xп = x – 1).
  2. Поиск новых затравочных пикселей в интервале хл <= x <= хп на двух соседних строках по отношению к текущей (ув = у + 1, ун = у – 1).

В заданном интервале поиска необходимо найти не первый попавшийся затравочный пиксель, а до тех пор искать, пока не найдем **самый правый** затравочный пиксель). Это нужно делать, т. к. рассматриваемый интервал может прерываться граничными пикселями/закрашенными, т. е. **исходный интервал содержит несколько непрерывных интервалов**. Поэтому если возьмем левый, задача не выполнится. Нужно брать самый правый затравочный пиксель и помещать его в стек.

**Анализ эффективности:**

1. Цвет каждого пикселя меняется только один раз.
2. Количество обрабатываемых пикселей: обрабатываем пиксели внутри выбранной области, а также пиксели, расположенные на границе с областью закраски за пределами фигуры.
3. Размещаем в стеке не каждый затравочный пиксель, а один затравочный пиксель для непрерывного интервала пикселей.

**Инструкция к программе:**

В данной программе я предусмотрел рисование горизонтальный линий. Для этого необходимо на клавиатуре нажать кнопку «H», далее нажать кнопкой мыши, и координата Y у новой точки будет та же, что у предыдущей, для выключения режима рисования горизонтальных линий, необходимо повторно нажать на эту кнопку.

Тоже самое я реализовал и для рисования вертикальных линий, но на кнопку «V».

Для соединения последней точки и первой у многоугольника необходимо нажать клавишу Shift.

**Код алгоритма.**

****

Я прикрепил код алгоритма изображением, если вы хотите что-то скопировать, или тут слишком мелко то в самом низу я прикрепил полный код программы текстом.

В программе я реализовал выбор цвета заполнения и выбор цвета рисования границ.

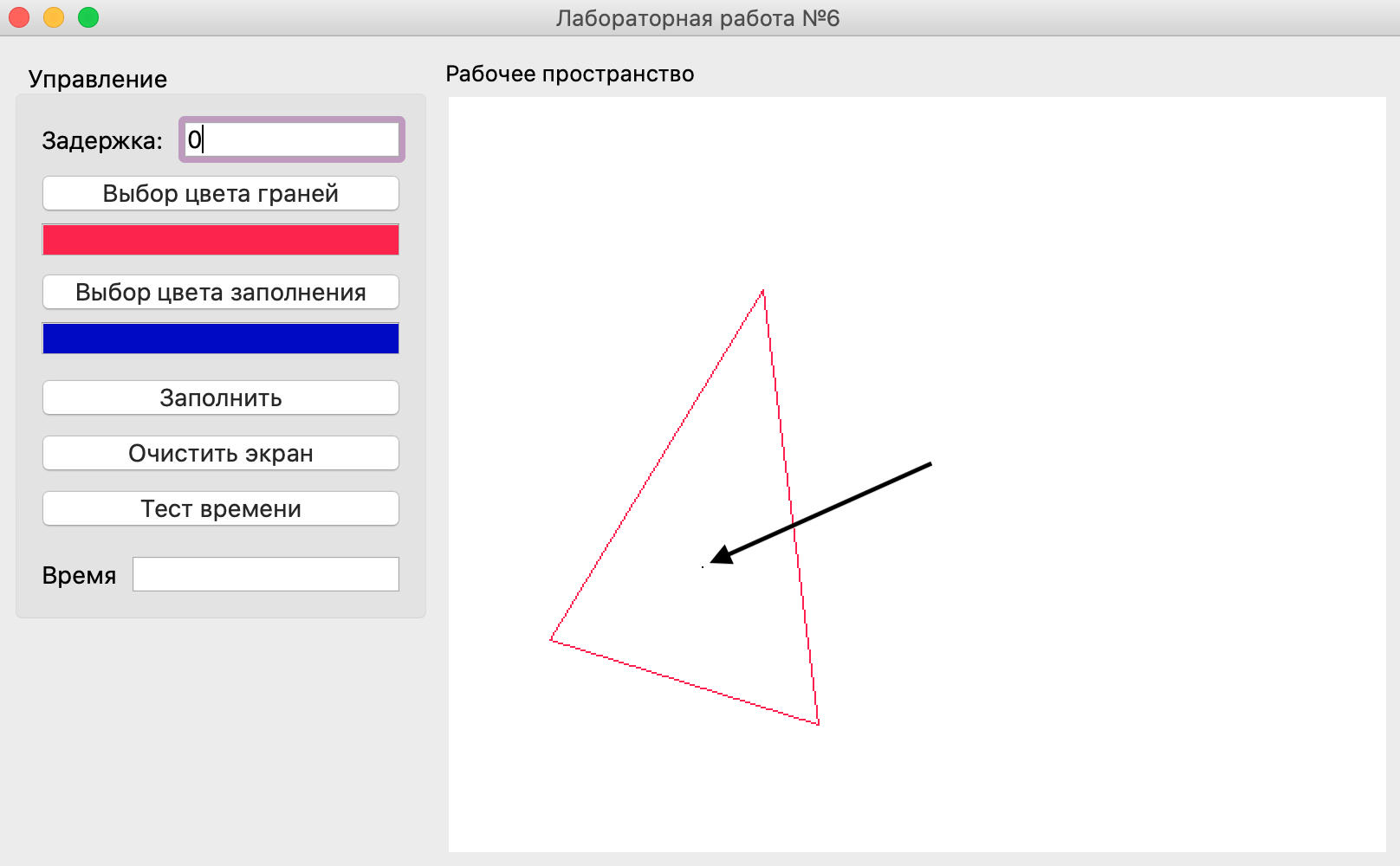
Границы я рисую с помощью целочисленного алгоритма Брезенхема.

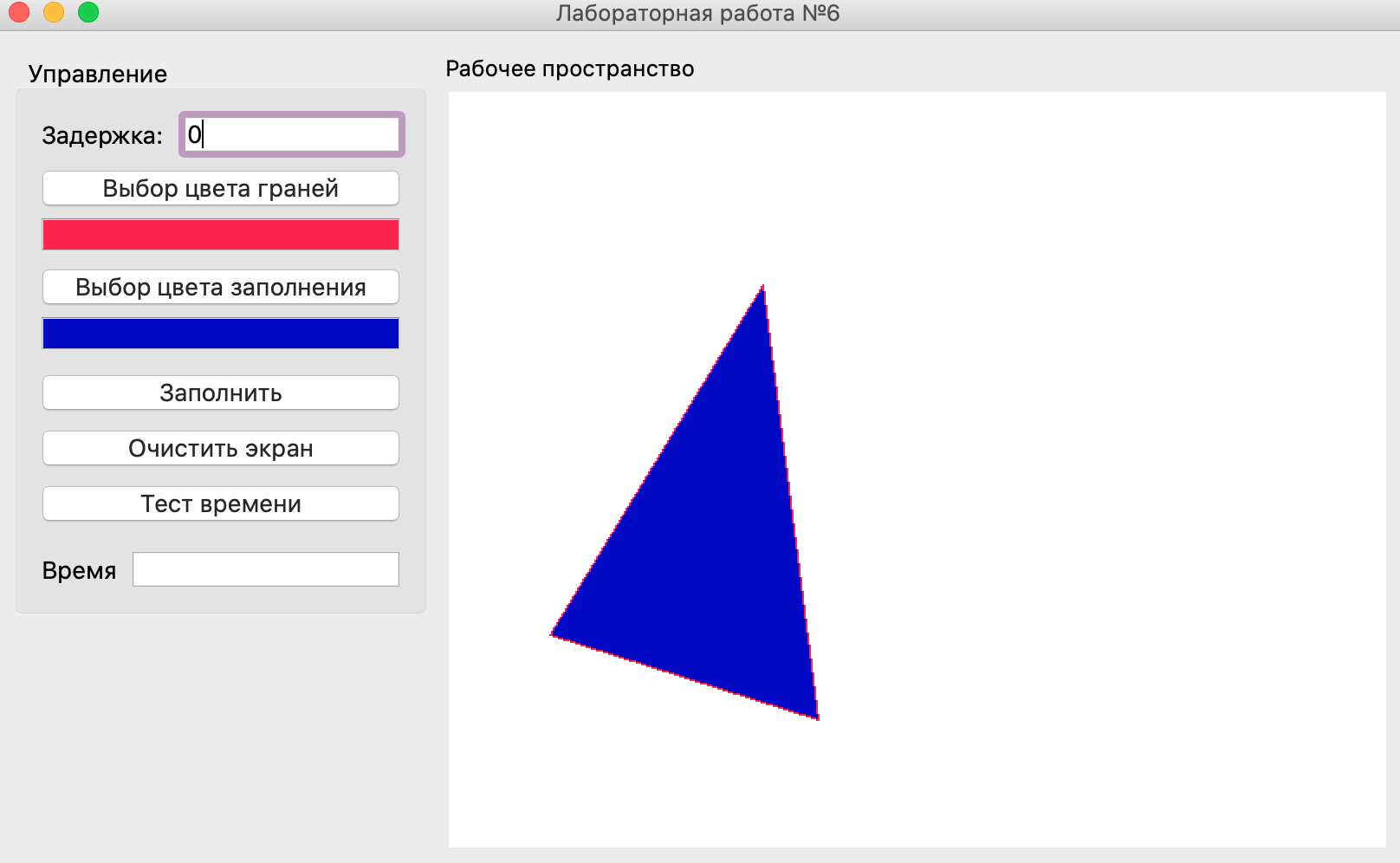
Затравочный пиксел рисуется чёрным цветом.

**Перейдём к демонстрации работы программы.**

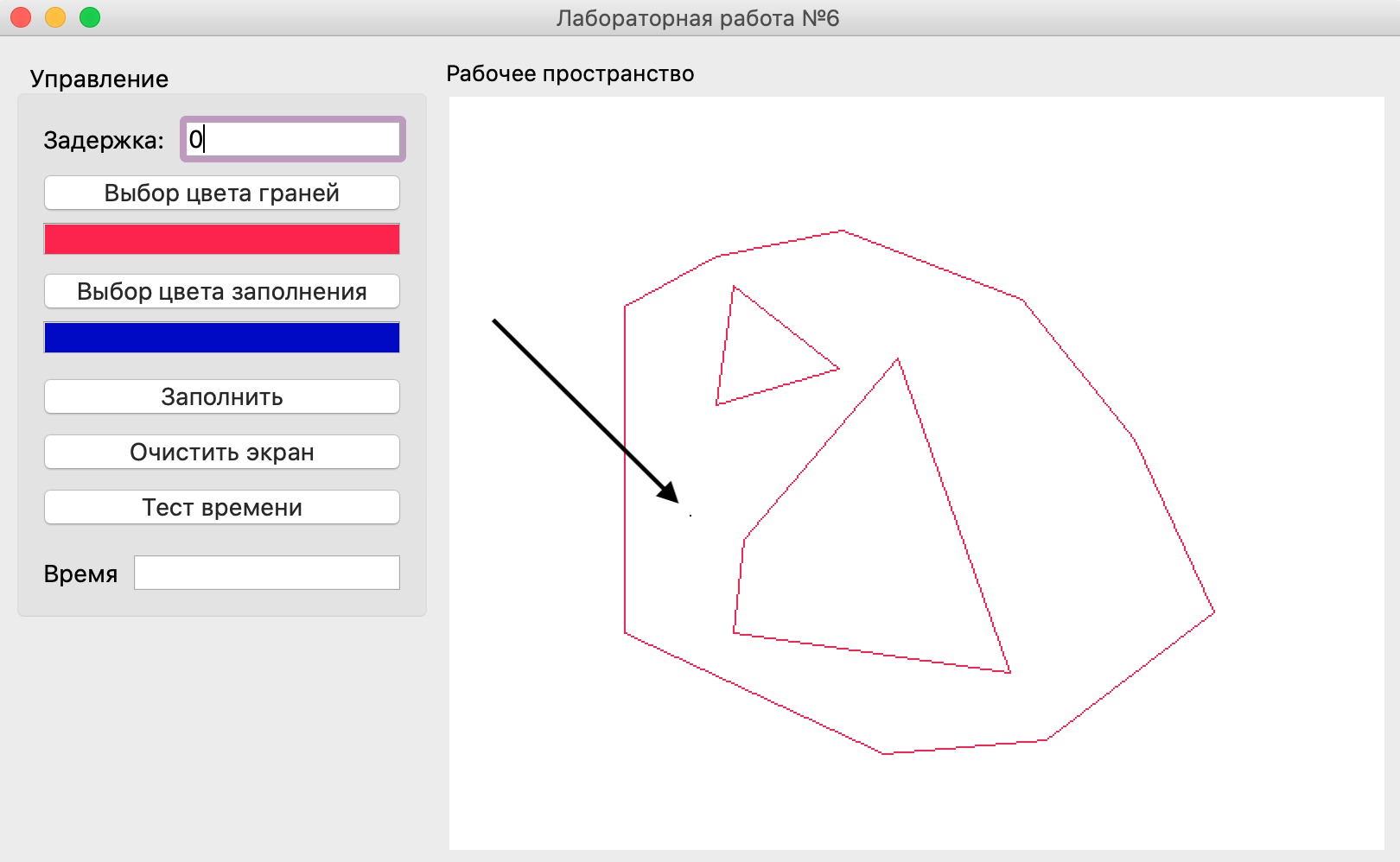
**Пример работы №1.**

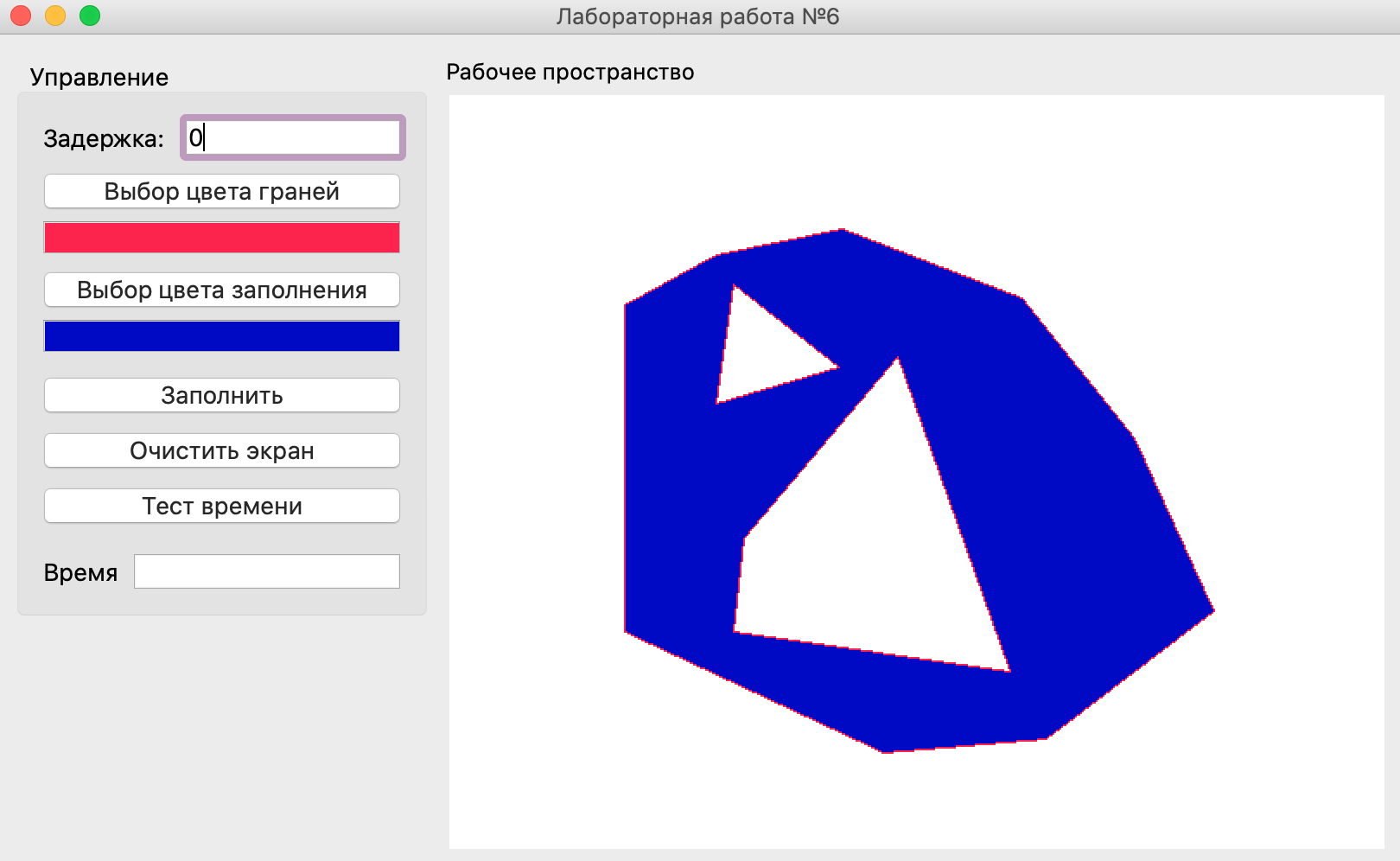
**Затравочные пиксели я буду помечать стрелочкой (т.к. они маленькие)**

****

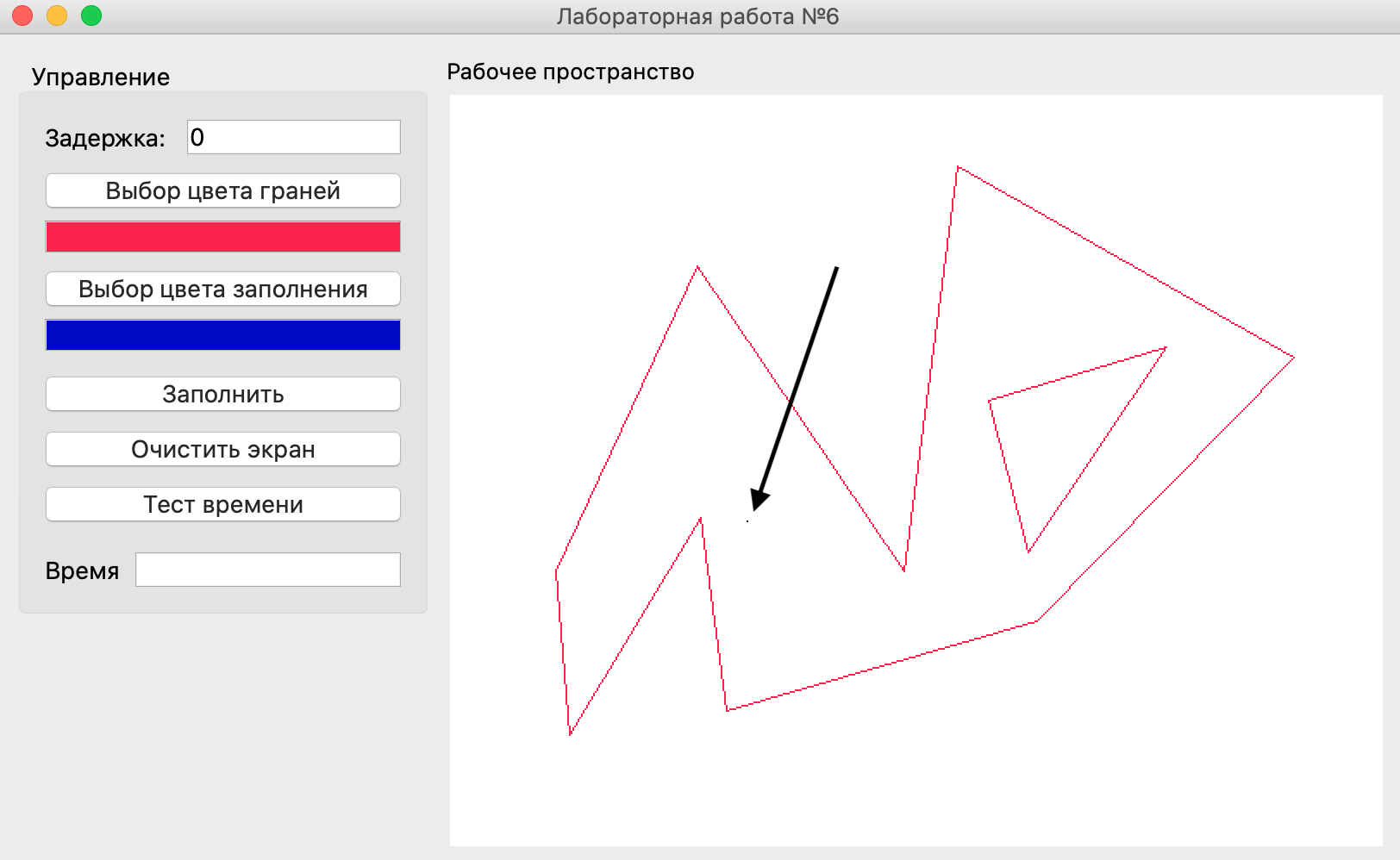
****

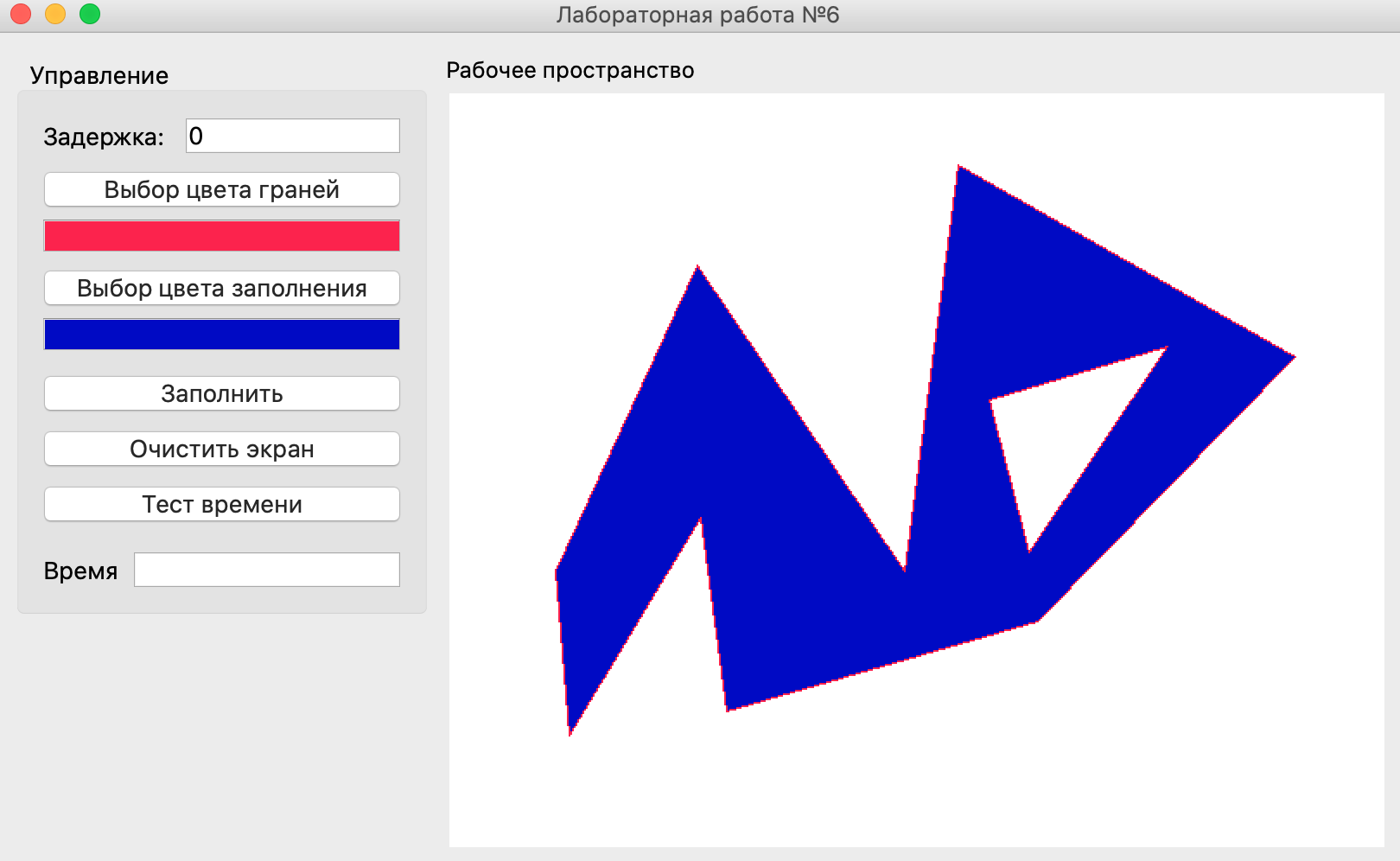
**Пример работы №2.**

****

****

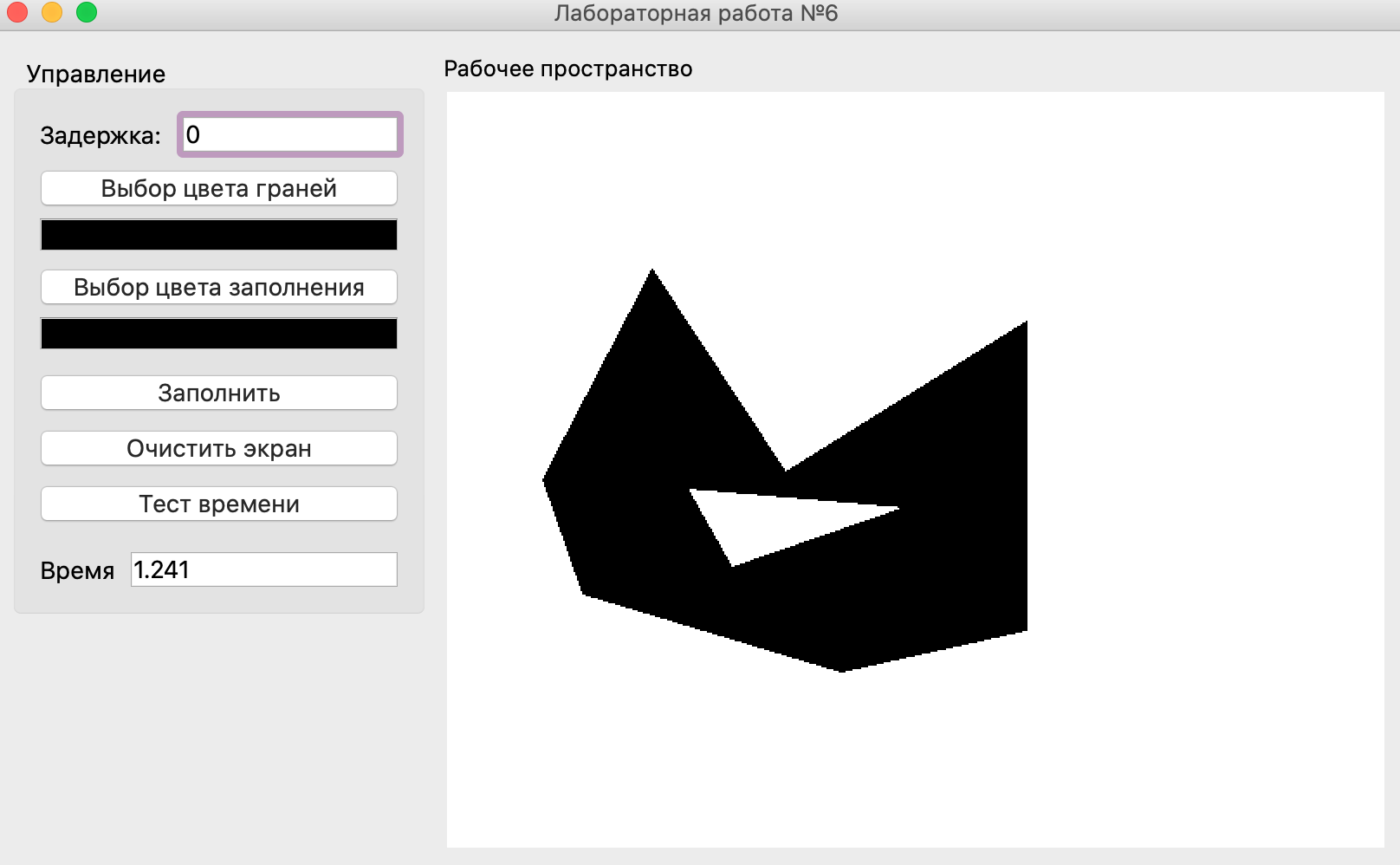
**Пример работы №3.**

****

****

**Демонстрацию работы заполнения с задержкой я прикрепил к письму, там gif файл, который должен проигрываться прям в браузере.**

**Демонстрация работы тестирования времени заполнения:**

****

**Полный код программы:**

import sys  
from PyQt5 import QtWidgets, QtGui, QtCore  
import design  
import numpy as np  
from math import fabs  
import time  
  
class Visual(QtWidgets.QMainWindow, design.Ui\_MainWindow):  
 def \_\_init\_\_(*self*):  
 super().\_\_init\_\_()  
 *self*.setupUi(*self*)  
  
 *self*.label.setPalette(QtGui.QPalette(QtCore.Qt.white))  
 *self*.white = QtGui.QColor(QtCore.Qt.white)  
 *self*.black = QtGui.QColor(QtCore.Qt.black)  
 *self*.colour\_fill = QtGui.QColor(QtCore.Qt.black)  
 *self*.colour\_edge = QtGui.QColor(QtCore.Qt.black)  
  
 *self*.labelColourEdges.setPalette(QtGui.QPalette(QtCore.Qt.black))  
 *self*.labelColourFill.setPalette(QtGui.QPalette(QtCore.Qt.black))  
  
 *self*.pm = QtGui.QPixmap(541, 436)  
 *self*.pm.fill(*self*.white)  
 *self*.img = *self*.pm.toImage()  
  
 *self*.horizontal = False  
 *self*.vertical = False  
  
 *self*.objX = np.array([], dtype=int)  
 *self*.objY = np.array([], dtype=int)  
 *self*.pixel\_x = 0  
 *self*.pixel\_y = 0  
  
 # Связи кнопок и функций.  
 *self*.pushButtonScreenClean.clicked.connect(*self*.clean\_screen)  
 *self*.pushButtonChooseColourEdges.clicked.connect(*self*.choose\_color\_edge)  
 *self*.pushButtonChooseColourFill.clicked.connect(*self*.choose\_color\_fill)  
 *self*.pushButtonFill.clicked.connect(*self*.fill)  
 *self*.pushButtonTime.clicked.connect(*self*.count\_time)  
  
 def bres\_int(*self*, x\_start, y\_start, x\_end, y\_end):  
 dx = x\_end - x\_start  
 dy = y\_end - y\_start  
 sx = sign(dx)  
 sy = sign(dy)  
 dx = fabs(dx)  
 dy = fabs(dy)  
 if dy >= dx:  
 dx, dy = dy, dx  
 fl = 1  
 else:  
 fl = 0  
 f = 2 \* dy - dx  
 x = round(x\_start)  
 y = round(y\_start)  
 i = 1  
 while i <= dx + 1:  
 *self*.img.setPixel(x, y, *self*.colour\_edge.rgb())  
 if f >= 0:  
 if fl == 1:  
 x += sx  
 else:  
 y += sy  
 f -= 2 \* dx  
 if f <= 0:  
 if fl == 1:  
 y += sy  
 else:  
 x += sx  
 f += 2 \* dy  
 i += 1  
  
 def choose\_color\_fill(*self*):  
 *self*.colour\_fill = QtWidgets.QColorDialog.getColor()  
 *self*.labelColourFill.setPalette(QtGui.QPalette(*self*.colour\_fill))  
  
 def choose\_color\_edge(*self*):  
 *self*.colour\_edge = QtWidgets.QColorDialog.getColor()  
 *self*.labelColourEdges.setPalette(QtGui.QPalette(*self*.colour\_edge))  
  
 def mousePressEvent(*self*, press):  
 *self*.img = *self*.pm.toImage()  
 if press.button() == QtCore.Qt.LeftButton:  
 if *self*.vertical:  
 *self*.objX = np.append(*self*.objX, *self*.objX[*self*.objX.size - 1])  
 else:  
 *self*.objX = np.append(*self*.objX, (press.pos() - *self*.label.pos()).x())  
 if *self*.horizontal:  
 *self*.objY = np.append(*self*.objY, *self*.objY[*self*.objY.size - 1])  
 else:  
 *self*.objY = np.append(*self*.objY, (press.pos() - *self*.label.pos()).y())  
 if *self*.objX.size >= 2:  
 *self*.bres\_int(*self*.objX[*self*.objX.size - 2], *self*.objY[*self*.objY.size - 2],  
 *self*.objX[*self*.objX.size - 1], *self*.objY[*self*.objY.size - 1])  
 elif press.button() == QtCore.Qt.RightButton:  
 *self*.pixel\_x = (press.pos() - *self*.label.pos()).x()  
 *self*.pixel\_y = (press.pos() - *self*.label.pos()).y()  
 *self*.img.setPixel(*self*.pixel\_x, *self*.pixel\_y, *self*.black.rgb())  
 *self*.pm = *self*.pm.fromImage(*self*.img)  
 *self*.label.setPixmap(*self*.pm)  
  
 def keyPressEvent(*self*, press):  
 *self*.img = *self*.pm.toImage()  
 if int(press.key()) == QtCore.Qt.Key\_Shift:  
 if *self*.objX.size >= 3:  
 *self*.bres\_int(*self*.objX[0], *self*.objY[0], *self*.objX[*self*.objX.size - 1], *self*.objY[*self*.objY.size - 1])  
 *self*.objX = np.array([], dtype=int)  
 *self*.objY = np.array([], dtype=int)  
 elif int(press.key()) == QtCore.Qt.Key\_H:  
 *self*.horizontal = not *self*.horizontal  
 *self*.vertical = False  
 elif int(press.key()) == QtCore.Qt.Key\_V:  
 *self*.vertical = not *self*.vertical  
 *self*.horizontal = False  
 *self*.pm = *self*.pm.fromImage(*self*.img)  
 *self*.label.setPixmap(*self*.pm)  
  
 def wait(*self*):  
 t = int(*self*.lineEditDelay.text())  
 if t > 0:  
 *self*.update\_image()  
 loop = QtCore.QEventLoop()  
 QtCore.QTimer.singleShot(t, loop.quit)  
 loop.exec\_()  
  
 def update\_image(*self*):  
 *self*.pm = *self*.pm.fromImage(*self*.img)  
 *self*.label.setPixmap(*self*.pm)  
 *self*.label.repaint()  
  
 def fill(*self*):  
 stack = [*self*.pixel\_y, *self*.pixel\_x] # Стек с начальным затр. пикселем.  
 while len(stack) != 0: # Цикл пока стек не пуст.  
 p\_x = stack.pop(); p\_y = stack.pop() # Pop координат затравочного пиксела из стека.  
 x = p\_x; y = p\_y # Получение координат затравочного пиксела.  
 x\_buff = x # Сохранение начальной позиции.  
 while *self*.img.pixelColor(x, y).rgb() \  
 != *self*.colour\_edge.rgb(): # Цикл пока цвет пиксела отличен от цвета грани.  
 if x == 0:  
 return  
 *self*.img.setPixel(x, y, *self*.colour\_fill.rgb()) # Высвечиваем пиксел цветом заполнения.  
 x -= 1 # Движемся влево.  
 x\_left = x + 1 # Запомнили левую границу.  
 x = x\_buff + 1 # Вернулись на исходное положение заполнения + 1.  
 while *self*.img.pixelColor(x, y).rgb() \  
 != *self*.colour\_edge.rgb(): # Цикл пока цвет пиксела отличен от цвета грани.  
 *self*.img.setPixel(x, y, *self*.colour\_fill.rgb()) # Высвечивание пиксела цветом заполнения.  
 x += 1 # Движемся вправо.  
 x\_right = x - 1 # Запомнили правую границу.  
 x = x\_left; y += 1 # Переходим на строку выше в левую границу.  
 for i in range(0, 2): # Обрабатываем две строки (Выше и ниже).  
 while x <= x\_right: # Цикл до правой границы.  
 flag = False # Наличие пикселов для высвечивания.  
 while *self*.img.pixelColor(x, y).rgb() != *self*.colour\_edge.rgb() and \  
 *self*.img.pixelColor(x, y).rgb() != *self*.colour\_fill.rgb() and \  
 x <= x\_right: # Цикл пока пиксел пуст.  
 flag = True # Пикселы для высвечивания есть.  
 x += 1 # Движемся вправо.  
 if flag: # Если есть пикселы для высвечивания.  
 if *self*.img.pixelColor(x, y).rgb() != *self*.colour\_edge.rgb() and \  
 *self*.img.pixelColor(x, y).rgb() != *self*.colour\_fill.rgb() and \  
 x == x\_right: # Если достигли правой границы и пуст.  
 stack.append(y); stack.append(x) # Заносим пиксел в стек.  
 else:  
 stack.append(y); stack.append(x - 1) # Иначе заносим предыдущий пиксел в стек.  
 x\_buff = x # Сохраняем текущий X.  
 while *self*.img.pixelColor(x, y).rgb() == *self*.colour\_fill.rgb() and \  
 x < x\_right: # Пока встречены заполненные пикселы.  
 x += 1 # Движемся вправо.  
 if x == x\_buff: # Если X не изменился.  
 x += 1 # Сдвигаемся вправо.  
 y -= 2 # Переходим на строку вниз от начальной.  
 x = x\_left # Переходим на левую границу.  
 *self*.wait() # Задержка если она больше 0  
 *self*.update\_image() # Обновление изображения  
  
 def count\_time(*self*):  
 start = time.time()  
 *self*.fill()  
 *self*.lineEditTime.setText(str(round(time.time() - start, 4)))  
 *self*.lineEditTime.repaint()  
  
 def clean\_screen(*self*):  
 *self*.pm.fill(*self*.white)  
 *self*.img = *self*.pm.toImage()  
 *self*.label.setPixmap(*self*.pm)  
 *self*.label.repaint()  
 *self*.objX = np.array([], dtype=int)  
 *self*.objY = np.array([], dtype=int)  
  
def sign(x):  
 if x < 0:  
 return -1  
 elif x > 0:  
 return 1  
 return 0  
  
def main():  
 app = QtWidgets.QApplication(sys.argv)  
 window = Visual()  
 window.show()  
 app.exec\_()