|  |  |
| --- | --- |
| 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 И ИССЛЕДОВАНИЕ АЛГОРИТМОВ РАСТРОВОГО ЗАПОЛНЕНИЯ СПЛОШНЫХ ОБЛАСТЕЙ  **Студент** ВоякинА. Я.  **Группа ИУ7-44Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:**

Реализация и исследование одного из алгоритмов (Алгоритм заполнения со списком ребер и флагом) растрового заполнения области.

**Техническое задание:**

В рамках данной работы необходимо реализовать один из алгоритмов заполнения сплошной области.

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.

Пользователь должен иметь возможность задания цвета заполнения.

Работа программы должна предусматривать два режима – с задержкой и без задержки.

**Теоретический материал:**

Одной из особенностей растровых дисплеев является возможность представления сплошных областей. Растровая развертка сплошных областей (заполнение многоугольников или заполнение контуров) - генерация сплошных областей на основе простых описаний ребер или вершин. Для решения этой задачи используются различные методы, которые обычно разделяют на две категории: растровая развертка и затравочное заполнение.

В растровых методах делается попытка определить в порядке сканирования строк принадлежность точки внутренней области контура или многоугольника. Эти алгоритмы обычно просматривают многоугольники (контуры) от верхней точки до нижней. Методы растровой развертки применимы обычно и к векторным дисплеям, в которых они используются для штриховки или закраски контуров.

**Алгоритм заполнения со списком ребер и флагом.**

Алгоритм со списком ребер и флагом - двух шаговый алгоритм. Во-первых, обрисовывается контур, ограничивающий область, в результате чего на каждой строке сканирования определяются пары ограничивающих пикселов, а во-вторых, активизируются пикселы, расположенные между вычисленными на предыдущем шаге ограничивающими пикселами.

Процесс реализации алгоритма со списком ребер и флагом состоит из следующих этапов:

1. Обрисовка ребер многоугольника целочисленным методом Брезенхема.

2. Заполнение области. Для каждой строки сканирования, имеющей точки пересечения с многоугольником, ограничивающим область, выполняется следующая последовательность действий:

а) f=0, (f-промежуточная переменная (флаг), показывающая расположение очередного пиксела: fl=0 - пиксел лежит вне области заполнения, fl=1 - пиксел лежит внутри заполняемой области);

x=xl (xl - левая граница);

б) пока (x<=xr ) (xr - правая граница) выполнить следующие действия:

если пиксел (x,y) имеет граничное значение , то проинвертировать значение переменной f (f=0, если было f=1; f=1, если было f=0);

если f=1, то присвоить пикселу (x,y) цвет многоугольника, в противном случае, присвоить пикселу (x,y) цвет фона;

x=x+1.

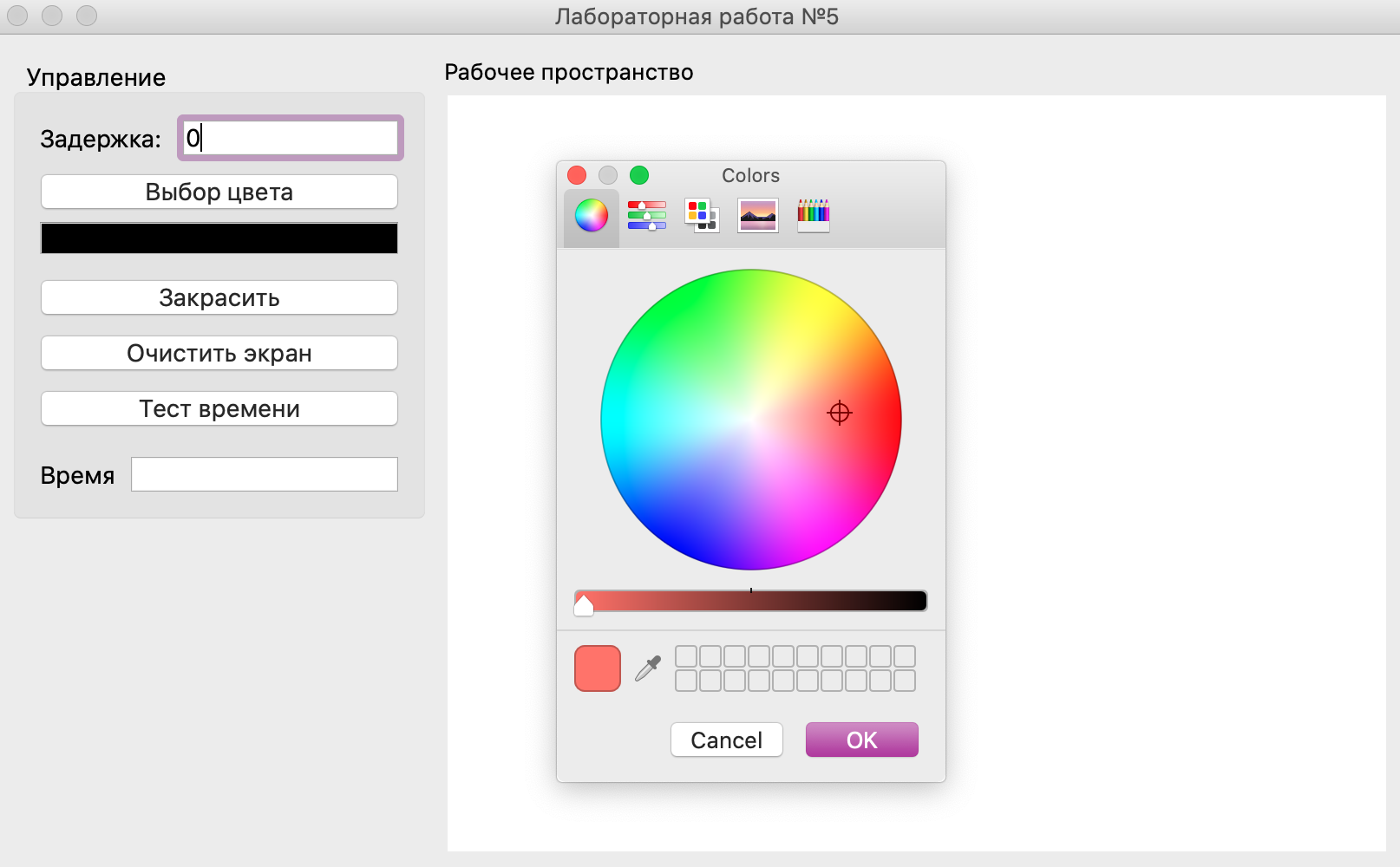
в) переход к следующей сканирующей строке y=y+1.

В рассмотренном алгоритме каждый пиксел обрабатывается только один раз, поэтому затраты на ввод/вывод значительно меньше, чем в алгоритме по ребрам или алгоритме с перегородкой.

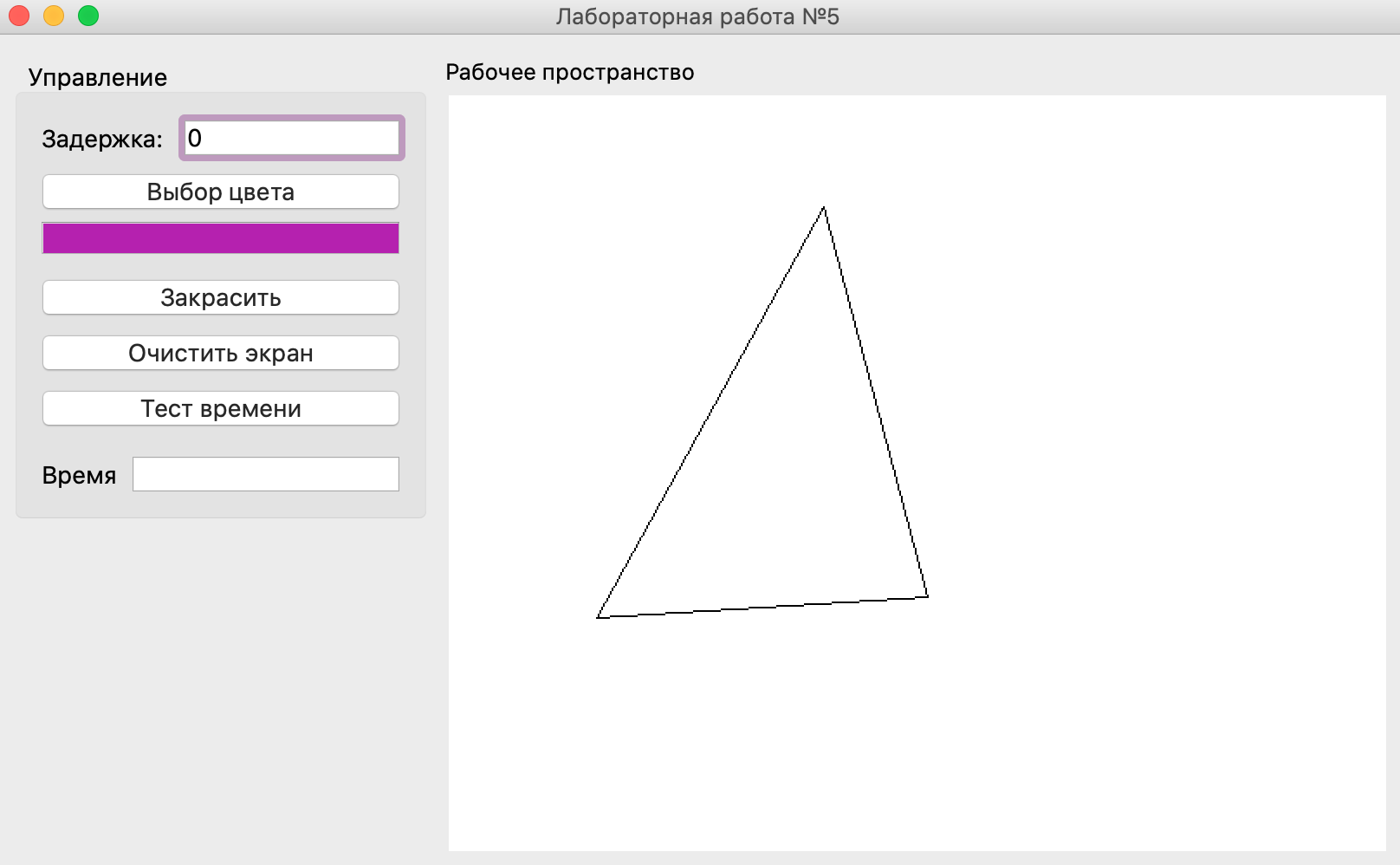
**Реализация:**

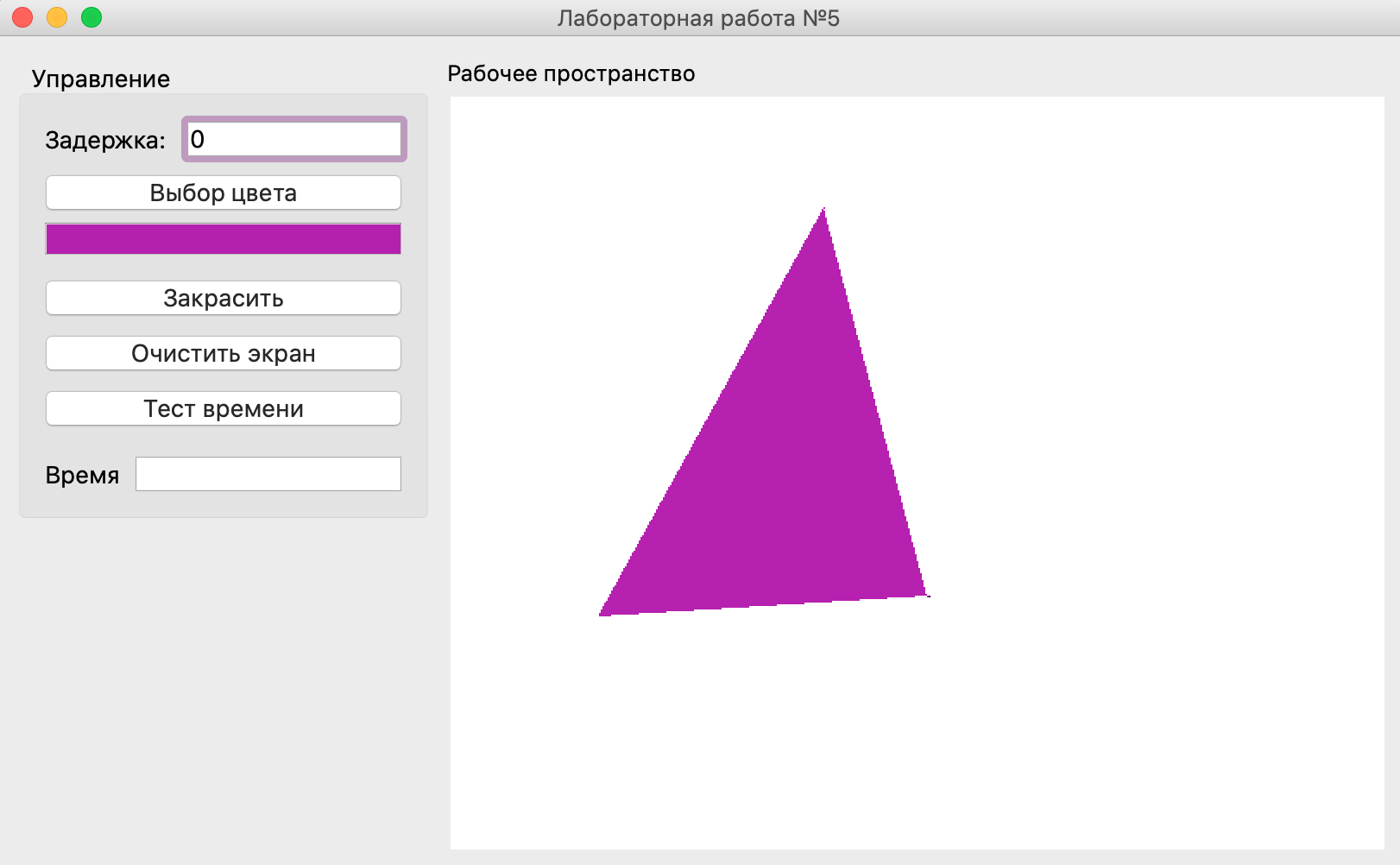
*def* fill(*self*, draw=*True*, delay=*True*):  
 *for* y *in* range(*self*.minY, *self*.maxY + 1):  
 flag = *False  
 for* x *in* range(*self*.minX, *self*.maxX + 1):  
 *if self*.imgbg.pixelColor(x, y) == *self*.black:  
 flag = *not* flag  
 *if* flag:  
 *self*.imgbg.setPixel(x, y, *self*.colour.rgb())  
 *else*:  
 *self*.imgbg.setPixel(x, y, *self*.white.rgb())  
 *if* delay *and* int(*self*.lineEditDelay.text()) > 0:  
 *self*.update\_image()  
 *self*.wait(int(*self*.lineEditDelay.text()))  
 *self*.update\_image()

Окно выбора цвета заполнения:

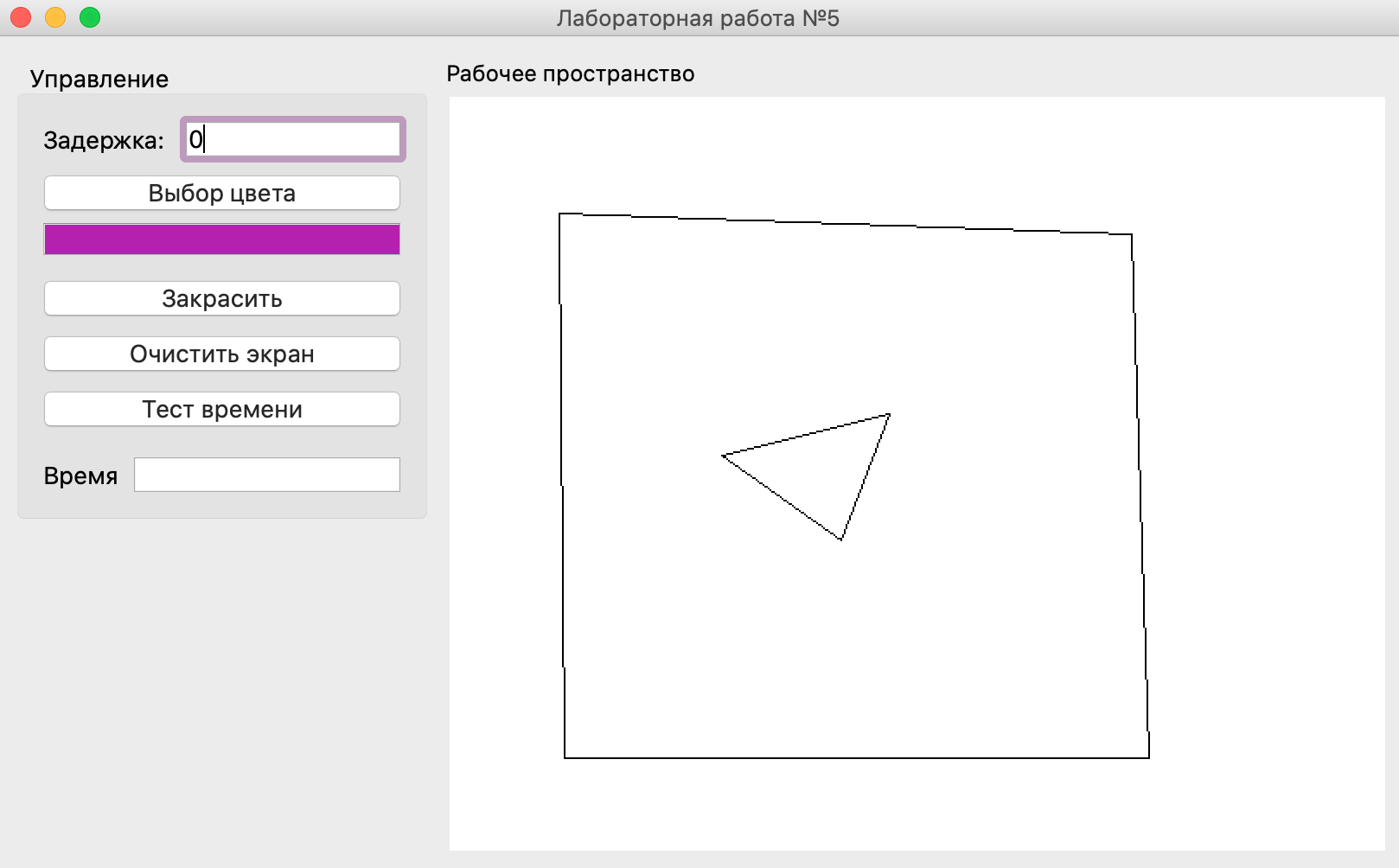


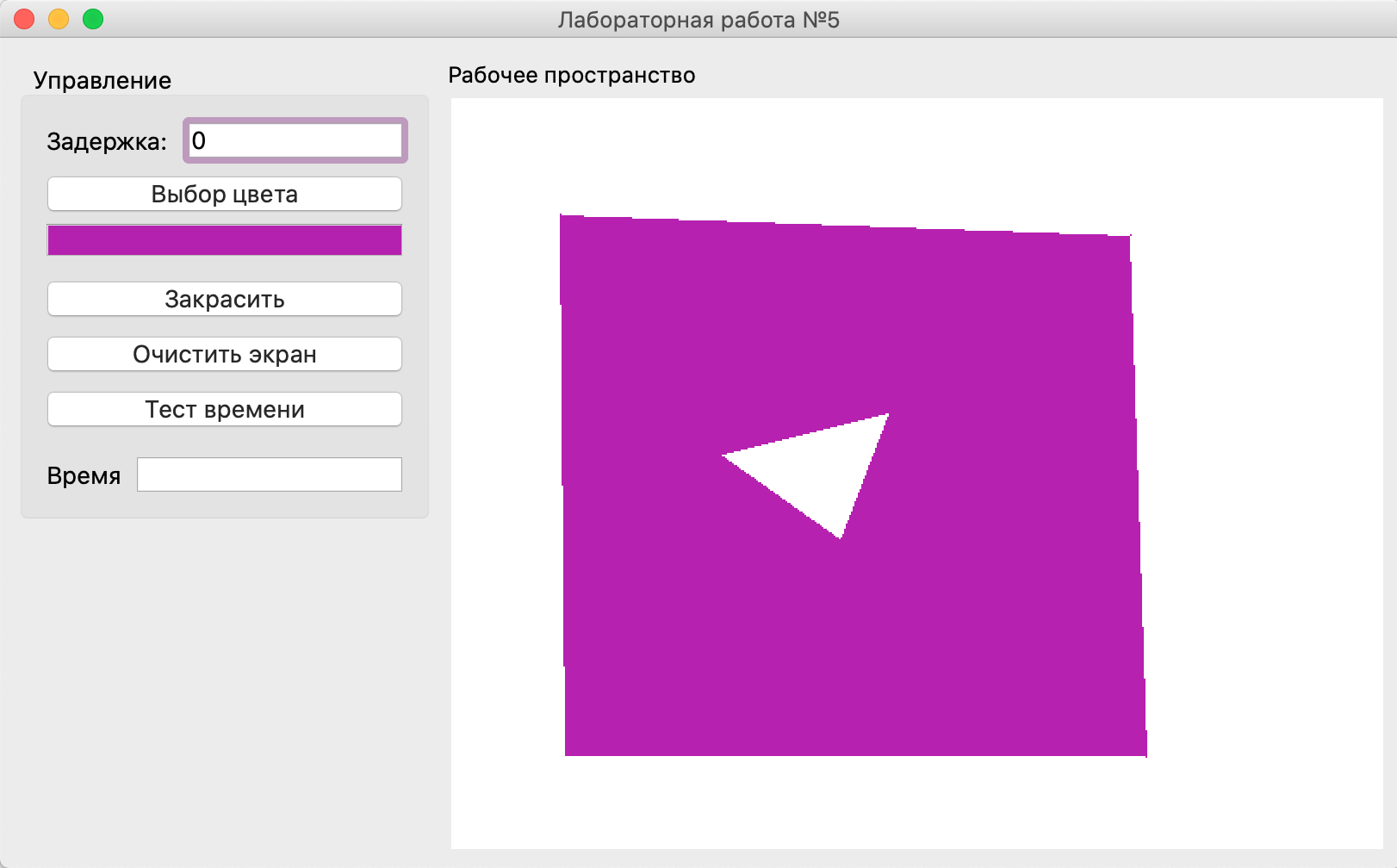
Пример заполнения №1:



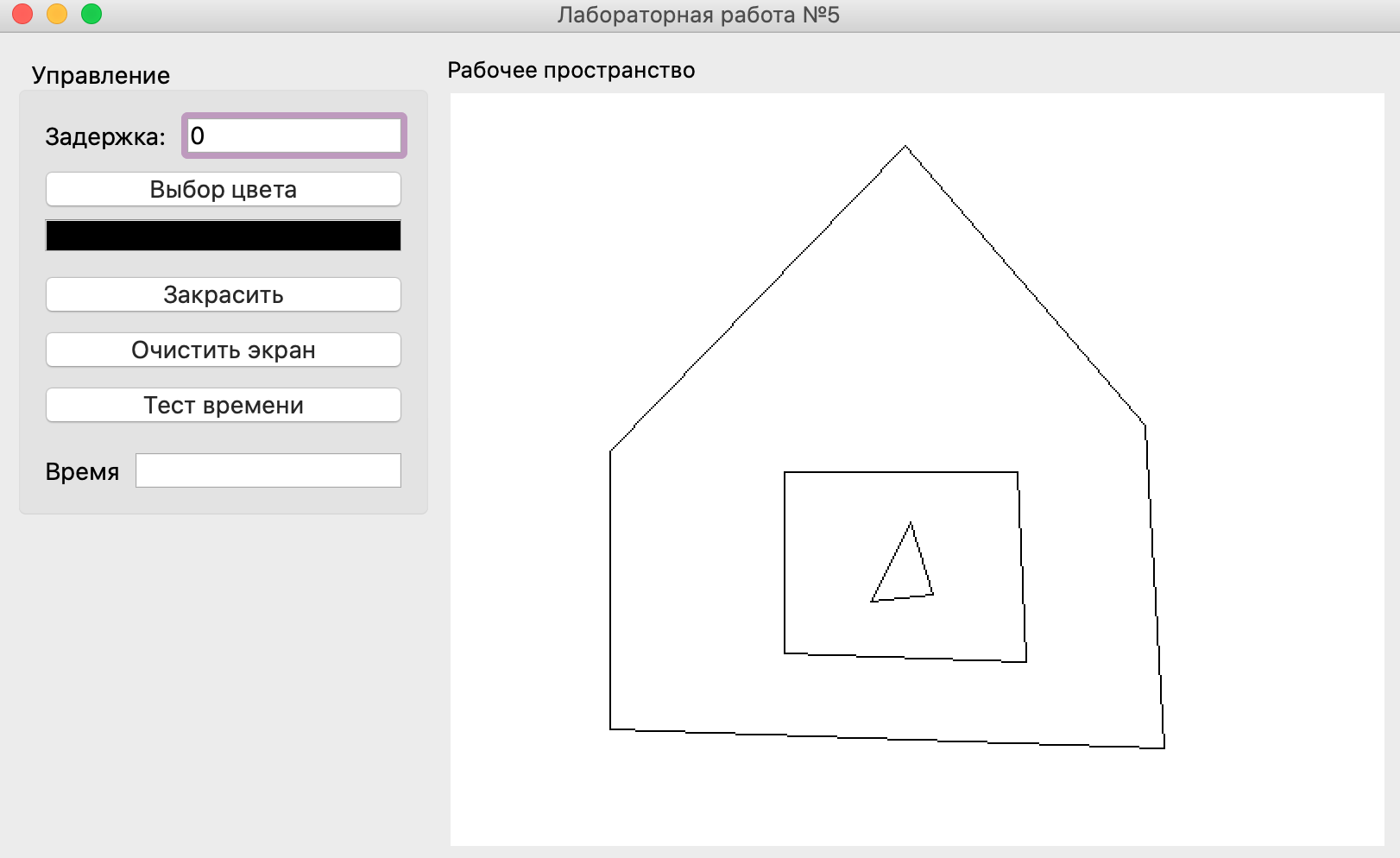


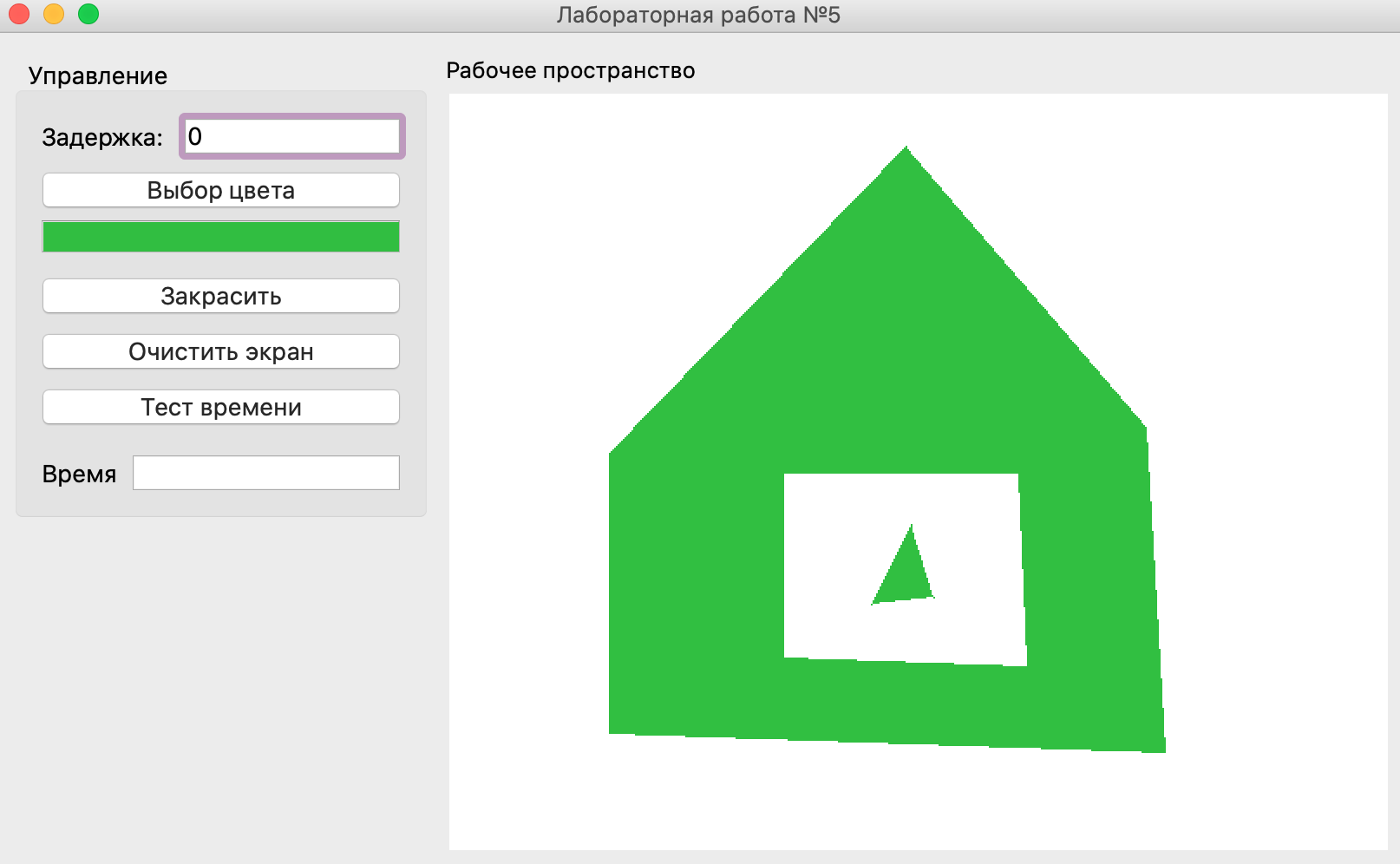
Пример заполнения №2:



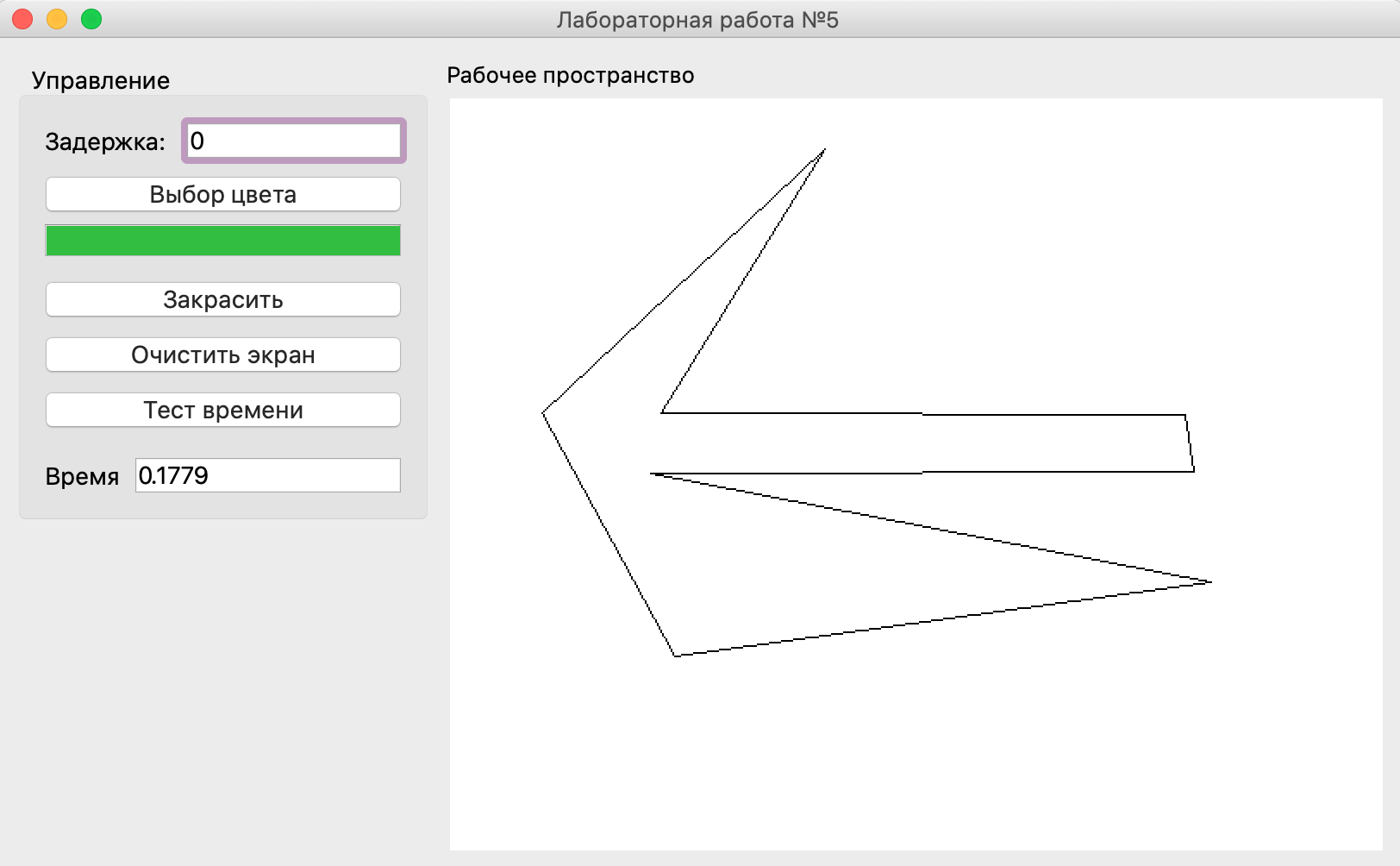


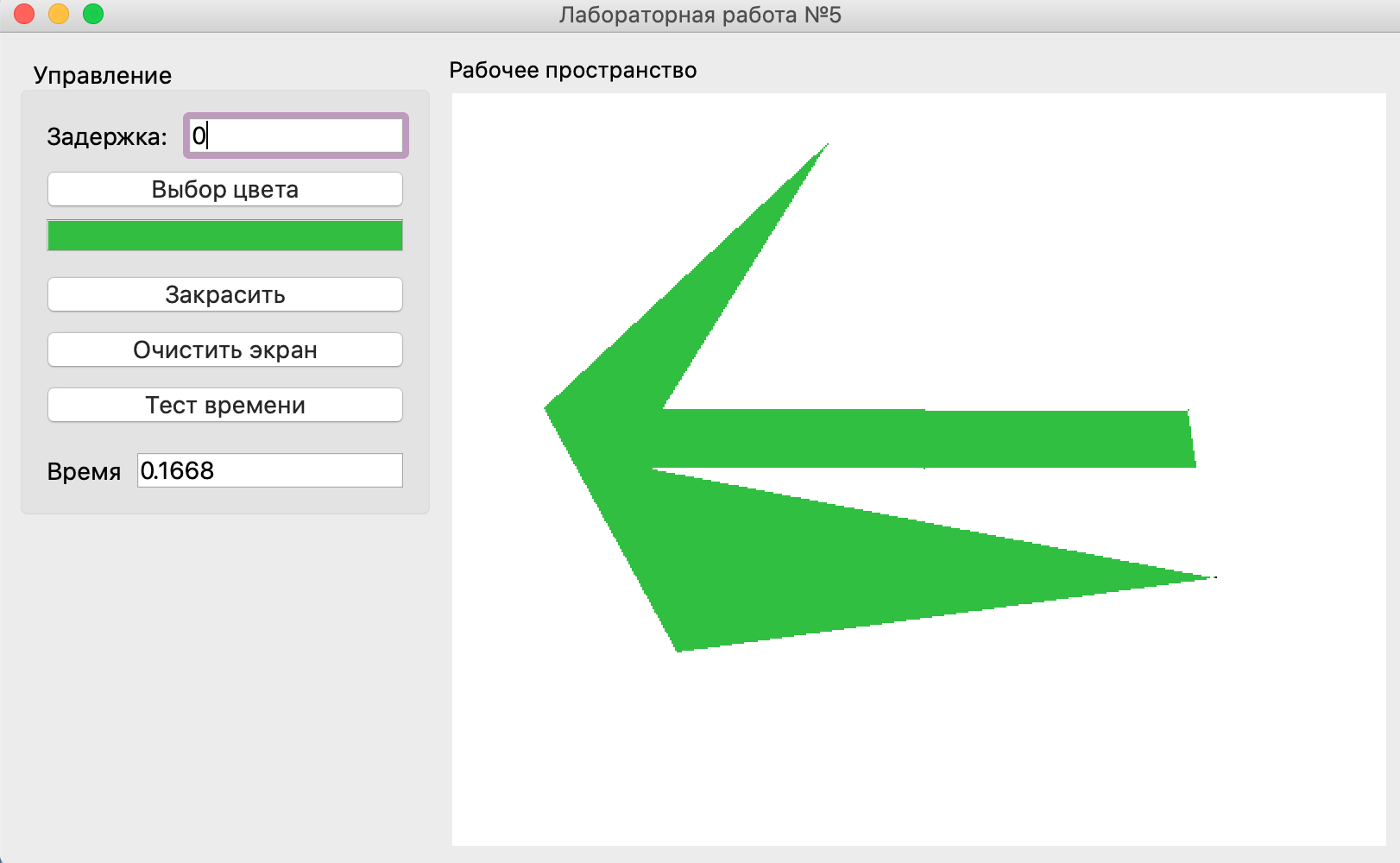
Пример заполнения №3:





Пример тестирования времени заполнения:





Пример заполнения с задержкой не могу продемонстрировать из-за демонстрации работы с помощью изображений.

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*.black = QtGui.QColor(QtCore.Qt.black)  
 *self*.white = QtGui.QColor(QtCore.Qt.white)  
 *self*.colour = QtGui.QColor(QtCore.Qt.black)  
 *self*.labelColour.setPalette(QtGui.QPalette(QtCore.Qt.black))  
 *self*.labelColour.setAutoFillBackground(*True*)  
  
 *self*.pm = QtGui.QPixmap(541, 436)  
 *self*.pm.fill(*self*.white)  
 *self*.img = *self*.pm.toImage()  
 *self*.imgbg = *self*.pm.toImage()  
  
 *self*.maxX = 0  
 *self*.minX = 1500  
 *self*.maxY = 0  
 *self*.minY = 1500  
 *self*.objX = np.array([], dtype=int)  
 *self*.objY = np.array([], dtype=int)  
  
 *# Связи кнопок и функций.  
 self*.pushButtonScreenClean.clicked.connect(*self*.clean\_screen)  
 *self*.pushButtonChooseColour.clicked.connect(*self*.choose\_color)  
 *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*.black.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* bres\_int\_special(*self*, x\_start, y\_start, x\_end, y\_end):  
 dx = fabs(x\_end - x\_start)  
 dy = fabs(y\_end - y\_start)  
 stepx = sign(x\_end - x\_start)  
 stepy = sign(y\_end - y\_start)  
 *if* dy > dx:  
 dx, dy = dy, dx  
 flag = 1  
 *else*:  
 flag = 0  
 e = 2 \* dy - dx  
 lasty = y\_start  
 x = x\_start  
 y = y\_start  
 *for* i *in* range(0, round(dx)):  
 *if* lasty != y:  
 lasty = y  
 *self*.imgbg.setPixel(x, y, *self*.black.rgb())  
 *if* e >= 0:  
 *if* flag == 1:  
 x += stepx  
 *else*:  
 y += stepy  
 e -= 2 \* dx  
 *if* e < 0:  
 *if* flag == 1:  
 y += stepy  
 *else*:  
 x += stepx  
 e += 2 \* dy  
  
 *def* add\_extra\_pixels(*self*, minX, minY, maxX, maxY):  
 *for* y *in* range(minY, maxY + 1):  
 flag = *False  
 for* x *in* range(minX, maxX + 1):  
 *if* flag *and self*.imgbg.pixelColor(x, y) == *self*.black:  
 flag = *False  
 continue  
 if self*.imgbg.pixelColor(x, y) == *self*.black:  
 flag = *True* bx = x  
 by = y  
 *if* flag:  
 *self*.imgbg.setPixel(bx + 1, by, *self*.black.rgb())  
  
  
 *def* choose\_color(*self*):  
 *self*.colour = QtWidgets.QColorDialog.getColor()  
 *self*.labelColour.setPalette(QtGui.QPalette(*self*.colour))  
  
 *def* mousePressEvent(*self*, press):  
 *self*.img = *self*.pm.toImage()  
 *if* press.button() == QtCore.Qt.LeftButton:  
 *self*.objX = np.append(*self*.objX, (press.pos() - *self*.label.pos()).x())  
 *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])  
 *self*.bres\_int\_special(*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:  
 *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*.bres\_int\_special(*self*.objX[0], *self*.objY[0],  
 *self*.objX[*self*.objX.size - 1], *self*.objY[*self*.objY.size - 1])  
 *self*.add\_extra\_pixels(min(*self*.objX), min(*self*.objY), max(*self*.objX), max(*self*.objY))  
 *if self*.minX > min(*self*.objX):  
 *self*.minX = min(*self*.objX)  
 *if self*.maxX < max(*self*.objX):  
 *self*.maxX = max(*self*.objX)  
 *if self*.minY > min(*self*.objY):  
 *self*.minY = min(*self*.objY)  
 *if self*.maxY < max(*self*.objY):  
 *self*.maxY = max(*self*.objY)  
 *self*.objX = np.array([], dtype=int)  
 *self*.objY = np.array([], dtype=int)  
 *self*.pm = *self*.pm.fromImage(*self*.img)  
 *self*.label.setPixmap(*self*.pm)  
  
 *def* wait(*self*, t):  
 loop = QtCore.QEventLoop()  
 QtCore.QTimer.singleShot(t, loop.quit)  
 loop.exec\_()  
  
 *def* update\_image(*self*):  
 *self*.pm = *self*.pm.fromImage(*self*.imgbg)  
 *self*.label.setPixmap(*self*.pm)  
 *self*.label.repaint()  
  
 *def* fill(*self*, draw=*True*, delay=*True*):  
 *for* y *in* range(*self*.minY, *self*.maxY + 1):  
 flag = *False  
 for* x *in* range(*self*.minX, *self*.maxX + 1):  
 *if self*.imgbg.pixelColor(x, y) == *self*.black:  
 flag = *not* flag  
 *if* flag:  
 *self*.imgbg.setPixel(x, y, *self*.colour.rgb())  
 *else*:  
 *self*.imgbg.setPixel(x, y, *self*.white.rgb())  
 *if* delay *and* int(*self*.lineEditDelay.text()) > 0:  
 *self*.update\_image()  
 *self*.wait(int(*self*.lineEditDelay.text()))  
 *self*.update\_image()  
  
 *def* count\_time(*self*):  
 start = time.time()  
 *self*.fill(*True*, *False*)  
 *self*.lineEditTime.setText(str(round(time.time() - start, 4)))  
 *self*.lineEditTime.repaint()  
  
 *def* clean\_screen(*self*):  
 *self*.pm.fill(*self*.white)  
 *self*.imgbg = *self*.pm.toImage()  
 *self*.label.setPixmap(*self*.pm)  
 *self*.label.repaint()  
 *self*.objX = np.array([], dtype=int)  
 *self*.objY = np.array([], dtype=int)  
 *self*.minX = 1500  
 *self*.minY = 1500  
 *self*.maxX = 0  
 *self*.maxY = 0  
  
*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\_()