|  |  |
| --- | --- |
| 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**

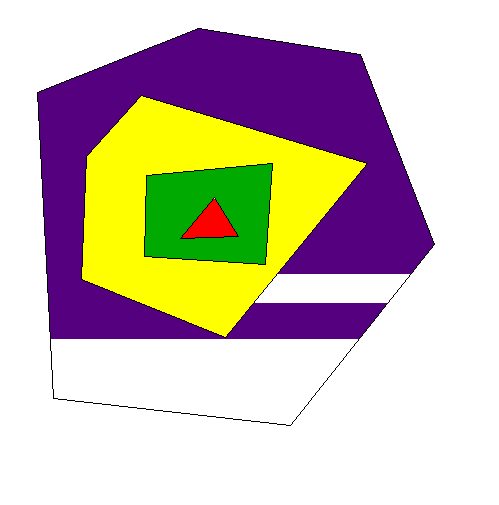
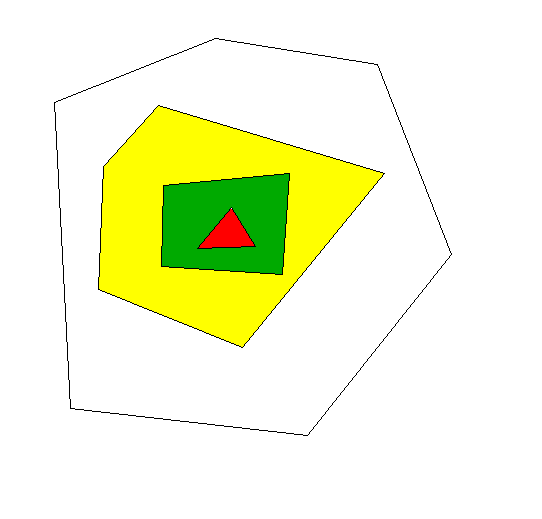
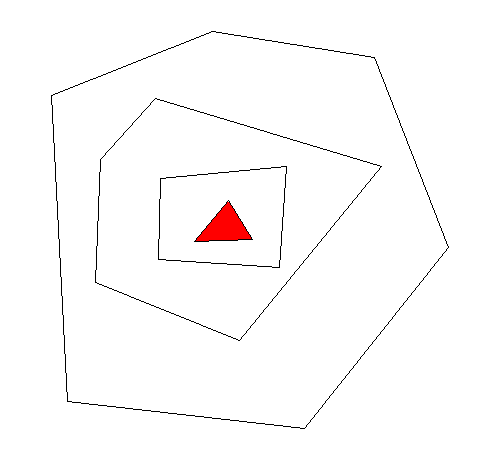
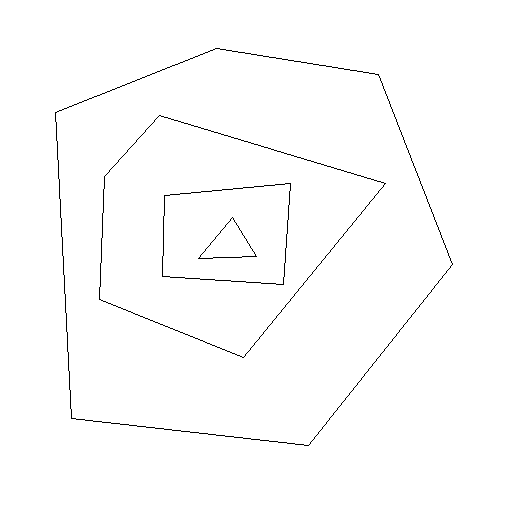
|  |  |
| --- | --- |
| **Тема: РЕАЛИЗАЦИЯ И ИССЛЕДОВАНИЕ АЛГОРИТМА ПОСТРОЧНОГО ЗАТРАВОЧНОГО ЗАПОЛНЕНИЯ СПЛОШНЫХ ОБЛАСТЕЙ**  **Студент: Нгуен Ань Тхы**  **Группа: ИУ7-46Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель: Куров.А.В** |  |

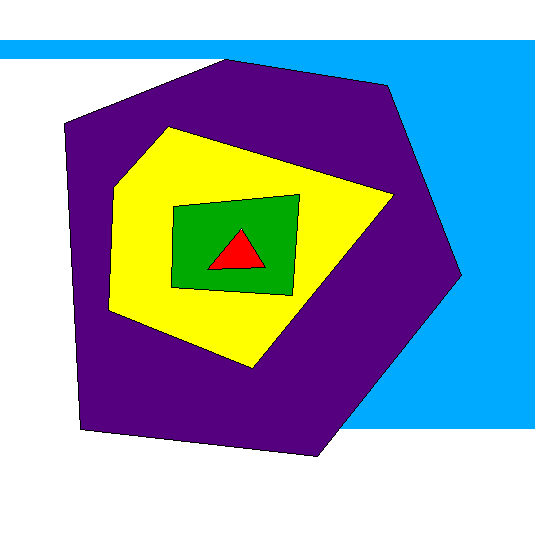
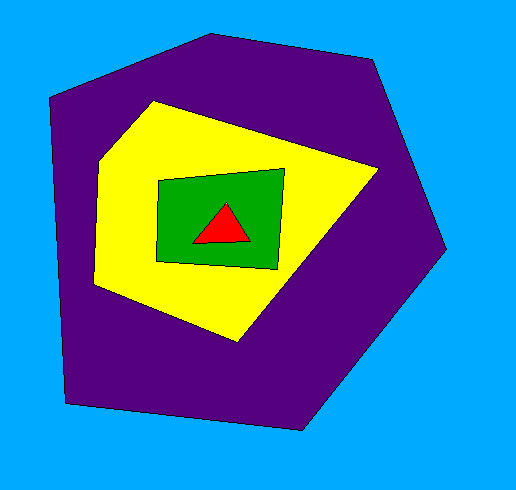
Москва.

2020 г.

**Цель работы:** реализация и исследование алгоритма построчного затравочного заполнения.

**I. Пример выполнения программы:**



**II. Алгоритм:**

1. Извлечение затравочного пиксела на интервале из стека, содержащего затравочные пикселы.

2. Присвоение извлеченному из стека пикселу цвета закраски:

пиксел (x,y) = цвет закраски.

3. Запоминание абсциссы текущего пиксела в промежуточной переменной wx: wx=x;

4. Заполнение интервала справа от затравочного пиксела:

1. x=x+1;
2. пока цвет пиксела (x,y)≠цвет границы выполниять:

пиксел (x,y)=цвет закраски;

x=x+1;

5. Запоминание в переменной xпр абсциссы крайнего правого пиксела

xпр =x -1.

1. Восстановление абсциссы затравочного пиксела x=wx.
2. Заполнение интервала слева от затравочного пиксела:
3. x=x-1;
4. пока цвет пиксела (x,y)≠цвет границы выполнить:

пиксел (x,y)=цвет закраски;

x=x-1.

8. Запоминание в переменной xлев абсциссы крайнего левого пиксела

xлев =x +1.

9. Восстановление абсциссы затравочного пиксела x=wx.

1. Проверка в интервале xлев ≤ x≤ xпр вышележащей строки с целью определения ее принадлежности границе многоугольника или ее полного заполнения и поиск затравки, начиная с левого края в случае ее незаполненности:

а) присвоение переменной f нулевого значения f=0, показывающего, что незакрашенный пиксел еще не найден;

б) выполнение следующих действий пока текущий пиксел не является граничным и цвет пиксела не совпадает с цветом закраски и пока не достигнута правая граница (пиксел(x,y) ≠цвет границы)∧(пиксел (x,y)≠цвет закраски)∧(x≤ xпр ):

в) присвоение переменной f единичного значения, показывающего наличие еще не закрашенного пиксела, f=1;

г) переход к следующему пикселу x=x+1;

д) помещение в стек крайнего правого незаполненного пиксела:

- если f=1, то незаполненные пикселы есть и в стек помещается затра-вочный пиксел: если абсцисса пиксела совпадает со значением xпр и пиксел не является граничным и его цвет не совпадает с цветом закраски ((пиксел(x,y) ≠цвет границы)∧( пиксел (x,y)≠цвет закраски)∧(x=xпр ), то в стек помещается пиксел с координатами (x,y), в противном случае в стек помещается пиксел с координатами (x-1,y);

е)присвоение переменной f нулевого значения f=0;

1. Поиск затравочного пиксела в случае прерывания интервала пикселов уже заполненными или граничными пикселами:

а) присвоение промежуточной переменной xn абсциссы текущего пиксела xn=x;

б) переход к соседнему пикселу, пока цвет текущего пиксела совпадает с цветом границы или цвет пиксела совпадает с цветом закраски и не достигнута правая граница ((пиксел(x,y)=цвет границы)∨(пиксел (x,y)=цвет закраски))∧(x<xпр ): x=x+1;

в) проверка абсциссы текущего пиксела: если x=xn, то увеличение абсциссы пиксела на единицу: x=x+1;

г)поиск затравочного пиксела для нового интервала, т.е. выполнение действий, начиная с п.10.

1. Проверка строки, лежащей ниже первоначальной, т.е. строки с ординатой y=y-2 (выполнение действий, начиная с п..9).

**III. Код программы:**

from PyQt5.QtWidgets import \*

from PyQt5.QtCore import \*

from PyQt5.QtGui import \*

from PyQt5 import uic

import sys

from algorithms import \*

# class Mainwindow

class MyWindow(QMainWindow):

def \_\_init\_\_(self):

QWidget.\_\_init\_\_(self)

uic.loadUi("design.ui", self)

self.bg\_color = QColor(Qt.white)

self.bd\_color = QColor(Qt.black)

self.fill\_color = QColor("#ffff00")

self.can\_w = 600

self.can\_h = 600

self.seed = None

self.choosing\_seed = False

self.ctrl\_pressed = False

self.first\_dot = None

self.cur\_figure = []

self.follow\_line = None

self.drawing\_circle = False

self.circle = None

self.follow\_circle = None

self.scene = QGraphicsScene(0, 0, self.can\_w, self.can\_h)

self.mainview.setScene(self.scene)

self.image = QImage(self.can\_w, self.can\_h, QImage.Format\_ARGB32\_Premultiplied)

self.pen = QPen()

self.pen.setColor(self.bd\_color)

self.pixmap = QPixmap(self.can\_w, self.can\_h)

#Input

self.inputs = [

self.pushButton\_bd\_clr,

self.pushButton\_fill\_clr,

self.pushButton\_seed,

self.pushButton\_add,

self.pushButton\_end,

self.pushButton\_clear,

self.pushButton\_fill,

self.lineEdit\_x,

self.lineEdit\_y,

self.lineEdit\_seed\_x,

self.lineEdit\_seed\_y,

self.checkBox

]

reg\_ex = QRegExp("[0-9]+")

int\_validator = QRegExpValidator(reg\_ex, self)

self.lineEdit\_x.setValidator(int\_validator)

self.lineEdit\_y.setValidator(int\_validator)

self.lineEdit\_seed\_x.setValidator(int\_validator)

self.lineEdit\_seed\_y.setValidator(int\_validator)

# Привязка кнопок

self.pushButton\_bd\_clr.clicked.connect(lambda: get\_color\_bd(self))

self.pushButton\_fill\_clr.clicked.connect(lambda: get\_color\_fill(self))

self.pushButton\_seed.clicked.connect(lambda: get\_seed(self))

self.pushButton\_add.clicked.connect(lambda: press\_add\_dot(self))

self.pushButton\_end.clicked.connect(lambda: end(self))

self.pushButton\_clear.clicked.connect(lambda: clear(self))

self.pushButton\_fill.clicked.connect(lambda: fill(self))

self.pushButton\_circle.clicked.connect(lambda: add\_circle(self))

self.mainview.setMouseTracking(True)

self.mainview.viewport().installEventFilter(self)

self.label\_bd.setStyleSheet("background-color: rgb(%d, %d, %d)" % self.bd\_color.getRgb()[:3])

self.label\_fill.setStyleSheet("background-color: rgb(%d, %d, %d)" % self.fill\_color.getRgb()[:3])

def eventFilter(self, source, event):

if event.type() == QEvent.MouseMove and source is self.mainview.viewport():

x = event.x()

y = event.y()

if len(self.cur\_figure) > 0:

prev = self.cur\_figure[-1]

if self.ctrl\_pressed:

if self.follow\_line:

self.scene.removeItem(self.follow\_line)

dx = x - prev[0]

dy = y - prev[1]

if abs(dy) >= abs(dx):

cur = (prev[0], y)

else:

cur = (x, prev[1])

self.follow\_line = self.scene.addLine(prev[0], prev[1], cur[0], cur[1], self.pen)

else:

if self.follow\_line:

self.scene.removeItem(self.follow\_line)

self.follow\_line = self.scene.addLine(prev[0], prev[1], x, y, self.pen)

if self.drawing\_circle:

if self.circle:

xc, yc = self.circle

if self.follow\_circle:

self.scene.removeItem(self.follow\_circle)

r = (abs(x - xc)\*\*2 + abs(y - yc)\*\*2)\*\*0.5

self.follow\_circle = self.scene.addEllipse(xc - r, yc - r, 2 \* r, 2 \* r, self.pen)

return QWidget.eventFilter(self, source, event)

def keyPressEvent(self, event):

key = event.key()

if key == Qt.Key\_Control:

self.ctrl\_pressed = True

def keyReleaseEvent(self, event):

key = event.key()

if key == Qt.Key\_Control:

self.ctrl\_pressed = False

def mousePressEvent(self, event):

but = event.button()

if but == 1 or but == 2:

x = event.x()

y = event.y()

borders = self.mainview.geometry().getCoords()

if borders[0] <= x < borders[2] and borders[1] <= y < borders[3]:

x -= borders[0]

y -= borders[1]

if but == 1:

# Затравка

if self.choosing\_seed:

self.lineEdit\_seed\_x.setText(str(x))

self.lineEdit\_seed\_y.setText(str(y))

QApplication.restoreOverrideCursor()

self.choosing\_seed = False

enable\_buttons(self.inputs)

# Окружность

elif self.drawing\_circle:

if not self.circle:

self.circle = (x, y)

else:

r = round((abs(self.circle[0] - x)\*\*2 + abs(self.circle[1] - y)\*\*2)\*\*0.5)

create\_circle(self, self.circle, r)

self.drawing\_circle = False

self.circle = None

add\_pixmap(self)

self.scene.removeItem(self.follow\_circle)

enable\_buttons(self.inputs)

# Обычный отрезок

elif self.ctrl\_pressed == 0 or len(self.cur\_figure) == 0:

add\_dot(self, x, y)

# Отрезок под прямым углом

else:

prev = self.cur\_figure[-1]

dx = x - prev[0]

dy = y - prev[1]

if abs(dy) >= abs(dx):

add\_dot(self, prev[0], y)

else:

add\_dot(self, x, prev[1])

elif but == 2:

end(self)

# pixmap на сцену (для закраски с задержкой)

def paintEvent(self, event):

self.scene.clear()

add\_pixmap(self)

# Fill figure

def fill(self):

try:

x = int(self.lineEdit\_seed\_x.text())

y = int(self.lineEdit\_seed\_y.text())

except ValueError:

mes("Неверные координаты затравки")

return -1

if not 0 < x < self.can\_w - 1 or not 0 < y < self.can\_h - 1:

mes("Неверные координаты затравки")

return -2

self.seed = (x, y)

put\_borders(self, self.bd\_color)

if self.checkBox.checkState() == 0:

fill\_default(self)

else:

fill\_delay(self)

put\_borders(self, self.bg\_color)

add\_pixmap(self)

def press\_add\_dot(self):

try:

x = int(self.lineEdit\_x.text())

y = int(self.lineEdit\_y.text())

except ValueError:

mes("Неверные координаты точки")

return -1

if x <= self.can\_w and y <= self.can\_h:

add\_dot(self, x, y)

def add\_dot(self, x, y):

if 0 <= x < self.can\_w and 0 <= y < self.can\_h:

self.cur\_figure.append((x, y))

if len(self.cur\_figure) > 1:

create\_line(self, self.cur\_figure[-2], self.cur\_figure[-1])

add\_pixmap(self)

self.scene.removeItem(self.follow\_line)

def end(self):

if len(self.cur\_figure) > 2:

create\_line(self, self.cur\_figure[-1], self.cur\_figure[0])

self.cur\_figure.clear()

add\_pixmap(self)

self.scene.removeItem(self.follow\_line)

def add\_circle(self):

self.drawing\_circle = True

disable\_buttons(self.inputs)

def add\_pixmap(self):

self.pixmap.convertFromImage(self.image)

self.scene.addPixmap(self.pixmap)

def get\_color\_bd(self):

color = QColorDialog.getColor()

if color.isValid():

self.label\_bd.setStyleSheet("background-color: rgb(%d, %d, %d)" % color.getRgb()[:3])

self.bd\_color = color

self.pen.setColor(self.bd\_color)

def get\_color\_fill(self):

color = QColorDialog.getColor()

if color.isValid():

self.label\_fill.setStyleSheet("background-color: rgb(%d, %d, %d)" % color.getRgb()[:3])

self.fill\_color = color

def get\_seed(self):

disable\_buttons(self.inputs)

QApplication.setOverrideCursor(Qt.CrossCursor)

self.choosing\_seed = True

def clear(self):

self.scene.clear()

self.image.fill(self.bg\_color)

def mes(text):

msg = QMessageBox()

msg.setIcon(QMessageBox.Warning)

msg.setWindowTitle("Внимание")

msg.setText(text)

msg.setStandardButtons(QMessageBox.Ok)

retval = msg.exec\_()

def disable\_buttons(buttons):

for b in buttons:

b.setEnabled(False)

def enable\_buttons(buttons):

for b in buttons:

b.setEnabled(True)

def without(array, element):

new\_array = array.copy()

new\_array.remove(element)

return new\_array

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

app = QApplication([])

application = MyWindow()

application.show()

sys.exit(app.exec())

from PyQt5.QtWidgets import QApplication

from math import sqrt

def put\_pix(self, x, y, color):

self.image.setPixel(x, y, color.rgb())

def get\_pix(self, x, y):

return self.image.pixelColor(x, y)

def create\_line(self, dot1, dot2):

x1, y1 = dot1

x2, y2 = dot2

dx = x2 - x1

dy = y2 - y1

if dx == 0 and dy == 0:

put\_pix(self, x1, y1, self.bd\_color)

return 1

sx = sign(dx)

sy = sign(dy)

dx = abs(dx)

dy = abs(dy)

if dx > dy:

change = 0

else:

change = 1

dx, dy = dy, dx

e = 2 \* dy - dx

new\_dx = 2 \* dx

new\_dy = dy \* 2

while x1 != x2 or y1 != y2:

put\_pix(self, x1, y1, self.bd\_color)

if e >= 0:

if change == 0:

y1 += sy

else:

x1 += sx

e -= new\_dx

if e < 0:

if change == 0:

x1 += sx

else:

y1 += sy

e += new\_dy

return 0

def sign(x):

if x > 0:

return 1

elif x < 0:

return -1

else:

return 0

# Рисует окружность каноническим методом

def create\_circle(self, centre, r):

xc, yc = centre

for x in range(0, r + 1, 1):

y = round(sqrt(r \*\* 2 - x \*\* 2))

put\_pix(self, xc + x, yc + y, self.bd\_color)

put\_pix(self, xc + x, yc - y, self.bd\_color)

put\_pix(self, xc - x, yc + y, self.bd\_color)

put\_pix(self, xc - x, yc - y, self.bd\_color)

for y in range(0, r + 1, 1):

x = round(sqrt(r \*\* 2 - y \*\* 2))

put\_pix(self, xc + x, yc + y, self.bd\_color)

put\_pix(self, xc + x, yc - y, self.bd\_color)

put\_pix(self, xc - x, yc + y, self.bd\_color)

put\_pix(self, xc - x, yc - y, self.bd\_color)

# Рисует границы вокруг полотна указанным цветом

def put\_borders(self, color):

for i in range(self.can\_w):

put\_pix(self, i, 0, color)

put\_pix(self, i, self.can\_h - 1, color)

for i in range(self.can\_h):

put\_pix(self, 0, i, color)

put\_pix(self, self.can\_w - 1, i, color)

# Построчная закраска с затравкой

def fill\_default(self):

stack = []

stack.append(self.seed)

while len(stack) > 0:

x, y = stack.pop()

x\_temp = x

put\_pix(self, x, y, self.fill\_color)

x += 1

while get\_pix(self, x, y) != self.bd\_color:

put\_pix(self, x, y, self.fill\_color)

x += 1

x\_right = x - 1

x = x\_temp

x -= 1

while get\_pix(self, x, y) != self.bd\_color:

put\_pix(self, x, y, self.fill\_color)

x -= 1

x\_left = x + 1

for i in range(1, -2, -2):

x = x\_left

y += i

while x <= x\_right:

flag = False

while get\_pix(self, x, y) != self.bd\_color and get\_pix(self, x, y) != self.fill\_color and x < x\_right:

if not flag:

flag = True

x += 1

if flag:

if get\_pix(self, x, y) != self.bd\_color and get\_pix(self, x, y) != self.fill\_color and x == x\_right:

stack.append((x, y))

else:

stack.append((x - 1, y))

x\_enter = x

while (get\_pix(self, x, y) == self.bd\_color or get\_pix(self, x, y) == self.fill\_color) and x < x\_right:

x += 1

if x == x\_enter:

x += 1

y -= i

# Построчная закраска с затравкой с задержкой

def fill\_delay(self):

stack = []

stack.append(self.seed)

while len(stack) > 0:

QApplication.processEvents()

x, y = stack.pop()

x\_temp = x

put\_pix(self, x, y, self.fill\_color)

x += 1

while get\_pix(self, x, y) != self.bd\_color:

put\_pix(self, x, y, self.fill\_color)

x += 1

x\_right = x - 1

x = x\_temp

x -= 1

while get\_pix(self, x, y) != self.bd\_color:

put\_pix(self, x, y, self.fill\_color)

x -= 1

x\_left = x + 1

self.repaint()

for i in range(1, -2, -2):

x = x\_left

y += i

while x <= x\_right:

flag = False

while get\_pix(self, x, y) != self.bd\_color and get\_pix(self, x, y) != self.fill\_color and x < x\_right:

if not flag:

flag = True

x += 1

if flag:

if get\_pix(self, x, y) != self.bd\_color and get\_pix(self, x, y) != self.fill\_color and x == x\_right:

stack.append((x, y))

else:

stack.append((x - 1, y))

flag = False

x\_enter = x

while (get\_pix(self, x, y) == self.bd\_color or get\_pix(self, x, y) == self.fill\_color) and x < x\_right:

x += 1

if x == x\_enter:

x += 1

y -= i