**Main.py**

if \_\_name\_\_ == '\_\_main\_\_':  
 from PyQt5 import QtWidgets  
 import sys  
 app = QtWidgets.QApplication(sys.argv) # Конструтирование QApp  
 from UI.window\_switcher import WindowSwitcher  
 window\_switcher = WindowSwitcher()  
 window\_switcher.showFullScreen() # Создание и показ основного окна на полный экран  
 sys.exit(app.exec\_()) # Запуск QApp

**Global.py**

from PyQt5.QtWidgets import QWidget  
from PyQt5.QtCore import pyqtSignal  
  
# Класс переменной с сигналом. Базируется на QWidget, поскольку по-другому QT сигналы работают через раз.  
# Нужен [сигнал] для плавных двусторонних бесшовных переходов между экранными формами  
class VariableWithSignal(QWidget):  
 valueChanged = pyqtSignal(object)  
 def \_\_init\_\_(self):  
 super(VariableWithSignal, self).\_\_init\_\_()  
 self.text = ''  
  
 def setText(self, str):  
 self.text = str  
 # Объявление своего метода "геттера" свойства text  
 @property  
 def text(self):  
 return self.\_text  
 # Объявление своего метода "сеттера" свойства text (нужно для отслеживания и активации сигнала)  
 @text.setter  
 def text(self, value):  
 self.\_text = value  
 self.valueChanged.emit(value)  
  
  
DIFFICULTY = VariableWithSignal()  
DIFFICULTY.text = 'EASY'  
CURRENT\_LEVEL\_NUMBER = VariableWithSignal()  
CURRENT\_LEVEL\_NUMBER.text = 1  
CURRENT\_WINDOW = VariableWithSignal()  
CURRENT\_WINDOW.text = 'MainMenu'  
# Гениальный костыль для интерфейса во многом. Словарь подошел как нельзя кстати  
difficulty\_translator = {'EASY': 'ПРОСТАЯ', 'MEDIUM': 'СРЕДНЯЯ', 'HARD': 'ВЫСОКАЯ'}

**Level\_sample.py**

# Шаблон уровня  
class Level:  
 # Объявление некоторых переменных, отвечающих за прохождение и наполнение каждого из урвоней  
 def \_\_init\_\_(self, vertices\_positions, edges\_list):  
 self.vertices\_positions = vertices\_positions  
 self.edges\_list = edges\_list  
 self.edges\_positions\_list = calculate\_edges\_positions\_list(self.vertices\_positions, self.edges\_list)  
 self.bypassed\_edges = []  
 self.current\_painted\_vertex = -1  
 self.previous\_painted\_vertex = -1  
  
 # Проверка того, что заданное ребро имеется в списке ребер  
 def edge\_exists(self, edge):  
 return edge in self.edges\_list  
  
 # Проверка существования вершины графа (разница между заданной координатой и пройденной пользователем может быть  
 # чуть больше нуля, поскольку вершины графа изображаются кругом с некоторым радиусом (a/2)  
 # В случае нахождения такой вершины возвращает ее номер в списке вершин, иначе -1  
 def vertex\_exists(self, vertex\_x, vertex\_y):  
 for i in range(len(self.vertices\_positions)):  
 vertex = self.vertices\_positions[i]  
 if abs(vertex[0] - vertex\_x) <= 15 and abs(vertex[1] - vertex\_y) <= 15:  
 return i  
 return -1  
  
# эта функция пишет координаты  
# (координаты начальной и конечной точки)  
def calculate\_edges\_positions\_list(vertices\_positions, edges\_list):  
 edges\_positions\_list = []  
 for edge in edges\_list:  
 edges\_positions\_list.append(vertices\_positions[edge[0]] + vertices\_positions[edge[1]])  
 return edges\_positions\_list

**Levels.py**

from core.level\_sample import Level as Level  
  
# Описание всех уроовней в игре  
# Level ([(vertex\_x, vertex\_y)], [(vertex\_number, vertex\_number)])  
EASY\_1 = Level([(411, 162), (411, 454), (565, 454), (565, 250), (699, 294), (699, 103), (565, 162)],  
 [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [0, 6], [3, 6]])  
EASY\_2 = Level([(287, 234), (542, 67), (795, 234), (721, 475), (369, 475)], [[0, 3], [0, 2], [1, 3], [1, 4], [2, 4]])  
EASY\_3 = Level([(426, 78), (426, 332), (426, 526), (716, 78), (716, 332)], [[0, 1], [0, 3], [1, 2], [1, 3], [1, 4],  
 [2, 4], [3, 4]])  
EASY\_4 = Level([(332, 42), (332, 478), (766, 42), (766, 478), (554, 260)], [[0, 1], [0, 4], [1, 3], [1, 4], [2, 3],  
 [2, 4], [3, 4]])  
EASY\_5 = Level([(391, 155), (542, 155), (693, 155), (468, 53), (618, 53),  
 (391, 395), (542, 395), (693, 395), (468, 478), (618, 478)  
 ], [[0, 1], [0, 3], [1, 2], [1, 3], [1, 4], [2, 4],  
 [0, 5], [2, 7],  
 [5, 8], [6, 8], [6, 9], [7, 9]])

**back\_menu.py**

class BackMenu(QGridLayout):  
 # full отвечает за наличие/отсутствие кнопки возвращения к списку уровней  
 def \_\_init\_\_(self, full: bool = False):  
 super(BackMenu, self).\_\_init\_\_()  
 # Объявление кнопки  
 self.back\_to\_main\_menu\_button = QToolButton()  
 self.back\_to\_main\_menu\_button.setFixedSize(261, 111)  
 self.back\_to\_main\_menu\_button.setText('В\nГЛАВНОЕ\nМЕНЮ')  
 # Установка стиля кнопки (css)  
 self.back\_to\_main\_menu\_button.setStyleSheet('''background: rgba(0, 0, 0, 0.88);  
 border-radius: 50px;  
 font-size: 29px;  
 color: #FFFFFF;''')  
 # Установка функции-ответчика на нажатие на кнопку  
 self.back\_to\_main\_menu\_button.clicked.connect(lambda: return\_to('MainMenu'))  
 self.back\_to\_level\_list = QToolButton()  
 self.back\_to\_level\_list.setFixedSize(261, 111)  
 self.back\_to\_level\_list.setText('НАЗАД К\nСПИСКУ\nУРОВНЕЙ')  
 self.back\_to\_level\_list.clicked.connect(lambda: return\_to('LevelSwitcher'))  
 self.back\_to\_level\_list.setStyleSheet('''background: rgba(0, 0, 0, 0.88);  
 border-radius: 50px;  
 font-size: 29px;  
 color: #FFFFFF;''')  
 # Добавление виджетов в список виджетов. Последние две цифры означают координатное положение в сетке  
 # (строка, столбец)  
 if full:  
 self.addWidget(self.back\_to\_level\_list, 0, 2)  
 self.addWidget(self.back\_to\_main\_menu\_button, 0, 3)  
 else:  
 self.addWidget(self.back\_to\_main\_menu\_button, 0, 3)  
  
# Простая функция перехода к другой экранной форме (указывается в аргументе, позже анализируется в WindowSwitcher)  
# Как было описано ранее, аргумент этой функции передается через qtSignal в WindowSwitcher  
def return\_to(widget):  
 core.GLOBAL.CURRENT\_WINDOW.text = widget

**Ellipse.py**

from PyQt5.QtWidgets import QLabel  
  
# В игре много эллипсов. Решил запилить класс, отвечающий за создание такой красоты  
# Название аргументов буквально отражает их суть  
class Ellipse(QLabel):  
 def \_\_init\_\_(self, color, radius, border\_radius=50):  
 super(Ellipse, self).\_\_init\_\_()  
 self.setStyleSheet(f'''background: rgba{color};  
 border-radius: {border\_radius}%;   
 ''')  
 self.color = color  
 self.radius = radius  
 self.setFixedSize(radius, radius)  
  
 def correctStyleSheet(self, newStyleSheet):  
 self.setStyleSheet(newStyleSheet)

**Level\_switcer.py**

# Создание класса формы с кнопками уровней  
class LevelSwitcher(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 self.central\_widget = QWidget()  
 self.setCentralWidget(self.central\_widget)  
  
 # Объявление главного шаблона формы и добавление в него индикатора сложности (добавляется извне)  
 self.main\_layout = QVBoxLayout()  
 self.difficulty\_indicator = difficulty\_indicator.DifficultyIndicator()  
 self.difficulty\_indicator.main\_widget.setParent(self.central\_widget)  
 self.difficulty\_indicator.main\_widget.move(14, 46)  
  
 # Создание и добавление таблички УРОВНИ  
 self.level\_label = QLabel(self.central\_widget)  
 self.level\_label.move(890 - self.level\_label.width(), 249)  
 self.level\_label.setText(' УРОВНИ ')  
 self.level\_label.setFixedSize(354, 104)  
 self.level\_label.setStyleSheet('''background: rgba(3, 3, 3, 0.85);  
 border-radius: 50px;  
 font-weight: 400;  
 color:white;  
 font-size: 64px;''')  
  
 # Создание формы с кнопками  
 self.buttons\_layout = QHBoxLayout()  
 self.buttons\_layout.setContentsMargins(150, 0, 150, 0)  
 # Последовательная генерация кнопок  
 for i in range(1, 6):  
 self.level\_button = QToolButton()  
 self.level\_button.setText(str(i))  
 self.level\_button.setFixedSize(137, 131)  
 # Добавление только что созданной кнопке функции по нажатию. (sender запоминает и выдает указатель на конкретный объект)  
 self.level\_button.clicked.connect(lambda: setAndStartLevel(self.sender().text()))  
 self.level\_button.setStyleSheet('''  
 background: rgba(6, 6, 6, 0.66);  
 border-radius: 25px;  
 color: white;  
 font-size: 64px;  
 ''')  
 self.buttons\_layout.addWidget(self.level\_button)  
  
 # Добавление меню возвращения  
 self.back\_menu = BackMenu()  
 self.back\_menu.setContentsMargins(1000, 0, 0, 0)  
 self.main\_layout.setContentsMargins(0, 400, 0, 0)  
 self.main\_layout.addLayout(self.buttons\_layout)  
 self.main\_layout.addLayout(self.back\_menu)  
  
 self.central\_widget.setLayout(self.main\_layout)  
  
  
def setAndStartLevel(level\_number):  
 core.GLOBAL.CURRENT\_LEVEL\_NUMBER.text = level\_number  
 core.GLOBAL.CURRENT\_WINDOW.text = 'LevelWindow'

**Difficulty\_indicator.py**

from PyQt5.QtWidgets import QLabel, QWidget, QHBoxLayout  
from UI.common.ellipse import Ellipse  
import core.GLOBAL  
  
# Класс индикатора сложности. Встраивается в LevelSwitcher  
class DifficultyIndicator:  
 def \_\_init\_\_(self):  
 # Объявление главного (центрального виджета)  
 self.main\_widget = QWidget()  
 self.main\_widget.setFixedSize(504, 163)  
 self.main\_widget.setStyleSheet('background: rgba(1, 1, 1, 0.06);\  
 border-radius: 60px;')  
 # Объявление элементов индикатора  
 self.elements\_layout = QHBoxLayout()  
 self.ellipse = QLabel()  
 self.ellipse.setFixedSize(84, 84)  
 self.ellipse.setStyleSheet('background: rgba(1, 1, 1, 0.94);\  
 border-radius: 42px;')  
 self.rectangle\_with\_difficulty = QLabel()  
 self.rectangle\_with\_difficulty\  
 .setText(f'СЛОЖНОСТЬ\n {core.GLOBAL.difficulty\_translator[core.GLOBAL.DIFFICULTY.text]}')  
 core.GLOBAL.DIFFICULTY.valueChanged.connect(lambda: self.rectangle\_with\_difficulty.setText(  
 f'СЛОЖНОСТЬ\n {core.GLOBAL.difficulty\_translator[core.GLOBAL.DIFFICULTY.text]}'))  
 self.rectangle\_with\_difficulty.setStyleSheet("background: rgba(0, 0, 0, 0.84);\  
 font-size:36px;\  
 color:white;\  
 padding: 5px;\  
 border-radius: 30px;")  
  
 self.rectangle\_with\_difficulty.setFixedSize(309, 111)  
 self.elements\_layout.addWidget(self.ellipse)  
 self.elements\_layout.addWidget(self.rectangle\_with\_difficulty)  
 self.main\_widget.setLayout(self.elements\_layout)  
 # Создание некоторых беллых эллипсов возле текста индикатора  
 self.white\_ellipse\_1 = Ellipse((255, 255, 255, 1), 27)  
 self.white\_ellipse\_1.correctStyleSheet(f'''background: rgba{self.white\_ellipse\_1.color};  
 border-radius: 12px; ''')  
 self.white\_ellipse\_1.setParent(self.main\_widget)  
 self.white\_ellipse\_2 = Ellipse((255, 255, 255, 1), 27)  
 self.white\_ellipse\_2.correctStyleSheet(f'''background: rgba{self.white\_ellipse\_1.color};  
 border-radius: 12px; ''')  
 self.white\_ellipse\_2.setParent(self.main\_widget)  
 self.white\_ellipse\_1.move(193, 105)  
 self.white\_ellipse\_2.move(434, 45)

**Level\_window.py**

from PyQt5.QtWidgets import QMainWindow, QVBoxLayout, QWidget, QToolButton  
from PyQt5.QtGui import QPainter, QColor, QPen, QIcon  
from PyQt5.QtCore import Qt, QSize  
import core.GLOBAL  
from UI.LevelWindow.elements.level\_painter import LevelPainter  
from UI.LevelWindow.elements.level\_indicator import LevelIndicator  
from UI.common.back\_menu import BackMenu  
  
# Класс Окна с уровнем  
class LevelWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super(LevelWindow, self).\_\_init\_\_()  
 self.central\_widget = QWidget()  
 self.setCentralWidget(self.central\_widget)  
 self.mouse\_pos\_start = 0  
 self.mouse\_pos\_end\_list = []  
 self.clicked = False # Некоторые переменные для выполнения игровых действий  
 self.level\_over = False  
  
 self.central\_vertical\_layout = QVBoxLayout(self.central\_widget)  
 self.central\_vertical\_layout.setContentsMargins(100, 0, 100, 100)  
 self.level\_indicator = LevelIndicator()  
  
 self.back\_menu = BackMenu(True)  
  
 # Генерация содержимого окна  
 self.level\_content = LevelPainter(core.GLOBAL.CURRENT\_LEVEL\_NUMBER.text)  
 core.GLOBAL.CURRENT\_LEVEL\_NUMBER.valueChanged.connect(self.retake\_level)  
 self.central\_vertical\_layout.addWidget(self.level\_indicator.main\_widget)  
 self.central\_vertical\_layout.addWidget(self.level\_content)  
 self.central\_vertical\_layout.addLayout(self.back\_menu)  
  
 # Создание кнопки перехода на следующий уровень  
 self.next\_button = QToolButton(self)  
 self.next\_button.setIcon(QIcon('UI/MainMenu/elements/arrow.png'))  
 self.next\_button.setIconSize(QSize(95, 95))  
 self.next\_button.setFixedSize(95, 95)  
 self.next\_button.setStyleSheet('''color:white;  
 border-radius:47%;  
 background:#000000;  
 font-size:96px;''')  
 self.next\_button.clicked.connect(self.play\_next\_level)  
 self.next\_button.move(1774, 439)  
  
 # Функция перерисовки и смены уровня + генерация содержимого окна  
 def retake\_level(self):  
 self.central\_vertical\_layout.removeWidget(self.level\_indicator.main\_widget)  
 self.central\_vertical\_layout.removeWidget(self.level\_content)  
 self.central\_vertical\_layout.removeItem(self.back\_menu)  
 self.level\_indicator = LevelIndicator()  
 self.level\_content = LevelPainter(core.GLOBAL.CURRENT\_LEVEL\_NUMBER.text)  
 self.central\_vertical\_layout.addWidget(self.level\_indicator.main\_widget)  
 self.central\_vertical\_layout.addWidget(self.level\_content)  
 self.central\_vertical\_layout.addLayout(self.back\_menu)  
 self.restart\_level()  
 self.update()  
 def play\_next\_level(self):  
 core.GLOBAL.CURRENT\_LEVEL\_NUMBER.text = \  
 int(core.GLOBAL.CURRENT\_LEVEL\_NUMBER.text) + 1 if int(core.GLOBAL.CURRENT\_LEVEL\_NUMBER.text) + 1 < 6 else 1  
 self.retake\_level()  
 self.update()  
  
 # Отслеживание того что мышь зажата  
 def mousePressEvent(self, event):  
 super(LevelWindow, self).mousePressEvent(event)  
 if event.button() == Qt.MouseButton.LeftButton and not self.level\_over:  
 self.clicked = True  
 self.mouse\_pos\_start = event.pos()  
 self.mouse\_pos\_end\_list.append(event.pos())  
 self.update()  
  
 # Отслеживание того что ЛКМ отпущена  
 def mouseReleaseEvent(self, event):  
 if event.button() == Qt.MouseButton.LeftButton:  
 self.clicked = False  
 if not self.level\_over: self.restart\_level()  
 self.update()  
 super(LevelWindow, self).mouseReleaseEvent(event)  
  
 # Отслеживание того что мышь передвигается  
 def mouseMoveEvent(self, event):  
 super(LevelWindow, self).mouseMoveEvent(event)  
 if self.clicked:  
 self.mouse\_pos\_end\_list.append(event.pos())  
 self.do\_core\_operations()  
 if not self.level\_over:  
 self.update()  
 else:  
 self.update()  
 self.mouse\_pos\_end\_list.clear()  
  
 # Функция отрисовки перемещения мыши  
 def paintEvent(self, paint\_event):  
 super(LevelWindow, self).paintEvent(paint\_event)  
 if self.clicked:  
 content\_painter = QPainter(self.level\_content)  
 # Antialiasing on  
 content\_painter.setRenderHint(QPainter.RenderHint.Antialiasing)  
 color = QColor(255, 0, 0)  
 pen = QPen(Qt.PenStyle.SolidLine)  
 pen.setColor(color)  
 content\_painter.setPen(pen)  
 if not self.level\_over:  
 content\_painter.drawLine(self.mouse\_pos\_start, self.mouse\_pos\_end\_list[0])  
 for i in range(1, len(self.mouse\_pos\_end\_list) - 1):  
 content\_painter.drawLine(self.mouse\_pos\_end\_list[i], self.mouse\_pos\_end\_list[i + 1])  
 content\_painter.end()  
  
 # Функция нахождения пройденных вершин и ребер графа + передача оных в Level и LevelPainter  
 def do\_core\_operations(self):  
 # Try to find vertex in level  
 level = self.level\_content.level  
 index = self.level\_content.level.vertex\_exists(  
 self.mouse\_pos\_end\_list[len(self.mouse\_pos\_end\_list) - 1].x() - 300 - 66 \* 3 + 100,  
 self.mouse\_pos\_end\_list[len(self.mouse\_pos\_end\_list) - 1].y() - 216)  
 if index != -1:  
 level.previous\_painted\_vertex = level.current\_painted\_vertex  
 if index != level.current\_painted\_vertex:  
 level.current\_painted\_vertex = index  
 new\_edge = sorted((level.previous\_painted\_vertex, level.current\_painted\_vertex))  
 if level.edge\_exists(new\_edge):  
 new\_edge = self.level\_content.level.vertices\_positions[new\_edge[0]] \  
 + self.level\_content.level.vertices\_positions[new\_edge[1]]  
 if not (new\_edge in level.bypassed\_edges):  
 level.bypassed\_edges.append(new\_edge)  
 else:  
 self.restart\_level()  
  
 # Условие окончания уровня  
 if len(level.bypassed\_edges) == len(level.edges\_list):  
 self.level\_over = True  
  
 # Функция обнуления уровня  
 def restart\_level(self):  
 self.level\_over = False  
 self.clicked = False  
 level = self.level\_content.level  
 self.mouse\_pos\_start = 0  
 self.mouse\_pos\_end\_list.clear()  
 level.previous\_painted\_vertex = -1  
 level.current\_painted\_vertex = -1  
 level.bypassed\_edges.clear()

**Level\_indicator.py**

from PyQt5.QtWidgets import QLabel, QWidget, QHBoxLayout  
import core.GLOBAL  
  
# Индикатор уровня. Описание и применение аналогично индикатору сложности. За исключением лишь того, что номер уровня  
# берется из ядра  
class LevelIndicator:  
 def \_\_init\_\_(self):  
 self.main\_widget = QWidget()  
 self.main\_widget.setFixedSize(504, 163)  
 self.main\_widget.setStyleSheet('background: rgba(6, 6, 6, 0.08);\  
 border-radius: 60px;')  
 self.elements\_layout = QHBoxLayout()  
 self.ellipse = QLabel()  
 self.ellipse.setFixedSize(111, 111)  
 self.ellipse.setStyleSheet('background:rgba(6, 6, 6, 0.76);\  
 border-radius: 55%;')  
 self.rectangle\_with\_level\_number = QLabel()  
 self.rectangle\_with\_level\_number.setText(f'УРОВЕНЬ {core.GLOBAL.CURRENT\_LEVEL\_NUMBER.text}')  
 self.rectangle\_with\_level\_number.setStyleSheet("background: rgba(6, 6, 6, 0.76);\  
 font-size:36px;\  
 color:white;\  
 border-radius: 50px;")  
  
 self.rectangle\_with\_level\_number.setFixedSize(309, 111)  
 self.elements\_layout.addWidget(self.ellipse)  
 self.elements\_layout.addWidget(self.rectangle\_with\_level\_number)  
 self.main\_widget.setLayout(self.elements\_layout)

**Level\_painter.py**

from PyQt5.QtWidgets import QMainWindow, QLabel  
from PyQt5.QtGui import QPainter, QColor, QPen  
from PyQt5.QtCore import QPoint, Qt  
  
from core.levels import \*  
import core.GLOBAL  
  
# Класс отрсиовщика уровня. Используется для первоначальной (фоновой) отросвки графа-уровня  
class LevelPainter(QMainWindow):  
 def \_\_init\_\_(self, number):  
 super().\_\_init\_\_()  
 self.setContentsMargins(66, 0, 66, 0)  
 self.level\_number = number  
 self.dot\_radius = 11 # радиус отрисовываемых точек  
 self.line\_width = 1 # ширина длинны  
 self.horizontal\_margin = 300  
 self.vertical\_margin = 56  
 self.level = None # Placeholder  
 self.level\_content\_holder = QLabel()  
 self.level\_content\_holder.setStyleSheet('background: rgba(0, 0, 0, 0.05);border-radius: 92px;\  
 OneMorePoint{background:rgba(6, 6, 6, 0.76);\  
 border-radius: 55%};')  
 self.setCentralWidget(self.level\_content\_holder)  
 self.get\_and\_set\_level()  
  
 # Вызов операции получения уровня  
 def get\_and\_set\_level(self):  
 self.level = self.get\_level\_content()  
  
 # Запуск отрисовки фона (графа). Выполняет каждый раз при вызове self.update()  
 def paintEvent(self, event):  
 content\_painter = QPainter(self) # Создание экземпляра "отрисовщика"  
 # Antialiasing on  
 content\_painter.setRenderHint(QPainter.RenderHint.Antialiasing)  
 # Установка цвета для отрисовки линий и вершин  
 color = QColor(3, 3, 3)  
 color.setAlpha(200)  
 pen = QPen(Qt.PenStyle.SolidLine)  
 pen.setColor(color)  
 content\_painter.setPen(pen)  
 content\_painter.setBrush(color)  
 vertices\_list = self.level.vertices\_positions  
 edges\_list = self.level.edges\_positions\_list  
 bypassed\_edges\_list = self.level.bypassed\_edges  
 for vertex in vertices\_list:  
 content\_painter.drawEllipse(QPoint(vertex[0] + self.horizontal\_margin, vertex[1] + self.vertical\_margin),  
 self.dot\_radius, self.dot\_radius)  
 pen.setWidth(self.line\_width)  
 content\_painter.setPen(pen)  
 for edge in edges\_list:  
 content\_painter.drawLine(edge[0] + self.horizontal\_margin, edge[1] + self.vertical\_margin, edge[2]  
 + self.horizontal\_margin, edge[3] + self.vertical\_margin)  
 # Установка красного цвета для отрисовки пройденных ребер графа  
 color.setRgb(255, 0, 0)  
 pen.setColor(color)  
 content\_painter.setPen(pen)  
 for edge in bypassed\_edges\_list:  
 content\_painter.drawLine(edge[0] + self.horizontal\_margin, edge[1] + self.vertical\_margin, edge[2]  
 + self.horizontal\_margin, edge[3] + self.vertical\_margin)  
  
 content\_painter.end() # Обязательное заверещение цикла отрисовки  
  
 def get\_level\_content(self):  
 needed\_level\_name = f'{core.GLOBAL.DIFFICULTY.text}\_{self.level\_number}'  
 match needed\_level\_name:  
 case 'EASY\_1':  
 needed\_level = EASY\_1  
 case 'EASY\_2':  
 needed\_level = EASY\_2  
 case 'EASY\_3':  
 needed\_level = EASY\_3  
 case 'EASY\_4':  
 needed\_level = EASY\_4  
 case 'EASY\_5':  
 needed\_level = EASY\_5  
 return needed\_level

**Buttons\_layout.py**

from PyQt5.QtWidgets import QGridLayout, QToolButton, QVBoxLayout  
from PyQt5.QtGui import QIcon  
from PyQt5.QtCore import QSize, Qt  
import core.GLOBAL  
from sys import exit  
  
# Файл с содержанием кнопок главного меню. Название отражают содержание  
  
main\_layout = QGridLayout()  
play\_button = QToolButton()  
play\_button.setFixedSize(267, 240)  
play\_button.setStyleSheet('''  
 background: rgba(3, 3, 3, 0.89);  
 border: 1px solid #060606;  
 color:white;  
 font-size: 30px;  
 border-radius: 50px;''')  
play\_button.setIcon(QIcon('UI/MainMenu/elements/play\_button.png'))  
play\_button.setText('ИГРАТЬ')  
play\_button.setToolButtonStyle(Qt.ToolButtonStyle.ToolButtonTextUnderIcon)  
play\_button.setIconSize(QSize(111, 111))  
play\_button.clicked.connect(lambda : send\_change\_signal("LevelSwitcher"))  
  
about\_game\_button = QToolButton()  
about\_game\_button.clicked.connect(lambda : send\_change\_signal('RulesWindow'))  
about\_game\_button.setFixedSize(193, 180)  
about\_game\_button.setText('ОБ ИГРЕ')  
about\_game\_button.setToolButtonStyle(Qt.ToolButtonStyle.ToolButtonTextUnderIcon)  
about\_game\_button.setIcon(QIcon('UI/MainMenu/elements/settings\_button.png'))  
about\_game\_button.setIconSize(QSize(129, 112)) #размер шестеренки  
about\_game\_button.setStyleSheet('''  
 background: #010101;  
 font-size:25px;  
 color: white;  
 border-radius: 50px;  
 ''') #значения текста об игре  
  
next\_button = QToolButton()  
next\_button.clicked.connect(exit)  
next\_button.setIcon(QIcon('UI/MainMenu/elements/exit.png'))  
next\_button.setIconSize(QSize(95, 95))  
next\_button.setFixedSize(95, 95)  
next\_button.setStyleSheet('''color:white;  
 border-radius:47%;  
 background:#000000;  
 font-size:96px;''')  
main\_layout.addWidget(play\_button, 1, 1)  
main\_layout.addWidget(about\_game\_button, 2, 2)  
main\_layout.addWidget(next\_button, 3, 3)  
  
def send\_change\_signal(widget):  
 core.GLOBAL.CURRENT\_WINDOW.setText(widget)

**Main\_menu.py**

from PyQt5.QtWidgets import QMainWindow, QVBoxLayout, QWidget, QLabel  
from PyQt5.QtCore import Qt  
from UI.MainMenu.elements import buttons\_layout  
from PyQt5.QtGui import QPixmap  
  
# Класс главного меню  
class MainMenu:  
 def \_\_init\_\_(self):  
 self.main\_window = QMainWindow()  
 self.main\_window.setWindowTitle('Обведи не отрывая перо')  
 self.central\_widget = QWidget()  
  
 # Объявление подвиджета для правильной отрисовки кнопок над фоном  
 self.sub\_widget = QWidget(parent=self.central\_widget)  
 self.sub\_layout = QVBoxLayout()  
 self.main\_layout = QVBoxLayout()  
  
 self.sub\_layout.addLayout(self.main\_layout)  
 self.feather\_logo = QLabel(parent=self.central\_widget)  
 self.app\_name = QLabel(parent=self.central\_widget)  
 self.main\_layout.addLayout(buttons\_layout.main\_layout)  
  
 self.sub\_widget.setLayout(self.sub\_layout)  
 self.sub\_widget.setFixedSize(957, 593)  
 self.sub\_widget.setStyleSheet('''  
 background: rgba(46, 38, 38, 0.03);  
 border-radius: 68px;  
 ''')  
  
 self.sub\_widget.move(426, 263)  
 self.construct\_logo\_and\_app\_name()  
 self.main\_window.setCentralWidget(self.central\_widget)  
  
 # Создание лого в главном меню  
 def construct\_logo\_and\_app\_name(self):  
 self.feather\_logo.move(26, 31) #расположение пера  
 self.feather\_logo.setPixmap(QPixmap('UI/MainMenu/elements/feather\_logo\_t.png'))  
 self.app\_name.setFixedSize(345, 158) #расположение текста  
 self.app\_name.setPixmap(QPixmap('UI/MainMenu/elements/app\_namee.png'))  
 self.app\_name.setAlignment(Qt.AlignmentFlag.AlignCenter)  
 self.app\_name.move(317, 78)

**Rules\_window.py**

from PyQt5.QtWidgets import QWidget, QLabel, QToolButton, QMainWindow  
from PyQt5.QtGui import QPixmap  
import core.GLOBAL  
from UI.common.ellipse import Ellipse  
  
# Класс окна с правилами  
class RulesWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super(RulesWindow, self).\_\_init\_\_()  
 self.central\_widget = QWidget()  
 self.sub\_widget = QWidget()  
  
 # Создание таблички ПРАВИЛА ИГРЫ  
 self.rules\_label = QLabel(parent=self.central\_widget)  
 self.rules\_label.setFixedSize(550, 123)  
 self.rules\_label.setText(' ПРАВИЛА ИГРЫ')  
 self.rules\_label.setStyleSheet('''background: rgba(0, 0, 0, 0.83);  
 color:white;  
 font-size: 50px;  
 border-radius: 50px;''')  
 self.rules\_label.move(139, 48)  
  
 # Создание таблички с правилами непосредственно  
 self.rules\_itself = QLabel(parent=self.central\_widget)  
 self.rules\_itself.setText(  
 ''' Правила игры\n заключаются в том, что\n игрок должен обвести\n сложную замкнутую\n фигуру, не отрывая пера\n и не проходить по одним и\n тем же линиям дважды.\n Если же пользователь\n оторвет перо или дважды\n пройдет по той же линии,\n то игра начнется сначала.\n ''')  
 self.rules\_itself.setStyleSheet('''  
 background: rgba(1, 1, 1, 0.69);  
 color:white;  
 font-weight: 400;  
 font-size: 50px;  
 border-radius: 50px;''')  
 self.rules\_itself.setFixedSize(815, 744)  
 self.rules\_itself.move(346, 209)  
  
 # Объявление кнопки возвращения в главное меню  
 self.back\_to\_main\_menu\_button = QToolButton(parent=self.central\_widget)  
 self.back\_to\_main\_menu\_button.setFixedSize(220, 88)  
 self.back\_to\_main\_menu\_button.setText('В ГЛАВНОЕ\nМЕНЮ')  
 self.back\_to\_main\_menu\_button.setStyleSheet('''background: rgba(0, 0, 0, 0.88);  
 border-radius: 37px;  
 font-size: 29px;  
 color: #FFFFFF;''')  
 self.back\_to\_main\_menu\_button.clicked.connect(lambda: return\_to('MainMenu'))  
 self.back\_to\_main\_menu\_button.move(self.rules\_itself.x() + self.rules\_itself.width() + 100, 865)  
 self.setCentralWidget(self.central\_widget)  
  
 # Создание эллипсов  
 self.white\_ellipse\_1 = Ellipse((196, 196, 196, 1), 32)  
 self.white\_ellipse\_1.correctStyleSheet(f'''background: rgba{self.white\_ellipse\_1.color};  
 border-radius: 16px; ''')  
 self.white\_ellipse\_1.setParent(self.central\_widget)  
 self.white\_ellipse\_2 = Ellipse((196, 196, 196, 1), 75)  
 self.white\_ellipse\_2.correctStyleSheet(f'''background: rgba{self.white\_ellipse\_1.color};  
 border-radius: 37px; ''')  
 self.white\_ellipse\_2.setParent(self.central\_widget)  
 self.white\_ellipse\_1.move(629, 119)  
 self.white\_ellipse\_2.move(1057, 229)  
  
  
def return\_to(widget):  
 core.GLOBAL.CURRENT\_WINDOW.text = widget

**Window\_switcher.py**

from PyQt5.QtWidgets import QStackedWidget  
import core.GLOBAL  
from UI.MainMenu.main\_menu import MainMenu  
from UI.LevelWindow.level\_window import LevelWindow  
from UI.LevelSwitcher.level\_switcher import LevelSwitcher  
from UI.RulesWindow.rules\_window import RulesWindow  
  
# Главный класс Ui, отвечающий за плавные переходы между экранными формами  
class WindowSwitcher(QStackedWidget):  
 # Инициализация основных экранных форм  
 def \_\_init\_\_(self):  
 super(WindowSwitcher, self).\_\_init\_\_()  
 self.main\_menu = MainMenu().main\_window  
 self.level\_switcher = LevelSwitcher()  
 self.level\_window = None  
 self.rules\_window = RulesWindow()  
  
 self.addWidget(self.main\_menu)  
 self.addWidget(self.level\_switcher)  
 self.addWidget(self.rules\_window)  
 core.GLOBAL.CURRENT\_WINDOW.valueChanged.connect(self.switch\_window\_to)  
  
 # Функция смены текущего виджета  
 def switch\_window\_to(self, widget):  
 if self.level\_window is not None: self.removeWidget(self.level\_window) # Проверка на наличие окна уровня (требуется для корректной перезаписи)  
 if widget == 'MainMenu':  
 self.setCurrentIndex(0)  
 elif widget == 'LevelSwitcher':  
 self.setCurrentIndex(1)  
 elif widget == 'RulesWindow':  
 self.setCurrentIndex(2)  
 elif widget == 'LevelWindow':  
 self.level\_window = LevelWindow()  
 self.addWidget(self.level\_window)  
 self.setCurrentIndex(3)  
 self.update()