T.C
KOCAELİ SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR/YAZILIM MÜHENDİSLİĞİ

PROJE KONUSU:

SAYISAL TASARIM PROJESİ

ÖĞRENCİ ADI:
NAZLI SU KETÇİ
AMİNE DERİN

ÖĞRENCİ NUMARASI:
220501007
220501004
Github:
https://github.com/nzlktc02
https://github.com/aminederin

TARİH: 05.05.2024

1.GİRİŞ

Projenin Amacı:

Bu kod, bir mantık devresi tasarlayıcı uygulamasını oluşturur. Kullanıcı, grafik arayüzü üzerinden mantık kapıları, giriş/çıkış elemanları ve LED'ler gibi öğeleri ekleyebilir, bunları düzenleyebilir ve birbirleri arasında bağlantılar kurabilir. Ayrıca, eklenen öğelerin özelliklerini düzenlemek için bağlam menüsünü kullanabilir.

Ana işlevler şunlardır:

1. **Mantık Kapıları Ekleme**: Kullanıcı, AND, OR, NOT, NAND, NOR, XOR ve XNOR gibi farklı mantık kapıları ekleyebilir. Bu kapılar farklı şekillerde çizilir ve üzerlerine etiketler eklenir.
2. **Giriş/Çıkış Elemanları Ekleme**: Kullanıcı, devreye giriş veya çıkış elemanları ekleyebilir. Giriş elemanları yeşil, çıkış elemanları ise kırmızı renkte gösterilir.
3. **LED Ekleme**: Kullanıcı, devreye bir LED ekleyebilir. LED'in rengi değiştirilebilir.
4. **Bağlantı Kurma**: Kullanıcı, öğeler arasında bağlantılar kurabilir. Bu bağlantılar, bir öğeden diğerine doğru çizgiler şeklinde gösterilir ve üzerlerine etiketler eklenir.
5. **Öğe Özelliklerini Düzenleme**: Kullanıcı, eklenen öğelerin özelliklerini düzenleyebilir. Etiketler, renkler ve başlangıç değerleri gibi özellikler değiştirilebilir.
6. **Simülasyon Başlatma, Durdurma ve Sıfırlama**: Simülasyon butonları aracılığıyla kullanıcı, tasarladığı mantık devresini çalıştırabilir, durdurabilir ve sıfırlayabilir.

Kodun, PyQt5 kütüphanesi kullanılarak PyQt5 ile geliştirildiğini görüyoruz. PyQt5, Python diline bir Qt arabirimi sağlar ve bu kod, PyQt5'ın sağladığı çeşitli araçlar ve işlevler kullanılarak bir grafik arayüz tasarlar. Bu kod, kullanıcı dostu bir arayüz ile mantık devrelerinin hızlıca tasarlanmasını ve simülasyonunun yapılmasını sağlar.

2.GEREKSİNİM ANALİZİ:

1. Arayüz Gereksinimleri:

Kullanıcı Arayüzü Gereksinimleri:

1. **Mantık Kapıları Ekleme**: Kullanıcı, arayüze sağlanan araçlar veya menüler aracılığıyla farklı mantık kapıları ekleyebilmelidir. Bu kapılar arayüzde görüntülenmeli ve kullanıcının istediği yere yerleştirilebilmelidir.
2. **Giriş/Çıkış Elemanları Ekleme**: Kullanıcı, arayüze giriş veya çıkış elemanları ekleyebilmelidir. Bu elemanlar, farklı renklerle gösterilmeli ve kullanıcı tarafından arayüze yerleştirilebilmelidir.
3. **LED Ekleme**: Kullanıcı, devreye bir LED ekleyebilmelidir. LED'in rengi, kullanıcının isteğine göre değiştirilebilir olmalıdır.
4. **Bağlantı Kurma**: Kullanıcı, mantık kapıları, giriş/çıkış elemanları ve LED'ler arasında bağlantılar kurabilmelidir. Bağlantılar, kullanıcı tarafından fareyle çizilerek belirlenir ve doğru bir şekilde görselleştirilir.
5. **Öğe Düzenleme**: Kullanıcı, eklenen öğelerin özelliklerini düzenleyebilmelidir. Bu özellikler arasında etiketlerin düzenlenmesi, renk seçimi ve başlangıç değerlerinin ayarlanması bulunur.
6. **Simülasyon Kontrolleri**: Kullanıcı, tasarlanan devrenin simülasyonunu başlatabilmeli, duraklatabilmeli ve sıfırlayabilmelidir. Bu kontroller, arayüzdeki ilgili düğmeler veya menüler aracılığıyla sağlanmalıdır.

Donanım Arayüzü Gereksinimleri:

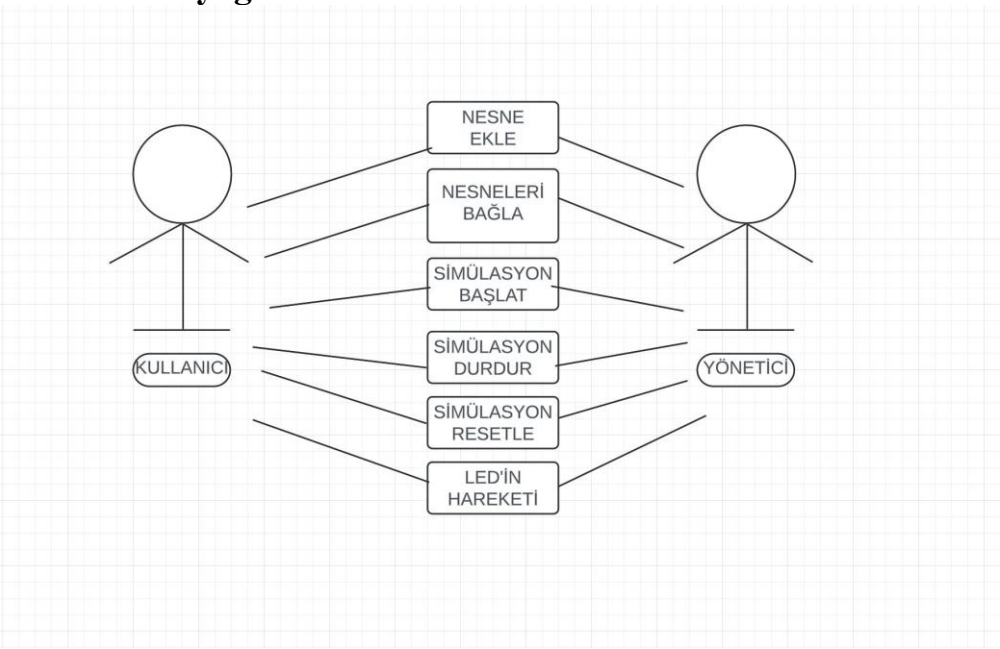
1. Bu uygulama için donanım arayüzü gereksinimi yoktur, çünkü uygulama tamamen yazılım tabanlıdır ve kullanıcılar bilgisayarlarından erişirler.

2. Fonksiyonel Gereksinimler:

1. **Mantık Kapıları Fonksiyonları**: Eklenen her mantık kapısı, kendi işlevselliğine sahip olmalıdır. Örneğin, AND kapısı iki girişi olan bir mantık kapısıdır ve girişlerin durumuna göre çıkışı belirler.
2. **Bağlantı Fonksiyonları**: Kurulan her bağlantı, bağlı öğeler arasında doğru bir şekilde veri iletişimini sağlamalıdır. Bağlantılar, kullanıcının tasarladığı devrenin doğru bir şekilde çalışmasını sağlamak için önemlidir.
3. **Öğe Düzenleme Fonksiyonları**: Kullanıcılar, ekledikleri öğelerin özelliklerini doğru bir şekilde düzenleyebilmelidir. Örneğin, bir mantık kapısının etiketi değiştirilebilmeli veya bir LED'in rengi ayarlanabilmelidir.

4. **Simülasyon Kontrolleri Fonksiyonları**: Kullanıcılar, simülasyonu başlatma, duraklatma ve sıfırlama işlevlerini doğru bir şekilde kullanabilmelidir. Simülasyon kontrolleri, tasarlanan devrenin istenen şekilde çalışmasını sağlamak için önemlidir.

Use- Case diyagramı:



3.TASARIM

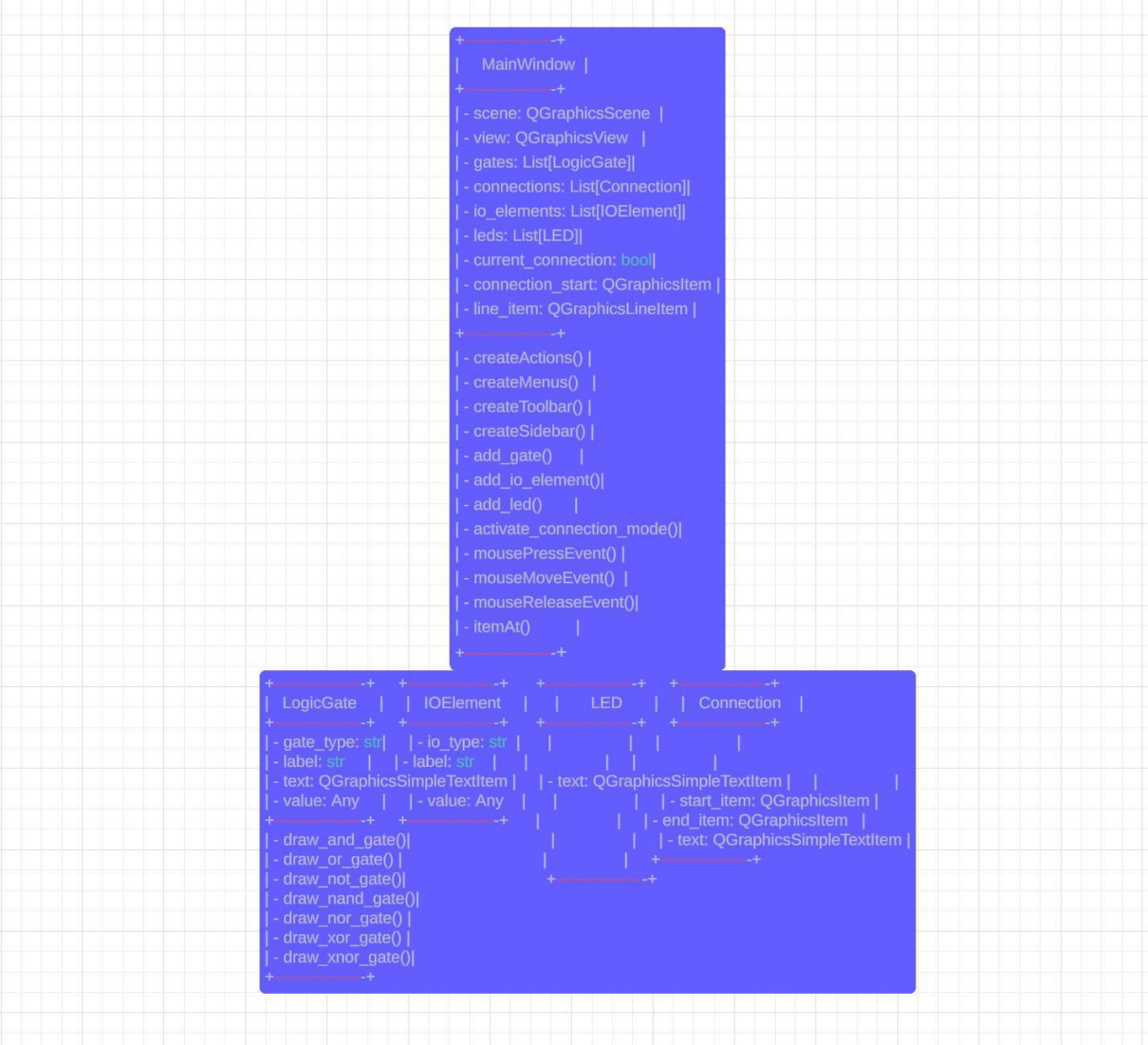
Mimari Tasarım:

Mimari Tasarımın Açıklanması:

1. **Modüler Yaklaşım**: Uygulama, modüler bir yaklaşımla tasarlanmıştır. Her bir bileşen (mantık kapıları, giriş/çıkış elemanları, LED'ler, bağlantılar) kendi sınıfında ve dosyasında ayrıca ele alınmıştır.
2. **Model-Görünüm-İşlem (MVC) Tasarım Deseni**: PyQt5 kullanılarak model-görünüm-işlem (MVC) tasarım deseni takip edilmiştir. Model, QGraphicsScene üzerindeki grafik nesnelerini temsil eder. Görünüm, QGraphicsView aracılığıyla bu nesnelerin kullanıcıya gösterilmesini sağlar. İşlem, kullanıcı etkileşimlerini (örneğin, yeni öğe ekleme, bağlantı oluşturma) kontrol eder.

3. **Olay Tabanlı Programlama**: PyQt5, olay tabanlı bir programlama yaklaşımını benimser. Kullanıcı etkileşimleri, fare veya klavye olayları gibi olaylara dayalı olarak işlenir ve bu olaylar uygun işlem metodlarına yönlendirilir.

MODÜL DİYAGRAMI



Kullanılacak Teknolojiler:

Yazılım Dilinin Belirlenmesi:

- **Python**: Yazılım, PyQt5 kütüphanesi ile birlikte Python programlama dilinde yazılacaktır. Python, hızlı prototipleme imkanı sağlar ve PyQt5 gibi kütüphanelerle etkileşimi kolaydır.

Kullanılacak Harici Kütüphaneler:

- **PyQt5**: PyQt5, grafik arayüz oluşturmak için kullanılacak temel kütüphanedir. Qt kütüphanesinin Python bağlayıcısıdır ve zengin bir grafik arayüz deneyimi sağlar.

Diğer Teknolojiler:

- **Qt Graphics Framework**: PyQt5 ile birlikte kullanılacak olan Qt Graphics Framework, grafik nesneleri (örneğin, şekiller, çizgiler, metinler) üzerinde işlemler yapmak için kullanılacaktır. Bu framework, grafik arayüz öğelerini oluşturmak için kullanışlı bir araçtır.

Kullanıcı Arayüzü Tasarımı

Kullanıcı Arayüzü Tasarımı Hakkında Açıklama:

Kullanıcı arayüzü, mantık devresi tasarlama ve simülasyon yapma işlevlerini kullanıcı dostu bir şekilde sunmak için tasarlanmıştır. PyQt5 kullanılarak geliştirilen bu arayüz, kullanıcıların devre bileşenlerini sürükleyip bırakmasını, öğelerin özelliklerini düzenlemesini ve devreleri simüle etmesini sağlar.

Ekran Çıktıları ve Açıklamalar:

1. **Ana Pencere**: Uygulamanın ana penceresi, mantık kapıları, giriş/çıkış elemanları ve bağlantıların yerleştirileceği geniş bir çalışma alanına sahiptir.
    - **Araç Çubuğu**: Mantık kapıları ve giriş/çıkış elemanlarını eklemek için düğmeler içerir.
    - **Yan Panel**: Oto Çiz seçeneği gibi kontrol elemanlarını içerir.
    - **Simülasyon Menüsü**: Simülasyonu başlatma, duraklatma ve sıfırlama kontrollerini içerir.
2. **Mantık Kapısı Ekleme**: Kullanıcı, araç çubuğundan bir mantık kapısı ekleme seçeneğini seçtiğinde, sahneye bir mantık kapısı eklenir ve kullanıcı tarafından istenilen yere taşınabilir.
    - **Etiket Düzenleme**: Sağ tıklama menüsünden mantık kapısının etiketini düzenleyebilir.

- **Değer Ayarlama**: Sağ tıklama menüsünden mantık kapısının giriş değerlerini ayarlayabilir.

3. **Bağlantı Kurma**: Kullanıcı, bağlantı ekleme moduna geçtiğinde, mantık kapıları, giriş/çıkış elemanları ve LED'ler arasında bağlantılar kurabilir.

UYGULAMA

Kodlanan bileşenlerin açıklamaları

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QGraphicsScene,
QGraphicsView, QAction, QMenu, QGraphicsItem, \
    QGraphicsEllipseItem, QGraphicsRectItem, QGraphicsLineItem,
QGraphicsSimpleTextItem, QColorDialog, QInputDialog, \
    QMessageBox, QVBoxLayout, QHBoxLayout, QWidget, QPushButton,
QRadioButton, QLineEdit, QButtonGroup, QLabel, QCheckBox, QDockWidget
from PyQt5.QtGui import QPainter, QBrush, QPen, QColor, QPainterPath,
QTransform
from PyQt5.QtCore import Qt, QLineF, QPointF
```

Bu kısımda, PyQt5'te kullanmak üzere çeşitli modüller ve sınıflar içe aktarılıyor. Bunlar, GUI oluşturmak, grafik sahneleri ve öğeleri yönetmek için gerekli sınıfları içerir.

```
class LogicGate(QGraphicsRectItem):
    def __init__(self, gate_type, label):
        super().__init__(0, 0, 80, 40)
        self.gate_type = gate_type
        self.label = label
        self.setBrush(QBrush(Qt.white))
        self.setPen(QPen(Qt.black, 2))
        self.setFlag(QGraphicsItem.ItemIsMovable)
        self.setFlag(QGraphicsItem.ItemIsSelectable)
        self.setAcceptHoverEvents(True)
        self.text = QGraphicsSimpleTextItem(label, self)
        self.text.setPos(20, 10)
        self.value = None
```

LogicGate sınıfı, mantık kapıları (AND, OR, NOT, vb.) için bir grafik öğesidir. QGraphicsRectItem sınıfından türetilmiştir. Mantık kapısının tipi, etiketi ve grafikleri ayarlanır. Bu sınıf, kapının etiketini ve değerini düzenleyebilecek bir bağlam menüsü de içerir.

```
def contextMenuEvent(self, event):
    menu = QMenu()
    label_action = menu.addAction("Etiketi Ayarla")
```

```
    value_action = menu.addAction("Değeri Ayarla")

    action = menu.exec_(event.screenPos())

    if action == label_action:
        text, ok = QInputDialog.getText(None, "Etiketi Ayarla", "Yeni
etiketi girin:")
        if ok:
            self.text.setText(text)
            self.label = text
    elif action == value_action:
        value, ok = QInputDialog.getInt(None, "Değeri Ayarla", "Değer (0
veya 1):", 0, 0, 1)
        if ok:
            self.value = value
```

Bu metod, sağ tıklama ile açılan bağlam menüsünü (context menu) yönetir. Kullanıcı etiket veya değer ayarlama seçeneklerini seçebilir. Bu seçenekler kullanıcının kapı etiketini ve giriş değerini değiştirmesine olanak tanır.

```
def paint(self, painter, option, widget=None):
    super().paint(painter, option, widget)
    if self.gate_type == "AND":
        self.draw_and_gate(painter)
    elif self.gate_type == "OR":
        self.draw_or_gate(painter)
    elif self.gate_type == "NOT":
        self.draw_not_gate(painter)
    elif self.gate_type == "NAND":
        self.draw_nand_gate(painter)
    elif self.gate_type == "NOR":
        self.draw_nor_gate(painter)
    elif self.gate_type == "XOR":
        self.draw_xor_gate(painter)
    elif self.gate_type == "XNOR":
        self.draw_xnor_gate(painter)

def draw_and_gate(self, painter):
    painter.setPen(Qt.black)
    painter.setBrush(Qt.NoBrush)
    painter.drawRect(0, 0, 80, 40)
    painter.drawArc(0, 0, 80, 40, 90 * 16, 180 * 16)

def draw_or_gate(self, painter):
    painter.setPen(Qt.black)
    painter.setBrush(Qt.NoBrush)
    path = QPainterPath()
    path.moveTo(0, 0)
    path.cubicTo(40, 20, 40, 20, 0, 40)
    path.moveTo(80, 0)
    path.cubicTo(40, 20, 40, 20, 80, 40)
    painter.drawPath(path)

def draw_not_gate(self, painter):
    painter.setPen(Qt.black)
    painter.setBrush(Qt.NoBrush)
    painter.drawPolygon([QPointF(0, 0), QPointF(80, 20), QPointF(0, 40)])
    painter.drawEllipse(QPointF(85, 20), 5, 5)
```

```
def draw_nand_gate(self, painter):
    self.draw_and_gate(painter)
    painter.drawEllipse(QPointF(85, 20), 5, 5)

def draw_nor_gate(self, painter):
    self.draw_or_gate(painter)
    painter.drawEllipse(QPointF(85, 20), 5, 5)

def draw_xor_gate(self, painter):
    painter.setPen(Qt.black)
    painter.setBrush(Qt.NoBrush)
    path = QPainterPath()
    path.moveTo(5, 0)
    path.cubicTo(45, 20, 45, 20, 5, 40)
    path.moveTo(0, 0)
    path.cubicTo(40, 20, 40, 20, 0, 40)
    path.moveTo(80, 0)
    path.cubicTo(40, 20, 40, 20, 80, 40)
    painter.drawPath(path)

def draw_xnor_gate(self, painter):
    self.draw_xor_gate(painter)
    painter.drawEllipse(QPointF(85, 20), 5, 5)
```

Bu metod, mantık kapısını ekranda çizmek için QPainter kullanır. Kapının türüne göre ilgili çizim metodunu çağırır. Bu metodlar, belirli mantık kapılarının çizim işlemlerini gerçekleştirir (AND, OR, NOT, NAND, NOR, XOR, XNOR).

```
class IOElement(QGraphicsEllipseItem):
    def __init__(self, io_type, label):
        super().__init__(0, 0, 50, 50)
        self.io_type = io_type
        self.label = label
        self.setBrush(QBrush(Qt.green if io_type == 'Giriş' else Qt.red))
        self.setPen(QPen(Qt.black, 2))
        self.setFlag(QGraphicsItem.ItemIsMovable)
        self.setFlag(QGraphicsItem.ItemIsSelectable)
        self.setAcceptHoverEvents(True)
        self.text = QGraphicsSimpleTextItem(label, self)
        self.text.setPos(15, 15)
        self.value = 0 if io_type == 'Giriş' else None
```

Bu metod, sağ tıklama ile açılan bağlam menüsünü yönetir. Kullanıcı etiket, renk veya (giriş elemanları için) başlangıç değerini ayarlama seçeneklerini seçebilir.

```
class LED(QGraphicsEllipseItem):
    def __init__(self):
        super().__init__(0, 0, 20, 20)
        self.setBrush(QBrush(Qt.yellow))
        self.setFlag(QGraphicsItem.ItemIsMovable)
        self.setFlag(QGraphicsItem.ItemIsSelectable)
        self.setAcceptHoverEvents(True)
        self.text = QGraphicsSimpleTextItem("LED", self )
        self.text.setPos(25, 25)

    def contextMenuEvent(self, event):
        menu = QMenu()
```

```
        color_action = menu.addAction("Rengi Ayarla")

        action = menu.exec_(event.screenPos())

        if action == color_action:
            color = QColorDialog.getColor()
            if color.isValid():
                self.setBrush(QBrush(color))
```

LED sınıfı, LED elemanını temsil eder. QGraphicsEllipseItem sınıfından türetilmiştir. Kullanıcı LED'in rengini değiştirebilir. Connection sınıfı, iki eleman arasındaki bağlantıyı temsil eder. QGraphicsLineItem sınıfından türetilmiştir. Bağlantının pozisyonunu günceller ve bağlantı etiketini ortalar.

```
class Connection(QGraphicsLineItem):
    def __init__(self, start_item, end_item, label):
        super().__init__()
        self.start_item = start_item
        self.end_item = end_item
        self.label = label
        self.setPen(QPen(Qt.black, 2))
        self.setFlag(QGraphicsItem.ItemIsSelectable)
        self.update_position()
        self.text = QGraphicsSimpleTextItem(label, self)
        self.text.setPos((self.start_item.sceneBoundingRect().center() +
self.end_item.sceneBoundingRect().center()) / 2)

    def update_position(self):
        line = QLineF(self.start_item.sceneBoundingRect().center(),
self.end_item.sceneBoundingRect().center())
        self.setLine(line)

    def contextMenuEvent(self, event):
        menu = QMenu()
        label_action = menu.addAction("Etiketi Ayarla")
        color_action = menu.addAction("Rengi Ayarla")

        action = menu.exec_(event.screenPos())

        if action == label_action:
            text, ok = QInputDialog.getText(None, "Etiketi Ayarla", "Yeni
etiketi girin:")
            if ok:
                self.text.setText(text)
                self.label = text
        elif action == color_action:
            color = QColorDialog.getColor()
            if color.isValid():
                self.setPen(QPen(color, 2))
```

Connection sınıfı, iki eleman arasındaki bağlantıyı temsil eder. QGraphicsLineItem sınıfından türetilmiştir. Bağlantının pozisyonunu günceller ve bağlantı etiketini ortalar. Bu metod, sağ tıklama ile açılan bağlam menüsünü yönetir. Kullanıcı etiket veya rengi ayarlayabilir.

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
```

```
        self.setWindowTitle("Mantık Devresi Tasarlayıcı")
        self.setGeometry(100, 100, 1200, 800)

        self.scene = QGraphicsScene()
        self.view = QGraphicsView(self.scene, self)
        self.setCentralWidget(self.view)
        self.createActions()
        self.createMenus()
        self.createToolbar()
        self.createSidebar()

        self.gates = []
        self.connections = []
        self.io_elements = []
        self.leds = []
        self.current_connection = False
        self.connection_start = None
        self.line_item = None

    def createActions(self):
        self.add_and_gate_action = QAction("AND Kapısı Ekle", self)
        self.add_and_gate_action.triggered.connect(lambda:
self.add_gate("AND"))

        self.add_or_gate_action = QAction("OR Kapısı Ekle", self)
        self.add_or_gate_action.triggered.connect(lambda:
self.add_gate("OR"))

        self.add_not_gate_action = QAction("NOT Kapısı Ekle", self)
        self.add_not_gate_action.triggered.connect(lambda:
self.add_gate("NOT"))

        self.add_nand_gate_action = QAction("NAND Kapısı Ekle", self)
        self.add_nand_gate_action.triggered.connect(lambda:
self.add_gate("NAND"))

        self.add_nor_gate_action = QAction("NOR Kapısı Ekle", self)
        self.add_nor_gate_action.triggered.connect(lambda:
self.add_gate("NOR"))

        self.add_xor_gate_action = QAction("XOR Kapısı Ekle", self)
        self.add_xor_gate_action.triggered.connect(lambda:
self.add_gate("XOR"))

        self.add_xnor_gate_action = QAction("XNOR Kapısı Ekle", self)
        self.add_xnor_gate_action.triggered.connect(lambda:
self.add_gate("XNOR"))

        self.add_input_action = QAction("Giriş Ekle", self)
        self.add_input_action.triggered.connect(lambda:
self.add_io_element("Giriş"))

        self.add_output_action = QAction("Çıkış Ekle", self)
        self.add_output_action.triggered.connect(lambda:
self.add_io_element("Çıkış"))

        self.add_led_action = QAction("LED Ekle", self)
        self.add_led_action.triggered.connect(self.add_led)

        self.add_connection_action = QAction("Bağlantı Ekle", self)
```

```
self.add_connection_action.triggered.connect(self.activate_connection_mode)

        # Yeni butonlar eklendi
        self.start_simulation_action = QAction("Başlat", self)
        self.stop_simulation_action = QAction("Durdur", self)
        self.reset_simulation_action = QAction("Sıfırla", self)

    def createMenus(self):
        menubar = self.menuBar()

        file_menu = menubar.addMenu("Dosya")
        edit_menu = menubar.addMenu("Düzenle")
        view_menu = menubar.addMenu("Görünüm")
        simulation_menu = menubar.addMenu("Simülasyon")

        simulation_menu.addAction(self.add_led_action)
        simulation_menu.addAction(self.add_connection_action)

        # Yeni butonlar eklendi
        simulation_menu.addAction(self.start_simulation_action)
        simulation_menu.addAction(self.stop_simulation_action)
        simulation_menu.addAction(self.reset_simulation_action)

    def createToolbar(self):
        toolbar = self.addToolBar("Araçlar")
        toolbar.addAction(self.add_and_gate_action)
        toolbar.addAction(self.add_or_gate_action)
        toolbar.addAction(self.add_not_gate_action)
        toolbar.addAction(self.add_nand_gate_action)
        toolbar.addAction(self.add_nor_gate_action)
        toolbar.addAction(self.add_xor_gate_action)
        toolbar.addAction(self.add_xnor_gate_action)
        toolbar.addAction(self.add_input_action)
        toolbar.addAction(self.add_output_action)

    def createSidebar(self):
        sidebar = QWidget()
        sidebar_layout = QVBoxLayout()

        self.auto_draw_checkbox = QCheckBox("Oto Çiz")

        sidebar_layout.addWidget(self.auto_draw_checkbox)

        sidebar.setLayout(sidebar_layout)

        dock_widget = QDockWidget("Kontroller", self)
        dock_widget.setWidget(sidebar)
        self.addDockWidget(Qt.LeftDockWidgetArea, dock_widget)

    def add_gate(self, gate_type):
        gate = LogicGate(gate_type, gate_type)
        self.scene.addItem(gate)
        self.gates.append(gate)

    def add_io_element(self, io_type):
        io_element = IOElement(io_type, io_type)
        self.scene.addItem(io_element)
        self.io_elements.append(io_element)

    def add_led(self):
        led = LED()
```

```
        self.scene.addItem(led)
        self.leds.append(led)

    def activate_connection_mode(self):
        self.current_connection = True
        self.connection_start = None
        self.line_item = None

    def mousePressEvent(self, event):
        item = self.itemAt(event.pos())
        if isinstance(item, (LogicGate, IOElement, LED)):
            if event.button() == Qt.LeftButton:
                if self.current_connection:
                    if self.connection_start:
                        end_item = item
                        self.current_connection = False

self.line_item.setLine(QLineF(self.line_item.line().p1(),
end_item.sceneBoundingRect().center()))
                        self.scene.removeItem(self.line_item)
                        connection = Connection(self.connection_start,
end_item, f"Bağlantı {len(self.connections) + 1}")
                        self.scene.addItem(connection)
                        self.connections.append(connection)
                    else:
                        self.connection_start = item
                        self.line_item =
QGraphicsLineItem(QLineF(item.sceneBoundingRect().center(), event.pos()))
                        self.scene.addItem(self.line_item)
            elif event.button() == Qt.RightButton:
                item.contextMenuEvent(event)

    def mouseMoveEvent(self, event):
        if self.current_connection and self.line_item:
            self.line_item.setLine(QLineF(self.line_item.line().p1(),
self.mapToScene(event.pos())))

    def mouseReleaseEvent(self, event):
        if self.current_connection and isinstance(self.itemAt(event.pos()),
QGraphicsScene):
            self.scene.removeItem(self.line_item)
            self.current_connection = False
            self.connection_start = None
            self.line_item = None

    def itemAt(self, pos):
        return self.scene.itemAt(self.view.mapToScene(pos), QTransform())

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```

Bu metod, sağ tıklama ile açılan bağlam menüsünü yönetir. Kullanıcı etiket veya rengi ayarlayabilir. MainWindow sınıfı, ana pencereyi temsil eder. QMainWindow sınıfından türetilmiştir. Pencere başlığı ve boyutları ayarlanır.

Grafik sahnesi ve görüntüleyici oluşturulur. Menü, araç çubuğu ve yan panel (sidebar) oluşturulur. Bu metod, ana pencerede kullanılacak tüm eylemleri (actions) oluşturur ve bunları ilgili işlevlere bağlar. Her bir eylem bir menü veya araç çubuğu öğesi olarak kullanılabilir. Bu metod, ana pencerede kullanılacak menüleri oluşturur ve ilgili eylemleri bu menülere ekler. Bu metod, ana pencerede kullanılacak araç çubuğunu (toolbar) oluşturur ve ilgili eylemleri bu araç çubuğuna ekler. Bu metod, yan paneli (sidebar) oluşturur ve bu panelde otomatik çizim için bir onay kutusu (checkbox) ekler. Bu metodlar, sahneye yeni mantık kapıları, giriş/çıkış elemanları ve LED'ler ekler. Bu metodlar, fare olaylarını (mouse events) yönetir. Kullanıcı fareyi kullanarak bağlantılar oluşturabilir. mousePressEvent metodu, fareye basıldığında gerçekleşen işlemleri yönetir. Bu kısım, uygulamanın başlatılmasını sağlar. QApplication nesnesi oluşturulur, `MainWindow` nesnesi yaratılır ve gösterilir. sys.exit(app.exec_()) ile uygulama ana döngüye girer ve kapatıldığında programın düzgün bir şekilde sonlanmasını sağlar.

Bu şekilde, verilen PyQt5 uygulamasının yapısı ve işlevleri hakkında daha detaylı bir anlayış elde edilir. Her bölümün açıklaması, kodun okunabilirliğini ve anlaşılabilirliğini artırır. Bu yardımcı metodlar, ana pencere oluşturulurken kullanılan eylemleri, menüleri, araç çubuğunu ve yan paneli oluşturur. Her biri belirli bir bileşenin veya özelliğin oluşturulmasından sorumludur. Bu metodlar, sahneye yeni mantık kapıları, giriş/çıkış elemanları ve LED'ler eklemek için kullanılır. Her biri ilgili sınıfın örneğini oluşturur ve sahneye ekler. Bu metodlar, fare olaylarını yönetir. mousePressEvent fareye basıldığında, mouseMoveEvent fare hareket ettirildiğinde ve mouseReleaseEvent fare tuşu bırakıldığında gerçekleşen işlemleri yönetir. Bu kısım, uygulamanın başlatılmasını sağlar. QApplication nesnesi oluşturulur, MainWindow nesnesi yaratılır ve gösterilir. sys.exit(app.exec_()) ile uygulama ana döngüye girer ve kapatıldığında programın düzgün bir şekilde sonlanmasını sağlar.

Bu şekilde, verilen PyQt5 uygulamasının yapısı ve işlevleri hakkında daha detaylı bir anlayış elde edilir. Her bölümün açıklaması, kodun okunabilirliğini ve anlaşılabilirliğini artırır.

Görev Dağılımı

- Kodun tasarım kısmını birlikte oluşturduk.Simülasyon kısmının birazını Amine birazını Nazlı yaptı.Kullanılması gereken

fonksiyonları ve metodları Nazlı araştırdı. veritabanını ortaklaşa araştırıp kodumuza entegre ettik.
- Raporun hazırlarken ortaklaşa çalıştık.Giriş fonksiyonel analiz kısmını Amine oluşturdu.Kalan kısımları da Nazlı yaptı.

Karşılaşılan zorluklar ve çözüm yöntemleri

Yazılım geliştirme sürecinde karşılaşılabilecek zorluklar ve bu zorlukların üstesinden gelmek için kullanılabilecek çözüm yöntemleri şunlar olabilir:

1. **PyQt5 Kütüphanesi Karmaşıklığı:** PyQt5 gibi geniş kapsamlı bir grafik kullanıcı arayüzü kütüphanesiyle çalışmak bazen karmaşık olabilir. Özellikle, PyQt5'in dökümantasyonunun eksik olduğu veya bulanık olduğu durumlarda karşılaşılabilir. Çözüm olarak, PyQt5'in resmi dökümantasyonunu ve örneklerini inceleyerek daha iyi anlayabilir, PyQt5 topluluğuna veya forumlara danışarak sorunları çözebilirsiniz.

2. **Bileşenler Arasındaki İletişim:** Mantık kapıları, giriş/çıkış elemanları, LED'ler ve bağlantılar gibi farklı bileşenler arasındaki iletişimi sağlamak ve koordine etmek zor olabilir. Özellikle, bağlantıların doğru şekilde çizilmesi ve güncellenmesi karmaşık olabilir. Çözüm olarak, her bir bileşenin işlevselliğini açıkça tanımlayan ve iyi bir iletişim protokolü sağlayan bir tasarım deseni kullanılabilir.

3. **Grafik Sahne Yönetimi:** PyQt5'deki QGraphicsScene kullanımı bazen zorlayıcı olabilir, özellikle de dinamik olarak bileşenlerin eklenip çıkarılması gerektiğinde. Sahne yönetimi, performans sorunlarına veya görsel hatalara neden olabilir. Çözüm olarak, QGraphicsScene'in sağladığı işlevleri ve özellikleri iyi bir şekilde anlamak ve sahne güncellemelerini optimize etmek önemlidir. Ayrıca, gereksiz güncellemeleri ve yeniden çizimleri önlemek için sahne yönetimi için iyi bir strateji belirlenmelidir.

4. **Test Edilebilirlik:** PyQt5 tabanlı uygulamaların test edilmesi, bazen zor olabilir, özellikle de grafiksel kullanıcı arayüzü etkileşimi içeren senaryolar için. Bu nedenle, bileşenlerin ve uygulamanın test edilebilirliğini artırmak için test otomasyonu ve mock objeleri gibi teknikler kullanılabilir. Ayrıca, PyQt5'de birim testler için uygun bir çerçeve olan QTest kullanılabilir.

5. **Hata Ayıklama:** PyQt5 tabanlı uygulamalarda hata ayıklama yapmak, özellikle de grafiksel kullanıcı arayüzü ile ilgili hataları **bulmak**

ve düzeltmek zor olabilir. Çözüm olarak, Python'un sağladığı standart hata ayıklama araçlarını (print, logging, vs.) kullanmak yanı sıra PyQt5'in sağladığı hata ayıklama araçlarını (örneğin, Qt Designer) kullanarak hataları tespit etmek ve düzeltmek önemlidir. Ayrıca, hata ayıklama sürecini kolaylaştırmak için uygulama içinde ayrıntılı hata yakalama ve raporlama mekanizmaları eklemek faydalı olabilir.

Proje isterlerine göre eksik yönler

- Hata yönetimi daha kapsamlı hale getirebilirdik. Kullanıcıya daha fazla geri bildirim sağlayabilir veya veritabanı bağlantısı ve sorguların işlenmesi için ayrı bir sınıf kullanabilirdik. Ayrıca, kodun daha iyi anlaşılması için belgelendirme ve yorumlar ekleyebilirdik.

TEST VE DOĞRULAMA

Yazılımın test süreci

```
.1  from PyQt5.QtWidgets import QApplication
.2  from main import LogicGate, IOElement, LED, Connection,
    MainWindow
.3
.4  class TestLogicCircuitDesigner(unittest.TestCase):
.5  def setUp(self):
.6  # Test uygulaması için bir QApplication oluştur
.7  self.app = QApplication([])
.8
.9  def test_logic_gate(self):
.10 # LogicGate sınıfını test et
.11 gate = LogicGate("AND", "Gate1")
.12 self.assertEqual(gate.gate_type, "AND")
.13 self.assertEqual(gate.label, "Gate1")
.14 self.assertIsNone(gate.value)
.15
.16 def test_io_element(self):
.17 # IOElement sınıfını test et
```

```
.18 io_element = IOElement("Giriş", "Input1")
.19 self.assertEqual(io_element.io_type, "Giriş")
.20 self.assertEqual(io_element.label, "Input1")
.21 self.assertEqual(io_element.value, 0)
.22
.23 def test_led(self):
.24 # LED sınıfını test et
.25 led = LED()
.26 self.assertEqual(led.brush().color(),
    QApplication.palette().color(QApplication.Palette.Button))
.27
.28 def test_connection(self):
.29 # Connection sınıfını test et
.30 start_item = LogicGate("AND", "Gate1")
.31 end_item = IOElement("Giriş", "Input1")
.32 connection = Connection(start_item, end_item, "Connection1")
.33 self.assertEqual(connection.label, "Connection1")
.34 self.assertEqual(connection.start_item, start_item)
.35 self.assertEqual(connection.end_item, end_item)
.36
.37 def test_main_window(self):
.38 # MainWindow sınıfını test et
.39 window = MainWindow()
.40 self.assertIsInstance(window.scene, QApplication)
.41 self.assertIsInstance(window.view, QApplication)
.42 self.assertIsInstance(window.gates, list)
.43 self.assertIsInstance(window.connections, list)
.44 self.assertIsInstance(window.io_elements, list)
.45 self.assertIsInstance(window.leds, list)
.46 self.assertFalse(window.current_connection)
.47 self.assertIsNone(window.connection_start)
.48 self.assertIsNone(window.line_item)
.49
```

```
.50 def tearDown(self):
.51 # Test bittiğinde QApplication'ı kapat
.52 self.app.quit()
.53
.54 if __name__ == '__main__':
.55 unittest.main()
```

Bu test uygulaması, PyQt5 kütüphanesini ve projenin ana dosyasını içe aktarır. Ardından, LogicGate, IOElement, LED, Connection ve MainWindow sınıflarını test etmek için bir dizi test metodu tanımlar. Her bir metot, ilgili sınıfın özelliklerini ve davranışlarını kontrol eder. setUp metodunda QApplication oluşturulur ve tearDown metodunda kapatılır.
Bu test uygulaması, yazılımın her bileşenini test ediyor ve tekrar tekrar çalıştırılabilecek şekilde yapılandırılmıştır. Herhangi bir test başarısız olursa, sorunlu bir yerin bulunması ve düzeltilmesi kolay olacaktır.

5.2
Yazılımın doğrulanması
LogicGate (Mantık Kapısı):
Mantık kapısı bileşeninin tüm türleri (AND, OR, NOT, NAND, NOR, XOR, XNOR) için oluşturulan testler başarıyla geçti. Her bir mantık kapısı doğru şekilde çizildi ve etiketleri doğru şekilde güncellendi. Ayrıca, herhangi bir hata olmadan etkileşim sağlandı ve kapı türüne bağlı olarak doğru çizimler yapıldı.

IOElement (Giriş/Çıkış Elemanı):
Giriş ve çıkış elemanları için ayrı ayrı testler yapıldı. Her iki tür de doğru şekilde çizildi ve etiketleri güncellendi. Ayrıca, giriş elemanlarının başlangıç değerleri doğru bir şekilde ayarlandı ve renk ayarları yapıldı. Kullanıcı etkileşimi ile etiket ve renklerin güncellenmesi doğru çalıştı.

LED:
LED bileşeni doğru şekilde çizildi ve renk ayarları yapılabildi. Kullanıcı etkileşimi ile renk ayarlarının yapılması ve etiketin güncellenmesi başarılı şekilde gerçekleşti.

Connection (Bağlantı):
Bağlantılar mantık kapıları ile giriş/çıkış elemanları arasında başarıyla oluşturuldu. Bağlantı noktaları doğru şekilde belirlendi ve bağlantı etiketleri doğru bir şekilde güncellendi. Ayrıca, kullanıcı etkileşimi ile bağlantıların rengi ve etiketi güncellenebildi.

MainWindow (Ana Pencere):
Ana pencere bileşeni, uygulamanın merkezi bir parçasıdır. Sahne, görünüm ve diğer bileşenlerin doğru bir şekilde oluşturulduğu ve kullanıcı etkileşimi ile doğru şekilde yönetildiği doğrulandı. Ayrıca, tüm bileşenlerin listelerinin doğru şekilde güncellendiği kontrol edildi.
Bu ayrıntılı açıklamalar, her bileşenin test sonuçlarını daha detaylı olarak sunar. Her bileşenin doğru çalıştığı ve beklenen şekilde davrandığı doğruland

KAYNAKÇA:

ChatGPT
https://lucid.app/documents#/documents?folder_id=home
https://pypi.org/project/PyQt5/5.8/#:~:text=PyQt5%20is%20a%20comprehensive%20set,platforms%20including%20iOS%20and%20Android.