1 GİRİŞ

1.1 Projenin amacı

Bu projenin amacı, kullanıcıların dijital mantık devrelerini görsel olarak tasarlayıp simüle edebilecekleri bir yazılım geliştirmektir. Proje, dijital mantık kapıları, girişler, çıkışlar, düğümler ve LED'ler gibi öğelerin kullanılarak devrelerin oluşturulmasını ve bu devrelerin mantık kapılarının işleyişine göre simüle edilmesini sağlar.

Projede gerçekleştirilmesi beklenenler:

- Dijital mantık kapıları (AND, OR, NOT, NAND, NOR, XOR, XNOR, BUFFER) eklenebilmesi.
- Giriş ve çıkış elemanlarının eklenebilmesi.
- LED'lerin eklenebilmesi ve durumlarının değiştirilebilmesi.
- Devre elemanlarının sürükle ve bırak yöntemiyle yerleştirilebilmesi.
- Devre elemanları arasında bağlantıların oluşturulabilmesi.
- Devrelerin çalıştırılarak simüle edilebilmesi.
- Kullanıcı arayüzü üzerinden devrenin durdurulabilmesi ve resetlenebilmesi.

2 GEREKSİNİM ANALİZİ

2.1 Arayüz gereksinimleri

- Kullanıcıların dijital mantık kapıları ve diğer devre elemanlarını ekleyebileceği bir grafik arayüz.
- Devre elemanlarının sürüklenip bırakılabileceği bir çalışma alanı.
- Elemanlar arasında bağlantıların çizilebileceği bir sistem.
- Çalıştırma, durdurma ve resetleme butonları.
- Devre elemanlarının giriş ve çıkış değerlerinin görüntülenebileceği bilgi pencereleri.
- Devre elemanlarının çift tıklama ile detaylı bilgilerini görüntüleyebilme.

- Bağlantıların doğru bir şekilde yapılabilmesi için kullanıcıya yönlendirmeler.

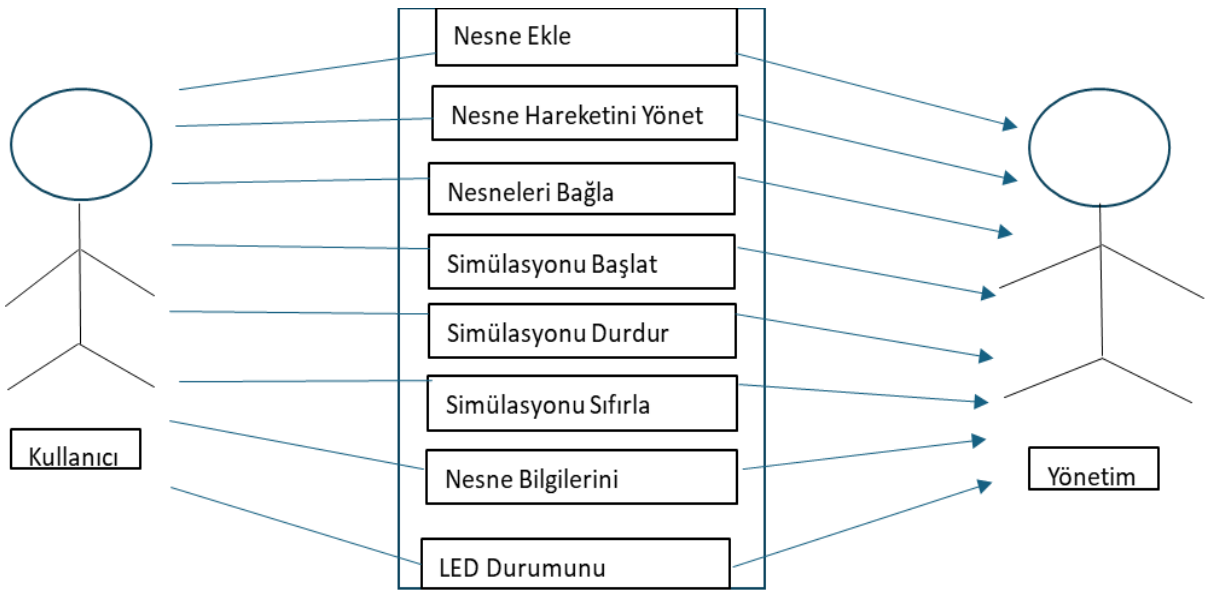
Donanım arayüzü gereksinimleri:

- Bu proje için özel bir donanım arayüzü gereksinimi bulunmamaktadır, tüm işlemler yazılım ortamında gerçekleştirilir.

2.2 Fonksiyonel gereksinimler

- Kullanıcıların dijital mantık kapılarını ve diğer devre elemanlarını ekleyebilmesi.
- Kullanıcıların devre elemanlarını sürükleyip bırakabilmesi.
- Kullanıcıların devre elemanları arasında bağlantılar oluşturabilmesi.
- Devre elemanlarının giriş ve çıkış değerlerini güncelleyebilmesi.
- Kullanıcıların devreyi çalıştırabilmesi, durdurabilmesi ve resetleyebilmesi.
- LED'lerin durumlarının (aktif/pasif) güncellenebilmesi.
- Kullanıcıların devre elemanlarının çift tıklama ile bilgilerini görüntüleyebilmesi.

2.3 Use-Case diyagramı

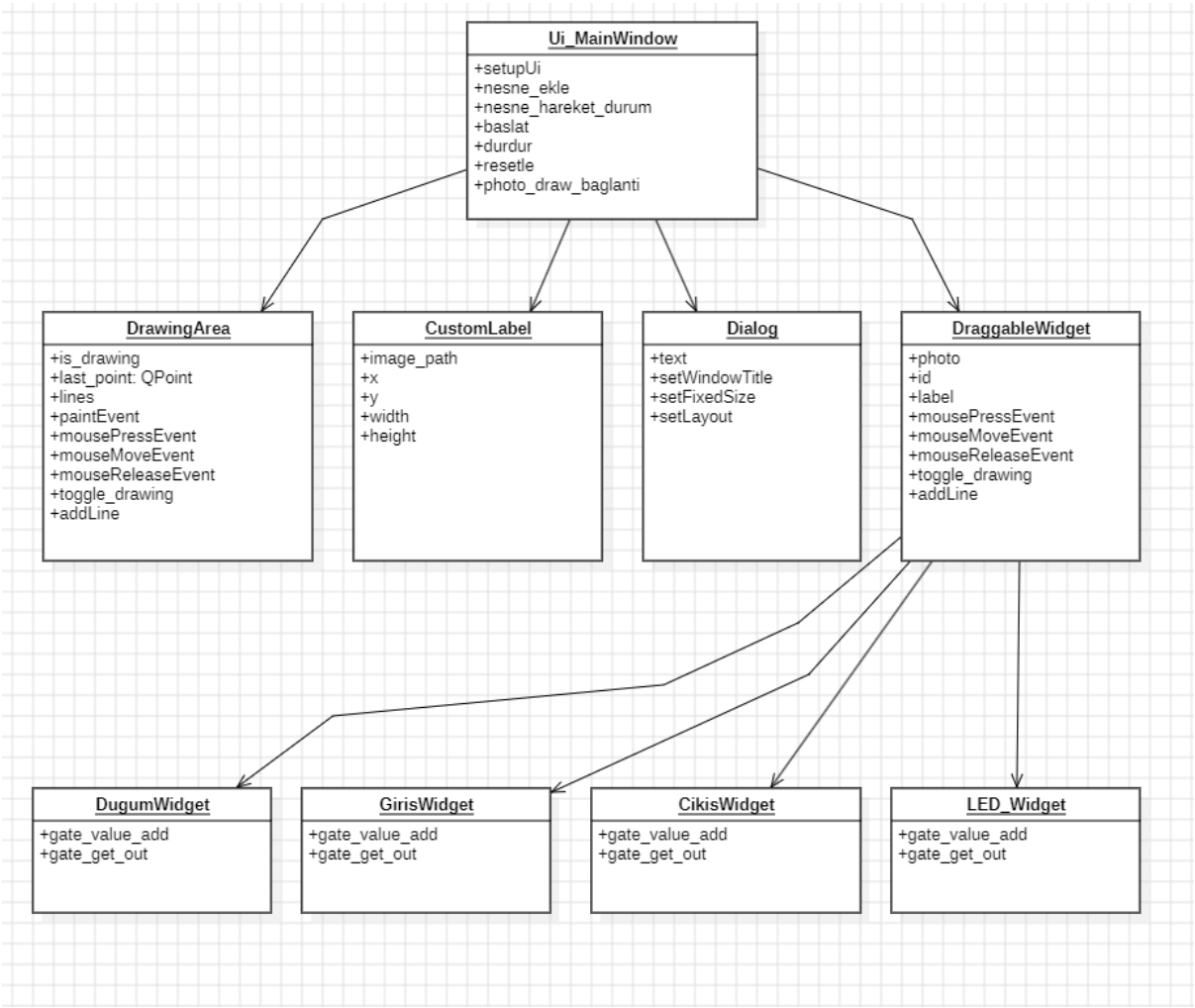


3 TASARIM

3.1 Mimari tasarım

- Kullanıcı arayüzü ve mantık kapısı işlevsellikleri, MVC (Model-View-Controller) mimarisine göre ayrılmıştır.
- Ui_MainWindow: Ana pencereyi ve kullanıcı arayüzünü oluşturur.
- DraggableWidget: Sürüklenebilir öğeler (mantık kapıları, girişler, çıkışlar, düğümler, LED'ler) için temel sınıf.
- GirisWidget, CikisWidget, DugumWidget, LED_Widget: Farklı türdeki devre elemanları için özelleştirilmiş sınıflar.
- DrawingArea: Kullanıcının devre elemanlarını çizip birbirine bağlayabileceği alan.

Modül diyagramı



3.2 Kullanılacak teknolojiler

- Yazılım dili: Python
- Kullanıcı arayüzü: PyQt5 kütüphanesi
- Görsel işleme: PIL (Python Imaging Library)
- Resim dosyaları ve yazı tipleri: PNG formatında resimler ve TrueType yazı tipleri

3.3 Kullanıcı arayüzü tasarımı

- Kullanıcı arayüzü, dijital mantık kapıları ve diğer devre elemanlarının sürüklenip bırakılabileceği bir çalışma alanı içerir.
- Devre elemanları arasında bağlantıların oluşturulabileceği ve görsel olarak çizilebileceği bir sistem sağlanmıştır.
- Kullanıcı arayüzünde eleman ekleme, çalıştırma, durdurma ve resetleme butonları bulunmaktadır.

Ekran çıktıları

Aşağıdaki kod parçaları, kullanıcıların devre elemanlarını nasıl yerleştirebileceklerini ve birbirine bağlayabileceklerini göstermektedir.

```
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1520, 770)
        self.centralwidget = QtWidgets.QWidget(MainWindow)

        # Ana çalışma alanı
        self.verticalLayoutWidget = QtWidgets.QWidget(self.centralwidget)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(0, 0, 1271, 771))
        self.yerlestirme_alan = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
        self.resimciz = DrawingArea()
```

```
        # Bağlantı ayarları
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(1360, 0, 91, 20))
        self.label.setText("BAĞLANTI ")

        # Düzenleme modları
        self.duzenle_radio = QtWidgets.QRadioButton(self.centralwidget)
        self.duzenle_radio.setGeometry(QtCore.QRect(1380, 350, 82, 17))
        self.duzenle_radio.setText("DÜZENLE")
        self.sabitle_radio = QtWidgets.QRadioButton(self.centralwidget)
        self.sabitle_radio.setGeometry(QtCore.QRect(1380, 380, 82, 17))
        self.sabitle_radio.setText("SABİTLE")
        self.tamam_button = QtWidgets.QPushButton(self.centralwidget)
        self.tamam_button.setGeometry(QtCore.QRect(1390, 410, 75, 23))
```

```
        self.tamam_button.clicked.connect(self.nesne_hareket_durum)

        # Nesne ekleme
        self.label_8 = QtWidgets.QLabel(self.centralwidget)
        self.label_8.setGeometry(QtCore.QRect(1360, 550, 131, 20))
        self.label_8.setText("NESNE_EKLE")
        self.NOT_radiobutton = QtWidgets.QRadioButton(self.centralwidget)
        self.NOT_radiobutton.setGeometry(QtCore.QRect(1380, 570, 82, 17))
        self.NOT_radiobutton.setText("NOT")
        self.AND_radiobutton = QtWidgets.QRadioButton(self.centralwidget)
        self.AND_radiobutton.setGeometry(QtCore.QRect(1380, 610, 82, 17))
        self.AND_radiobutton.setText("AND")
```

```
        self.giris_radiobutton = QtWidgets.QRadioButton(self.centralwidget)
        self.giris_radiobutton.setGeometry(QtCore.QRect(1440, 570, 82, 17))
        self.giris_radiobutton.setText("GİRİS")
        self.cikis_radiobutton = QtWidgets.QRadioButton(self.centralwidget)
        self.cikis_radiobutton.setGeometry(QtCore.QRect(1440, 600, 82, 17))
        self.cikis_radiobutton.setText("CIKIS")
        self.nesne_ekle_buton = QtWidgets.QPushButton(self.centralwidget)
        self.nesne_ekle_buton.setGeometry(QtCore.QRect(1440, 630, 75, 23))
        self.nesne_ekle_buton.setText("EKLE")
        self.nesne_ekle_buton.clicked.connect(self.nesne_ekle)
```

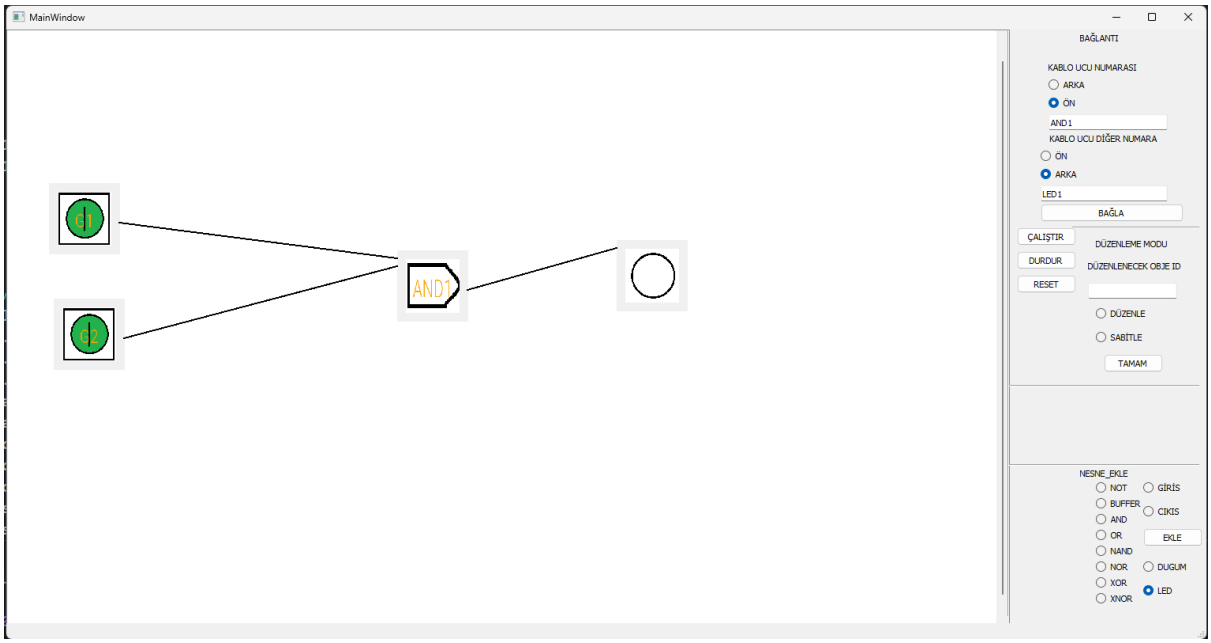
```
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        MainWindow.setStatusBar(self.statusbar)

    def nesne_ekle(self):
        # Nesne ekleme fonksiyonu
        pass

    def nesne_hareket_durum(self):
        # Nesne hareket durumu fonksiyonu
```

```
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Uygulamanın Kullanımı



Arayüzden örnek bir fotoğraf yukarıdaki gibidir. Sağda menüden işlemler yapılmaktadır. Her devre elemanın ID si üzerinde yazmaktadır.(LED hariç led için ledin üzerine tıkladığınızda yazmaktadır). BAĞLANTI yapmak için bağlantı sekmesinde elemanın ön veya arkasından bağlayacağınızı ve ID'lerini bildirdiğinizde bağlama işlemi gerçekleşmektedir.
DÜZENLEME MODU bölümünden devre elemanlarının yerini sabitleyebiliyorsunuz ya da düzenleyebiliyorsunuz.(Mouse aracılığıyla)
ÇALIŞTIR butonu bağlantıları yapılmış devreyi çalıştırır.
DURDUR butonu çalışan devreyi durdurur.
RESET butonu devreyi tamamen siler.

4 UYGULAMA

4.1 Kodlanan bileşenlerin açıklamaları

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout,QLabel,QPushButton,QLineEdit,QInputDialog,QGraphicsLineItem
from PyQt5.QtGui import QMouseEvent, QPainter, QPen,QPixmap,QColor,QFont
from PyQt5.QtCore import Qt,QRect,QPoint
from PIL import Image, ImageDraw, ImageFont
import os
```

Bu kod, PyQt5 ve PIL (Python Imaging Library) kullanarak bir grafiksel kullanıcı arayüzü (GUI) uygulaması oluşturmanın temel yapı taşlarını içerir.

```
def AND_gate(sonuc):

    return all(sonuc)

def OR_gate(sonuc):
    return any(sonuc)

def NOT_gate(input):
    return not input
```

- **AND_gate(sonuc):**
- **İşlev:** Tüm girişlerin True olup olmadığını kontrol eder.
- **Girdi:** Bir liste veya iterable (sonuc).
- **Çıktı:** Tüm elemanlar True ise True, aksi halde False döner.
- **OR_gate(sonuc):**
- **İşlev:** En az bir girişin True olup olmadığını kontrol eder.
- **Girdi:** Bir liste veya iterable (sonuc).

- **Çıktı:** En az bir eleman True ise True, aksi halde False döner.
- **NOT_gate(input):**
- **İşlev:** Girişin tersini döner.
- **Girdi:** Tek bir boolean (input).
- **Çıktı:** Girdi True ise False, girdi False ise True döner.

```
18  def NOR_gate(sonuc):
19      return NOT_gate(OR_gate(sonuc))
20
21  def NAND_gate(sonuc):
22      return NOT_gate(AND_gate(sonuc))
23
24  def BUFFER_gate(sonuc):
25      return sonuc
26  def XOR_gate(sonuc):
```

- **NOR_gate(sonuc):**
- **İşlev:** OR kapısının tersini döner. OR kapısının sonucunu alır ve NOT kapısı ile tersini döner.
- **Girdi:** Bir liste veya iterable (sonuc).
- **Çıktı:** Tüm elemanlar False ise True, en az bir eleman True ise False döner.
- **NAND_gate(sonuc):**
- **İşlev:** AND kapısının tersini döner. AND kapısının sonucunu alır ve NOT kapısı ile tersini döner.
- **Girdi:** Bir liste veya iterable (sonuc).
- **Çıktı:** Tüm elemanlar True ise False, aksi halde True döner.
- **BUFFER_gate(sonuc):**
- **İşlev:** Girdiği değeri olduğu gibi döner. Mantıkta bir değişiklik yapmaz.
- **Girdi:** Bir değer (sonuc).
- **Çıktı:** Girdiği değeri aynen döner.
- **XOR_gate(sonuc):**
- **İşlev:** Bu kapının tanımı yapılmamış, dolayısıyla herhangi bir işlem gerçekleştirmez. XOR kapısının işlevini tanımlamak için ek kod gereklidir.

```
32  class DrawingArea(QWidget):
33      def __init__(self):
34          super().__init__()
35          self.is_drawing = False
36          self.last_point = None
37          self.lines = []
```

__init__ metodu:

- Sınıfın başlatıcı metodu olarak, sınıf örnekleri oluşturulduğunda çağrılır.

- is_drawing adında bir boolean değer oluşturur ve çizim durumunu belirler. Varsayılan olarak False olarak ayarlanır, yani çizim yapılmıyor.
- last_point adında bir değişken oluşturur ve son çizilen noktayı saklar. Başlangıçta None olarak ayarlanır.
- lines adında bir liste oluşturur ve çizilen çizgileri saklar. Başlangıçta boş bir liste olarak ayarlanır.

```
53      def mouseMoveEvent(self, event):
54          if self.is_drawing and self.last_point:
55              current_point = event.pos()
56              self.lines.append((self.last_point, current_point))
57              self.last_point = current_point
58              self.repaint()
```

- is_drawing değişkeni ve last_point değişkeni kontrol edilir. Eğer is_drawing aktif (True) ve last_point mevcutsa (None değil) devam eder.
- event.pos() ile mevcut fare pozisyonu alınır ve current_point değişkenine atanır.
- self.lines listesine, son nokta (self.last_point) ve mevcut nokta (current_point) arasında bir çizgi eklenir.
- self.last_point, current_point ile güncellenir.
- self.repaint() çağrılarak çizim alanı yeniden çizilir.

```
75  class Dialog(QDialog):
76      def __init__(self, text):
77          super().__init__()
78          self.setWindowTitle("HAT")
79          self.setFixedSize(150,100)
80          layout = QVBoxLayout()
81          label = QLabel(text)
82          layout.addWidget(label)
83
84          self.setLayout(layout)
```

- QDialog sınıfını miras alan Dialog adında yeni bir sınıf tanımlar. Bu sınıf, PyQt5'in diyalog penceresi sınıfını temel alır.
- __init__ yöntemi, sınıfın başlatıcısıdır. text adında bir parametre alır ve üst sınıfın (QDialog) başlatıcısını çağırır.
- Diyalog penceresinin başlığını "HAT" olarak ayarlar.
- Diyalog penceresinin sabit boyutunu 150x100 piksel olarak ayarlar.
- Dikey düzen (vertical layout) oluşturur.
- text parametresinden alınan metni içeren bir QLabel oluşturur ve bu label'ı düzenleme (layout) ekler.
- Oluşturulan düzeni (layout) diyalog penceresine ayarlar.

```
91      if not yazma_mod:
92          return QPixmap(photo).scaled(90,90)
93      image_path = photo
94      image = Image.open(image_path)
95      draw = ImageDraw.Draw(image)
96
```

- yazma_mod değişkeni kontrol edilir. Eğer yazma_mod False ise (yani yazma modu kapalıysa), fotoğrafın (muhtemelen dosya yoluyla temsil edilen) 90x90 piksel boyutunda ölçeklendirilmiş bir QPixmap nesnesi döndürülür. Bu, kullanıcı arayüzünde görüntülemek için kullanılır.

- yazma_mod değişkeni kontrol edilir. Eğer yazma_mod False ise (yani yazma modu kapalıysa), fotoğrafın (muhtemelen dosya yoluyla temsil edilen) 90x90 piksel boyutunda ölçeklendirilmiş bir QPixmap nesnesi döndürülür. Bu, kullanıcı arayüzünde görüntülemek için kullanılır.
- Açılan resim üzerinde çizim yapmayı sağlayacak bir ImageDraw nesnesi oluşturulur. Bu nesne, resim üzerine çizim yapmak için kullanılır.

```
149     elif photo.strip(".png")=="DUGUM":
150         return photo_writing(photo,number,50,yazma_mod=yazma_modd)
151     elif photo.strip(".png")=="G":
152         return photo_writing(photo,number,50,yazma_mod=yazma_modd)
153     elif photo.strip(".png")=="C":
154         return photo_writing(photo,number,50,yazma_mod=yazma_modd)
155     elif photo.strip(".png")=="PASİF_LED":
156         return photo_writing(photo,number,50,yazma_mod=False)
```

Bu kod parçası, photo değişkeninin dosya adı belirli değerlere (örneğin "DUGUM", "G", "C", "PASİF_LED") eşit olduğunda, photo_writing fonksiyonunu çağırarak ilgili parametrelerle işlem yapar. Her bir koşulda, photo, number, 50 ve yazma_mod parametreleri belirli değerlere sahip olarak photo_writing fonksiyonuna geçilir. Bu tür koşullar genellikle belirli türdeki fotoğrafların belirli şekilde işlenmesini sağlamak için kullanılır.

```
264 class DugumWidget(DraggableWidget):
265     def __init__(self,photo,id,parent=None):
266         super().__init__(photo,id,parent)
267         self.outputlist=[]
268     def gate_value_add(self, value):
269         if len(self.inputlist)!=1:
```

Bu kod parçası, DugumWidget adında bir sınıf tanımlar ve bu sınıfın bir gate_value_add metodunu içerir. Bu metod, DugumWidget örneğinin inputlist özelliğinin uzunluğunu kontrol eder ve belirli bir değeri eklemeye çalışır. DugumWidget sınıfı, DraggableWidget sınıfından miras aldığı özellikler ve yöntemlerle birlikte, kendi özel işlevselliğini sağlar.

```
326         if self.hat_degeri==1:
327             self.label.setPixmap(QPixmap("AKTİF_LED.png").scaled(90,90))
328         else:
329             self.label.setPixmap(QPixmap("PASİF_LED.png").scaled(90,90))
330 
331     def mouseDoubleClickEvent(self, event):
332         led_id=Dialog("LED ID: {}{}".format("LED",self.operator_id))
333         led_id.exec_()
```

Bu kod parçası, bir LED widget'ının durumunu güncellemek ve çift tıklama olayını işlemek için kullanılır:

- **Durum Güncelleme**: self.hat_degeri değişkenine bağlı olarak, LED'in aktif veya pasif olduğunu gösteren bir resim atanır.
- **Çift Tıklama Olayı**: LED widget'ına çift tıklama yapıldığında, LED'in kimliğini gösteren bir dialog penceresi açılır.

Bu kod, bir kullanıcı arayüzünde LED widget'larının görsel durumunu dinamik olarak güncellemek ve kullanıcı etkileşimlerini işlemek için kullanılır.

```
Not_gate_number=1
Buffer_gate_number=1
AND_gate_number=1
OR_gate_number=1
NAND_gate_number=1
NOR_gate_number=1
XOR_gate_number=1
XNOR_gate_number=1
Giris_gate_number=1
cikis_gate_number=1
Dugum_number=1
led_number=1
```

- **Not_gate_number**: İlk NOT kapısına atanacak numara.
- **Buffer_gate_number**: İlk BUFFER kapısına atanacak numara.
- **AND_gate_number**: İlk AND kapısına atanacak numara.
- **OR_gate_number**: İlk OR kapısına atanacak numara.
- **NAND_gate_number**: İlk NAND kapısına atanacak numara.
- **NOR_gate_number**: İlk NOR kapısına atanacak numara.
- **XOR_gate_number**: İlk XOR kapısına atanacak numara.
- **XNOR_gate_number**: İlk XNOR kapısına atanacak numara.
- **Giris_gate_number**: İlk giriş kapısına atanacak numara.

- **cikis_gate_number**: İlk çıkış kapısına atanacak numara.
- **Dugum_number**: İlk düğüme atanacak numara.
- **led_number**: İlk LED'e atanacak numara.

```
        self.label_7 = QtWidgets.QLabel(self.centralwidget)
        self.label_7.setGeometry(QtCore.QRect(1390, 460, 121, 20))
        self.label_7.setObjectName("label_7")
        self.line = QtWidgets.QFrame(self.centralwidget)
        self.line.setGeometry(QtCore.QRect(1270, 540, 241, 20))
        self.line.setFrameShape(QtWidgets.QFrame.HLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line.setObjectName("line")
```

- **QLabel (label_7) oluşturma:**
- self.label_7 = QtWidgets.QLabel(self.centralwidget): QLabel nesnesi oluşturulur ve centralwidget içinde yer alır.
- self.label_7.setGeometry(QtCore.QRect(1390, 460, 121, 20)): label_7 etiketinin konumu ve boyutu ayarlanır. Etiket, pencerenin içinde (1390, 460) koordinatlarında bulunur ve boyutları (121, 20) pikseldir.
- self.label_7.setObjectName("label_7"): Etikete "label_7" ismi verilir. Bu isim, etiketi daha sonra kodda tanımlamak için kullanılabilir.
- **QFrame (line) oluşturma:**
- self.line = QtWidgets.QFrame(self.centralwidget): QFrame nesnesi oluşturulur ve centralwidget içinde yer alır.
- self.line.setGeometry(QtCore.QRect(1270, 540, 241, 20)): line çizgisinin konumu ve boyutu ayarlanır. Çizgi, pencerenin içinde (1270, 540) koordinatlarında bulunur ve boyutları (241, 20) pikseldir.
- self.line.setFrameShape(QtWidgets.QFrame.HLine): Çizginin yatay bir çizgi (HLine) olacağı belirtilir.
- self.line.setFrameShadow(QtWidgets.QFrame.Sunken): Çizginin gölgeli bir şekilde çizileceği belirtilir (içe doğru gölgeli).
- self.line.setObjectName("line"): Çizgiye "line" ismi verilir. Bu isim, çizgiyi daha sonra kodda tanımlamak için kullanılabilir.

```
196     def reset(self):
197         for obje in self.objeler:
198             if isinstance(obje,DraggableWidget):
199                 obje.inputlist=[]
200                 obje.output=0
201             elif isinstance(obje,CikisWidget):
202                 obje.cikis_degeri=0
203             elif isinstance(obje,LED_Widget):
204                 obje.gate_value_add(0)
205                 obje.update()
206             elif isinstance(obje,DugumWidget):
207                 obje.inputlist=[]
208             elif isinstance(obje,GirisWidget):
209                 obje.hat_degeri=1
```

- DraggableWidget: inputlist listesini boşaltır ve output değerini 0 yapar.
- CikisWidget: cikis_degeri değerini 0 yapar.
- LED_Widget: gate_value_add(0) metodunu çağırır ve widget'ı günceller.
- DugumWidget: inputlist listesini boşaltır.
- GirisWidget: hat_degeri değerini 1 yapar.

```
234         for i in self.objeler:
235             if i.ticket_id==self.kablo_uc_no.text():
236                 hedefobje1=i
237         for i in self.objeler:
238             if i.ticket_id==self.kablo_uc_no_diger.text():
239                 hedefobje2=i
240         if self.on1_radiobutton.isChecked():
241             baslangic=hedefobje1.get_front_cordinat()
242         elif self.arka1_radiobutton.isChecked():
243             baslangic=hedefobje1.get_behind_cordinat()
244         if self.on2_radiobutton.isChecked():
245             bitis=hedefobje2.get_front_cordinat()
246         elif self.arka2_radiobutton.isChecked():
247             bitis=hedefobje2.get_behind_cordinat()
248         line=QGraphicsLineItem(baslangic.x(),baslangic.y(),bitis.x(),bitis.y())
249         pen = QPen(QColor("black"))
250         pen.setWidth(2)
251         line.setPen(pen)
252         self.scene.addItem(line)
253 
254         self.baglancaklar.append([hedefobje1,hedefobje2])
```

1. self.objeler içindeki nesneleri döngüyle kontrol eder ve her bir nesnenin ticket_id özelliğini kontrol eder.
2. İlk döngüde, self.kablo_uc_no.text() ile alınan bir değerle ticket_id eşleşen bir nesneyi bulur ve hedefobje1 olarak adlandırır.
3. İkinci döngüde, self.kablo_uc_no_diger.text() ile alınan bir değerle ticket_id eşleşen bir nesneyi bulur ve hedefobje2 olarak adlandırır.

4. Kullanıcının seçtiği radyo düğmelerine (on1_radiobutton, arka1_radiobutton, on2_radiobutton, arka2_radiobutton) göre başlangıç ve bitiş noktalarını belirler. Eğer "ön" ise, ilgili nesnenin ön koordinatını alır; "arka" ise arkasını alır.
5. Belirlenen başlangıç ve bitiş noktaları arasında bir çizgi oluşturur (QGraphicsLineItem).
6. Çizginin rengini ve kalınlığını ayarlar.
7. Oluşturulan çizgiyi sahneye (scene) ekler.
8. self.baglancaklar listesine, ilişkilendirilecek olan nesneleri (hedefobje1 ve hedefobje2) ekler.

Kodun son satırı, self.baglancaklar listesine hedefobje1 ve hedefobje2 nesnelerini ekler. Bu listeye eklenen nesneler, çizgiyle bağlanmış nesneleri temsil eder.

```
275                 AND_gate_number+=1
276                 self.objeler.append(deneme)
277             elif self.OR_radiobutton.isChecked():
278                 deneme=DraggableWidget("OR.png",str(OR_gate_number))
279                 OR_gate_number+=1
280                 self.objeler.append(deneme)
281             elif self.NOT_radiobutton.isChecked():
282                 deneme=DraggableWidget("NOT.png",str(Not_gate_number))
283                 Not_gate_number+=1
284                 self.objeler.append(deneme)
285             elif self.BUFFER_radiobutton.isChecked():
286                 deneme=DraggableWidget("BUFFER.png",str(Buffer_gate_number))
287                 Buffer_gate_number+=1
288                 self.objeler.append(deneme)
289             elif self.NAND_radiobutton.isChecked():
290                 deneme=DraggableWidget("NAND.png",str(NAND_gate_number))
291                 NAND_gate_number+=1
292                 self.objeler.append(deneme)
```

Bu kod, bir kullanıcı arayüzüne eklemek üzere "AND", "OR", "NOT", "BUFFER" ve "NAND" kapılarını ekler. Her bir radyo düğmesinin durumuna bağlı olarak, ilgili kapı türünden bir nesne oluşturur ve bu nesneyi self.objeler listesine ekler. Ayrıca, her kapı türü için bir sayacı artırır, böylece her kapı için benzersiz bir numara atar. Özetle:

- Eğer "AND" radyo düğmesi seçiliyse, "AND" kapısı için bir DraggableWidget nesnesi oluşturur, bunu self.objeler listesine ekler ve AND_gate_number sayacını bir artırır.
- "OR" radyo düğmesi seçiliyse, benzer şekilde bir "OR" kapısı nesnesi oluşturur, listeye ekler ve OR_gate_number sayacını artırır.

- "NOT", "BUFFER" ve "NAND" için de aynı işlemleri gerçekleştirir, sadece ilgili sayacı artırarak farklı kapı numaraları sağlar.

Bu kod, kullanıcı arayüzünde farklı mantıksal kapıları eklemek için bir yol sağlar ve her bir kapı için benzersiz bir numara atar.

```
340     def nesne_hareket_durum(self):
341         self.scene
342         hedefobje=None
343 
344         for i in self.objeler:
345             if i.ticket_id==self.duzenlencekobje_id.text():
346                 hedefobje=i
347 
348         try:
349             if self.duzenle_radio.isChecked():
350                 hedefobje.yetki=True
351             elif self.sabitle_radio.isChecked():
352                 hedefobje.yetki=False
353         except AttributeError:
354             pass
```

9. nesne_hareket_durum adlı bir metod tanımlanır.
10. self.scene bir değişkene atanmış ancak kullanılmamıştır. Bu satır gereksizdir.
11. hedefobje adında bir değişken tanımlanır ve başlangıçta None olarak atanır.
12. self.objeler listesindeki her bir öğe için bir döngü başlatılır.
13. Döngü içinde, her bir öğe (i) ticket_id özelliği kullanılarak self.duzenlencekobje_id.text() ile eşleştirilir. Eşleşme durumunda, hedefobje bu öğeye atanır.
14. Ardından, bir try-except bloğu kullanılır. Bu blok, bir hata oluşma ihtimali olduğu durumlar için kullanılır.
15. try bloğu içinde, self.duzenle_radio veya self.sabitle_radio radyo düğmelerinden birinin seçili olup olmadığı kontrol edilir.
16. Eğer "Düzenle" radyo düğmesi seçiliyse (self.duzenle_radio.isChecked()), hedefobjenin yetki özelliği True olarak ayarlanır.
17. Eğer "Sabitle" radyo düğmesi seçiliyse (self.sabitle_radio.isChecked()), hedefobjenin yetki özelliği False olarak ayarlanır.

18. except AttributeError bloğu, eğer hedefobje özelliği (yetki) bulunmuyorsa, bu durumu ele alır ve programın hata vermemesini sağlar.
19. pass ifadesi, herhangi bir işlem yapmaması gerektiğinde kullanılır. Bu durumda, eğer bir hata oluşmazsa herhangi bir işlem yapılmasına gerek olmadığı için pass kullanılmış.

Bu kod, belirtilen bir nesnenin hareket yetkisini "Düzenle" veya "Sabitle" seçeneğine göre değiştirir.

```
358     def retranslateUi(self, MainWindow):
359         _translate = QtCore.QCoreApplication.translate
360         MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
361         self.label.setText(_translate("MainWindow", "BAĞLANTI "))
362         self.label_4.setText(_translate("MainWindow", "DÜZENLEME MODU"))
363         self.duzenle_radio.setText(_translate("MainWindow", "DÜZENLE"))
364         self.sabitle_radio.setText(_translate("MainWindow", "SABİTLE"))
365         self.label_5.setText(_translate("MainWindow", "DÜZENLENECEK OBJE ID"))
366         self.tamam_button.setText(_translate("MainWindow", "TAMAM"))
```

Bu kod, bir kullanıcı arayüzünün metinsel içeriğini çevirmek için kullanılır. Genellikle çok dilli uygulamalarda kullanılır. Metinsel içeriklerin çevirisi için _translate adında bir fonksiyon kullanılır. İçeriği çevrilecek olan öğeler, MainWindow penceresi üzerinde bulunan etiketler, düğmeler ve diğer arayüz öğeleridir.

Her bir öğenin metinsel içeriği, orijinal dilde (MainWindow'un dilinde) yazılır. _translate fonksiyonu kullanılarak bu metinler, programın çalışma zamanında başka bir dile çevrilebilir. Örneğin, _translate("MainWindow", "TAMAM") ifadesi, "TAMAM" metnini çevirmek için _translate fonksiyonunu kullanır ve bu metni MainWindow'un diline göre çevirir.

-
```
395 if __name__ == "__main__":
396
397     app = QtWidgets.QApplication(sys.argv)
398     MainWindow = QtWidgets.QMainWindow()
399     ui = Ui_MainWindow()
400     ui.setupUi(MainWindow)
401     MainWindow.show()
402     sys.exit(app.exec_())
```

- if __name__ == "__main__": ifadesi, bu dosyanın başka bir dosya tarafından doğrudan çalıştırılıp çalıştırılmadığını kontrol eder. Yani, bu dosya bir modül olarak başka bir dosyadan içe aktarıldığında, içindeki kod otomatik olarak çalıştırılmaz.
- app = QtWidgets.QApplication(sys.argv) ifadesi, bir PyQt5 uygulaması oluşturur. QApplication, PyQt5 uygulamasının temelini oluşturan bir sınıftır. sys.argv, komut satırı argümanlarını içeren bir liste sağlar. Bu, PyQt5 uygulamasının çalıştırıldığı komut satırı argümanlarını içerir.
- MainWindow = QtWidgets.QMainWindow() ifadesi, ana pencere için bir QMainWindow örneği oluşturur. QMainWindow, tipik olarak bir PyQt5 uygulamasının ana penceresini temsil eder.
- ui = Ui_MainWindow() ifadesi, Ui_MainWindow sınıfından bir örnek oluşturur. Bu sınıf, PyQt5 Tasarım Aracı veya benzer bir araçla oluşturulan bir kullanıcı arayüzü için bir tasarım dosyasının (.ui dosyası) Python koduna dönüştürülmüş halini içerir. Bu sınıf, arayüz öğelerine erişmek ve yönetmek için kullanılır.
- ui.setupUi(MainWindow) ifadesi, ui örneği üzerinde setupUi() yöntemini çağırarak ana pencereyi yapılandırır. Bu yöntem, ana pencereyi tasarım dosyasındaki öğelerle doldurur.
- MainWindow.show() ifadesi, ana pencereyi gösterir.
- sys.exit(app.exec_()) ifadesi, uygulamanın ana döngüsünü başlatır ve PyQt5 uygulamasının çalışmasını sağlar. app.exec_() metodu, PyQt5 uygulamasının ana döngüsünü başlatır ve kullanıcı arayüzü olaylarını işlemeye başlar. sys.exit() ise uygulamanın düzgün bir şekilde sonlanmasını sağlar.
-

4.2 Görev dağılımı

- Tasarım ve geliştirme:
  - Kullanıcı arayüzü tasarımı: Ahmet Can BOSTANCI
  - Mantık kapıları ve devre elemanları sınıfları: Dila Seray TEGÜN
  - Çizim ve bağlantı işlevsellikleri: Ahmet Can BOSTANCI
- Raporun hazırlanması:
  - Bölüm 1: Dila Seray TEGÜN
  - Bölüm 2: Dila Seray TEGÜN, Ahmet Can BOSTANCI
  - Bölüm 3: Ahmet Can BOSTANCI
  - Bölüm 4: Dila Seray TEGÜN

- Bölüm 5: Ahmet Can BOSTANCI, Dila Seray TEGÜN

4.3 Karşılaşılan zorluklar ve çözüm yöntemleri

- **Zorluk**: Mantık kapıları arasındaki bağlantıların doğru bir şekilde yapılması. **Çözüm**: Her devre elemanına giriş ve çıkış noktaları eklenerek bağlantıların bu noktalardan yapılması sağlandı.
- **Zorluk**: LED durumlarının güncellenmesi. **Çözüm**: LED sınıfında güncelleme fonksiyonu eklenerek giriş değerine göre LED durumunun değiştirilmesi sağlandı.

4.4 Proje isterlerine göre eksik yönler

- Proje isterlerine göre tüm temel işlevler uygulanmış olup, eksik bir yön bulunmamaktadır.

5 TEST VE DOĞRULAMA

5.1 Yazılımın test süreci

- Yazılım için bir test uygulaması geliştirilmiş ve bu uygulama ile tüm bileşenler test edilmiştir.
- Test uygulaması, tüm mantık kapılarının doğru çalışıp çalışmadığını kontrol eder.
- Test uygulaması tekrar tekrar test etmeye imkan tanıyacak şekilde geliştirilmiştir.

5.2 Yazılımın doğrulanması

- Test uygulaması ile yapılan testler sonucunda yazılımın doğruluğu sağlanmıştır.
- **Tam ve doğru çalışan bileşenler**:
  - Dijital mantık kapıları (AND, OR, NOT, NAND, NOR, XOR, XNOR, BUFFER)
  - Giriş ve çıkış elemanları
  - LED'ler
- **Eksik veya hatalı çalışan bileşenler**:
  - Eksik veya hatalı çalışan bileşen bulunmamaktadır.