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

PROJE KONUSU:
BASİT MANTIK DEVRESİ SİMÜLASYONU

ÖĞRENCİ ADI:
ÖĞRENCİ NUMARASI:
Senem ADALAN - 220502045
Sema Su YILMAZ - 220502016

DERS SORUMLUSU:
PROF. DR. NEVCİHAN DURU

TARİH:
2 HAZİRAN 2024

İÇİNDEKİLER

1. GİRİŞ

1.1 Projenin Amacı

Bu projenin amacı basit mantık devrelerini tasarlamak için bir platform geliştirmektir.Projede gerçekleştirilmesi beklenen işlemler şu şekildedir;

- Mantık Kapıları
- Giriş Çıkış Elemanları
- Bağlantı Elemanları
- Kontrol Tuşları
- Özellik Tabloları

2. GEREKSİNİM ANALİZİ

2.1 Arayüz Gereksinimleri

- **Araçlar Bölümü**
    - **Mantık Kapıları:**Kullanıcı, mantık kapıları arasında seçim yapabilir ve tasarım alanına ekleyebilir.Her mantık kapısı için bir özellik tablosu bulunmalıdır (etiket, giriş bağlantı sayısı).
    - **Giriş/Çıkış Elemanları:**Kullanıcı, giriş ve çıkış elemanları ekleyebilir.Her eleman için bir özellik tablosu bulunmalıdır (etiket, renk, başlangıç değeri).
    - **Bağlantı Elemanları:**Kullanıcı, bağlantı elemanları ekleyebilir.Her bağlantı elemanı için bir özellik tablosu bulunmalıdır (etiket, renk).
    - **Kontrol Tuşları:**Tasarım işlemi sırasında çalıştır, reset ve durdur tuşları olmalıdır.
- **Özellik Tabloları:**Her bir eleman için özellik tabloları olmalıdır.Özellik tablolarındaki değerler, tasarım alanına yerleştirilen elemanlar üzerinde sağ fare tıklaması ile görüntülenebilir ve değiştirilebilir.
- **Uygulama Ekranı:**Tasarım alanında kullanıcı, araçlar bölümünden seçtiği elemanları ekleyebilmelidir.Aynı devre elemanından birden fazla eklenmesine izin verilmelidir.
- **Bağlantılar:**Bağlantı hatlarının herhangi bir noktada birleştirilmesi gerekiyorsa bağlantı düğümleri kullanılmalıdır.
- **Simülasyon Arayüzü:**Tasarım işlemi bittikten sonra, çalıştır, reset ve durdur tuşları ile devrenin çalışması simüle edilmelidir.Kullanıcı, her bir elemanın giriş kutusunu seçerek giriş değerini değiştirebilir.Kullanıcı, devrenin çıkışında LED veya çıkış kutusu kullanabilir.Çıkış kutusu çıkış değerini göstermelidir ve LED, devrenin çıkış değerine göre yanıp sönmelidir.

2.2 Fonksiyonel Gereksinimler

Mantık Kapıları (İG-1)
- Kullanıcı, araçlar bölümünde bulunan mantık kapıları arasından seçim yapabilir.
- Her mantık kapısı için bir özellik tablosu bulunmalıdır (etiket, giriş bağlantı sayısı).
- Kullanıcı, seçtiği mantık kapısını tasarım alanına ekleyebilmelidir.
- Aynı mantık kapısından birden fazla eklenmesine izin verilmelidir.

Giriş/Çıkış Elemanları (İG-2)
- Kullanıcı, giriş ve çıkış elemanları arasından seçim yapabilir.
- Her eleman için bir özellik tablosu bulunmalıdır (etiket, renk, başlangıç değeri).
- Kullanıcı, seçtiği elemanı tasarım alanına ekleyebilmelidir.
- Aynı elemandan birden fazla eklenmesine izin verilmelidir.

Bağlantı Elemanları (İG-3)
- Kullanıcı, bağlantı elemanları arasından seçim yapabilir.
- Her bağlantı elemanı için bir özellik tablosu bulunmalıdır (etiket, renk).
- Kullanıcı, seçtiği bağlantı elemanını tasarım alanına ekleyebilmelidir.

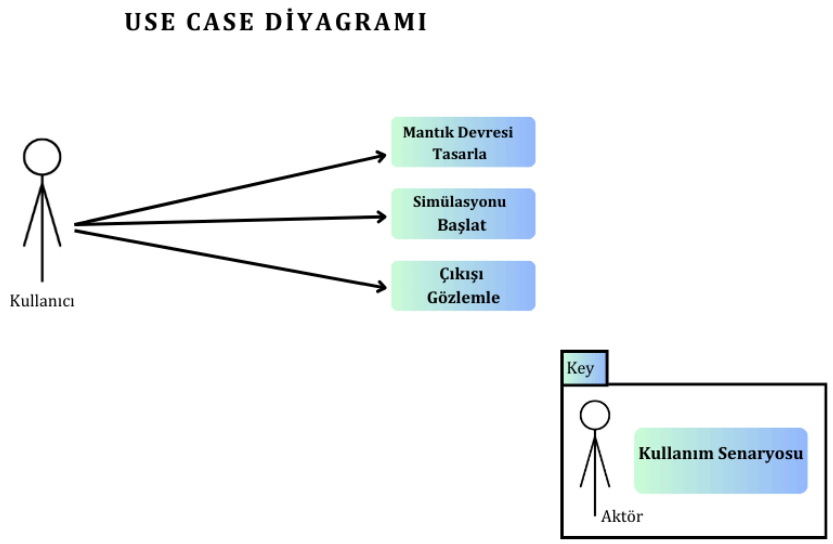
Kontrol Tuşları (İG-4)
- Tasarım işlemi sırasında çalıştır, reset ve durdur tuşları olmalıdır.
- Kullanıcı, bu tuşları kullanarak devrenin simülasyonunu kontrol edebilmelidir.

Özellik Tabloları (İG-5)
- Her bir eleman için özellik tabloları bulunmalıdır.
- Özellik tablolarındaki değerler, tasarım alanına yerleştirilen elemanlar üzerinde sağ fare tıklaması ile görüntülenebilir ve değiştirilebilir.

Bağlantılar (İG-6)
- Bağlantı hatlarının herhangi bir noktada birleştirilmesi gerekiyorsa, bağlantı düğümleri kullanılmalıdır.

Simülasyon Arayüzü (İG-7)
- Tasarım işlemi bittikten sonra, kullanıcı çalıştır, reset ve durdur tuşları ile devrenin çalışmasını simüle edebilmelidir.
- Kullanıcı, her bir elemanın giriş kutusunu seçerek giriş değerini değiştirebilir.
- Kullanıcı, devrenin çıkışında LED veya çıkış kutusu kullanabilir. Çıkış kutusu çıkış değeri gösterilmeli ve LED, devrenin çıkış değerine göre yanıp sönme özelliğine sahip olmalıdır.

2.3 Use-Case Diyagramı

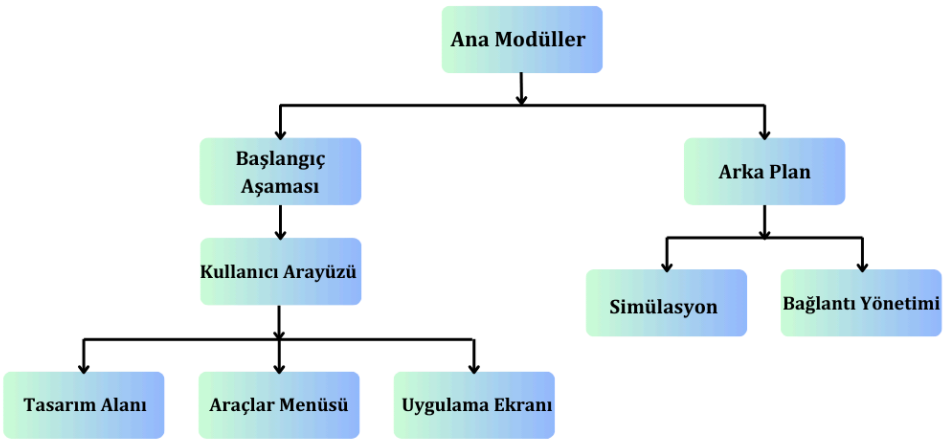


3. TASARIM

3.1 Mimari Tasarım

- Kullanıcı Arayüzü Tasarımı(UI)
  - Tasarım Alanı: Kullanıcının mantık devresini tasarlayacağı ana alan.
  - Arayüz, kullanıcı dostu ve kullanımı kolay olmalıdır.Araçlar menüsü, mantık kapıları, giriş-çıkış elemanları, bağlantı elemanları ve kontrol düğmeleri gibi araçları içerir.
- Mantık Kapıları
  - Her mantık kapısının tasarım araçları ve özelliği olacak.
  - Özellikler: Etiket (AND, OR, NOT vb.), Giriş Bağlantı Sayısı.
- Giriş-Çıkış Elemanları
  - Her elemanın özellikleri olacak.(Etiket, Renk)
  - LED'ler, Çıkış Kutusu gibi çıkış elemanları.
- Bağlantı Elemanları
  - Bağlantı hatlarını ve bağlantı noktalarını sağlar.
  - Özellikler: Etiket, Renk.
- Kontrol Düğmeleri
  - Çalıştır, Reset, Durdur gibi simülasyon kontrolü sağlayan düğmeler.
- Simülasyon
  - Kullanıcı girişlerini değiştirme ve çıkışları gözlemleme imkanı.
  - Çalıştırma, duraklatma, sıfırlama gibi işlevlerle devrenin simülasyonu.

MODÜL DİYAGRAMI

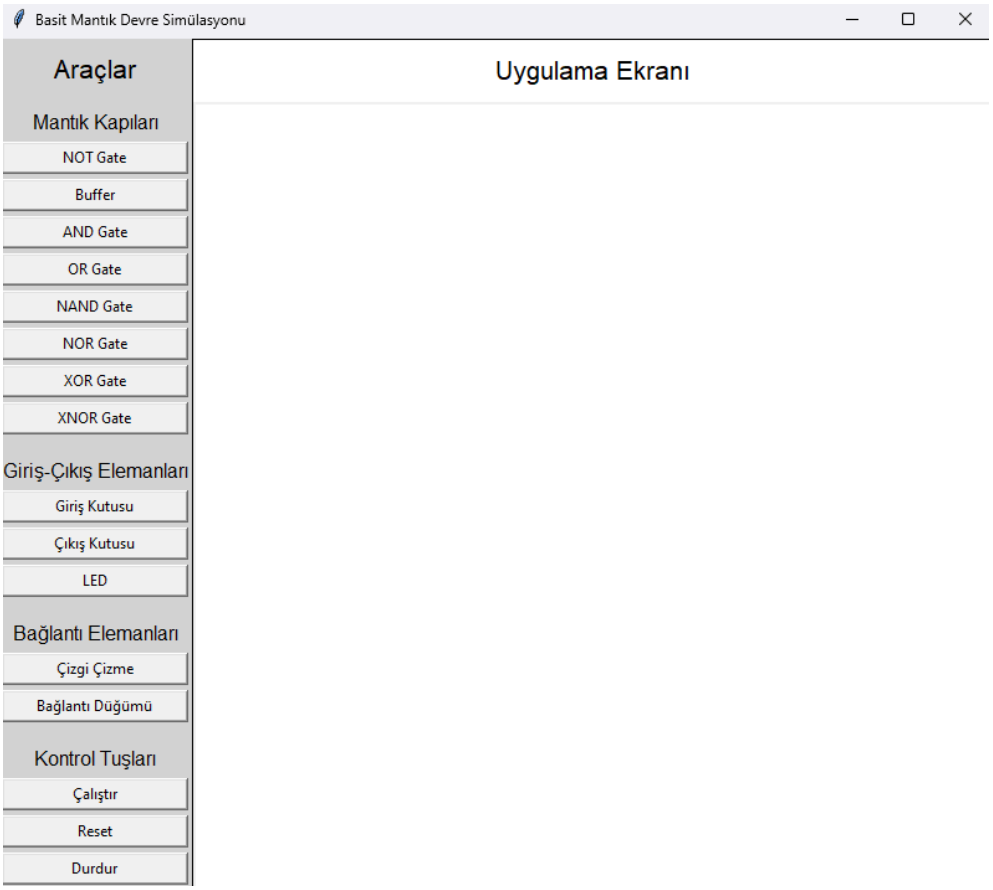


3.2 Kullanılacak Teknolojiler

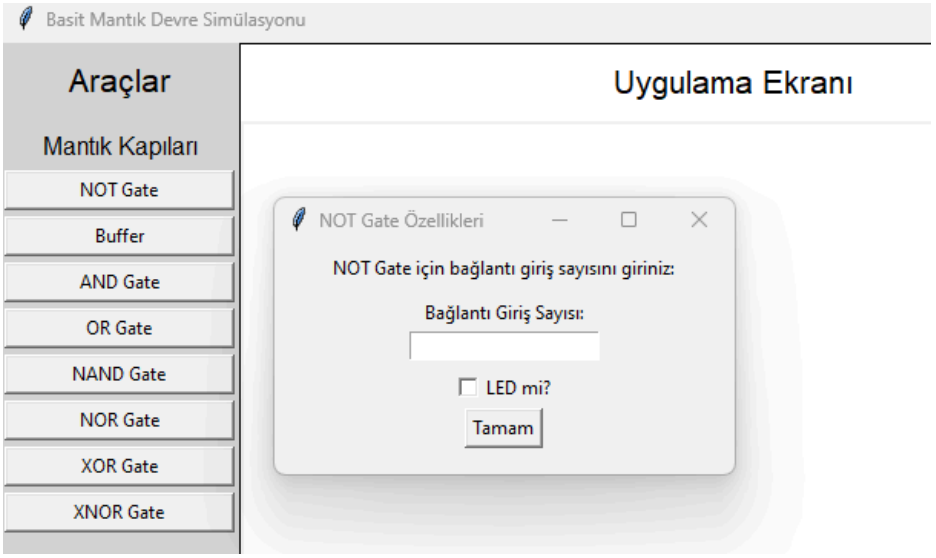
- Bu projede Python programlama dili kullanılmıştır.
- Tkinter modülü kullanılarak devre simülasyon ekranının oluşturulması sağlanmıştır.Bu simülasyon ekranı kullanılarak Basit Mantık Devre işlemlerinin gerçekleştirilmesi sağlanmıştır.
- Messagebox modülü kullanılarak giriş kutularına hatalı bir değerin girilmesi engellenmiştir.
- Tkinter modülünün Menu sınıfı kullanılarak simülasyon penceresinde yeni menülerin oluşturulması sağlanmıştır.

3.3 Kullanıcı Arayüzü Tasarımı

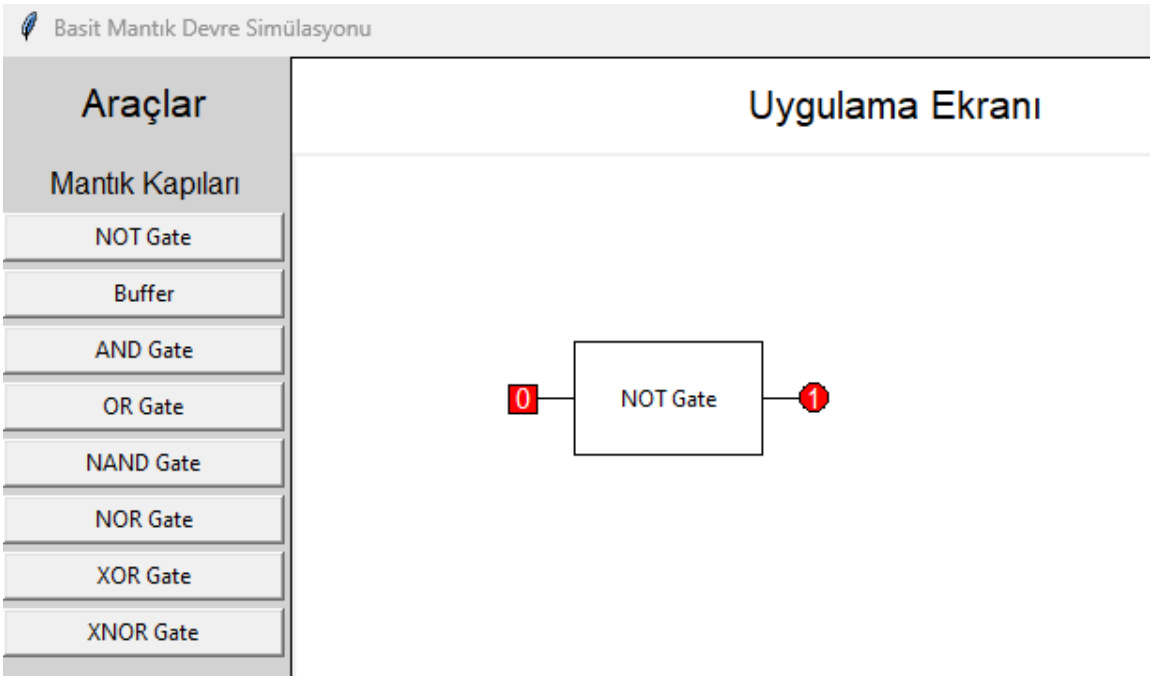
- **Ana Ekran:**Sol tarafta, araçlar paneli bulunmalıdır. Bu panelde Mantık Kapıları, Giriş/Çıkış Elemanları, Bağlantı Elemanları ve Kontrol Tuşları bulunmalıdır.Sağ tarafta tasarım alanı bulunmalıdır. Kullanıcı burada devre elemanlarını yerleştirebilir ve bağlantılarını yapabilir.
- **Araçlar Bölümü**
    - **Mantık Kapıları:**Bu alanda farklı mantık kapıları bulunmalıdır (AND, OR, NOT, vb.). Kullanıcı seçtiği kapıya ait butona tıklayarak tasarım alanına ekleyebilmelidir.
    - **Giriş/Çıkış Elemanları:**Bu alanda giriş ve çıkış elemanları bulunmalıdır (Giriş Kutusu, Çıkış Kutusu, LED). Kullanıcı seçtiği elemana ait butona tıklayarak tasarım alanına ekleyebilmelidir.
    - **Bağlantı Elemanları:** Bu alanda bağlantı elemanları bulunmalıdır (Bağlantı Düğümü, Çizgi Çizme). Kullanıcı seçtiği elemana ait butona tıklayarak tasarım alanına ekleyebilmelidir.
    - **Kontrol Tuşları:** Bu alanda çalıştır, reset ve durdur tuşları bulunmalıdır.
- **Simülasyon Ekranı:**Tasarım alanında yer alan elemanların bağlantıları ve çalışma durumu görsel olarak simüle edilir.Kullanıcı elemanların girişlerini değiştirerek devrenin çalışmasını gözlemleyebilir.Kullanıcı uygulama ekranına eklenen elemanlara sağ tıkladığında elemana ait özellikleri görüntüleyebilir. Devrenin çıkışında yer alan LED veya çıkış kutusu, devrenin çıkış değerini gösterir. LED, devrenin çıkış değerine göre yanıp söner.



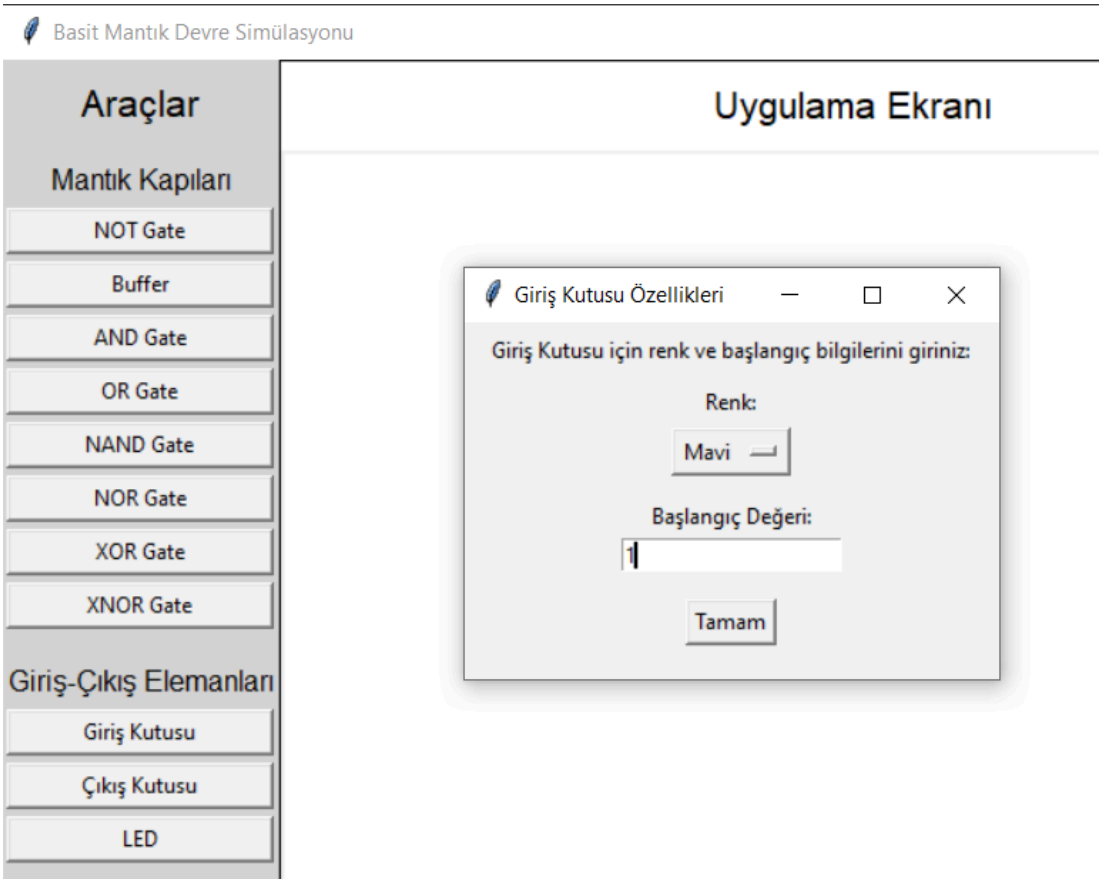
Uygulama Ekranı



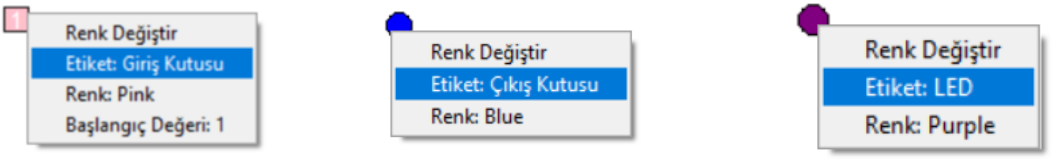
Mantık Kapıları İçin Özellik Giriş Ekranı



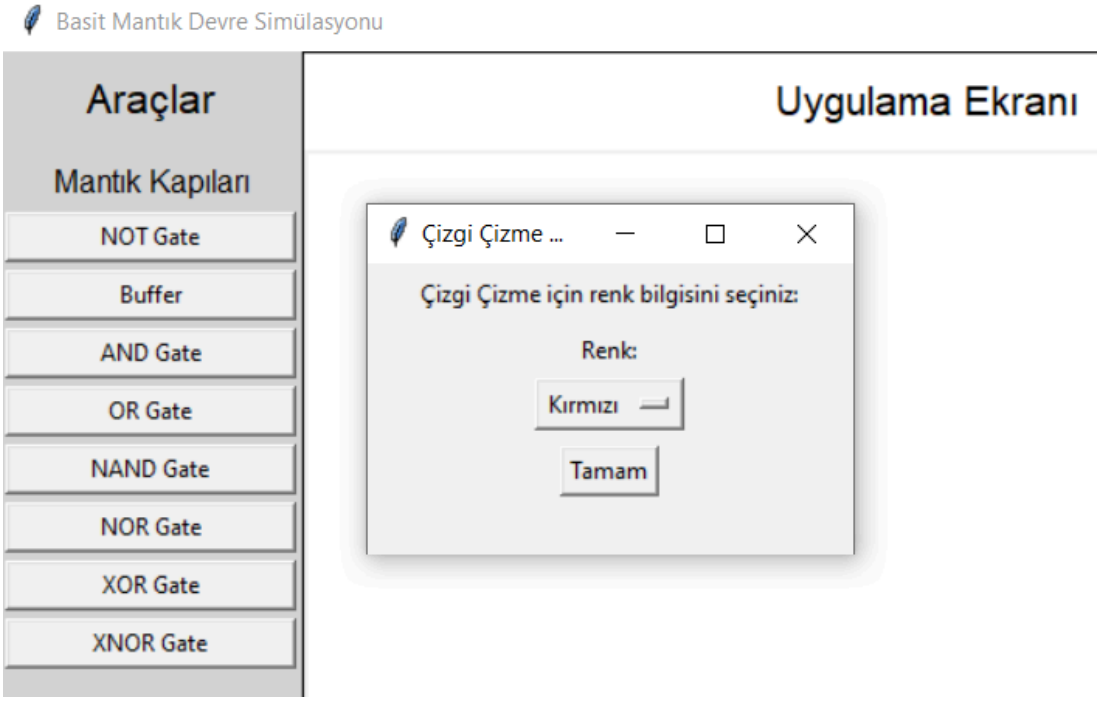
"Çalıştır" komutundan sonra örnek "Not Gate"



Giriş-Çıkış Elemanları için Özellik Giriş Ekranı



"Çalıştır" komutundan sonra örnek Giriş-Çıkış Elemanları



Bağlantı Elemanları için Özellik Giriş Ekranı

"Çalıştır" komutundan sonra örnek Bağlantı Elemanları

4. UYGULAMA
4.1 Kodlanan Bileşenlerin Açıklamaları

Bu Python kodu, tkinter kütüphanesini kullanarak basit bir mantık devre simülasyonu için bir grafik kullanıcı arayüzü (GUI) oluşturur. İlk olarak, tkinter modülü ve gerekli bileşenler (Menu ve messagebox) içe aktarılır. Ardından, global değişkenler tanımlanır: selected_gate seçilen kapı türünü, led_checkbox_var LED seçeneğini, gate_properties kapı özelliklerini, gate_instances mantık kapılarının örneklerini, ve connection_elements bağlantı elemanlarını saklamak için kullanılır. Türkçe renk isimlerini İngilizce karşılıklarıyla eşleyen bir sözlük oluşturulur (color_mapping). Son olarak, ana pencere (root) oluşturulur, başlığı "Basit Mantık Devre Simülasyonu" olarak ayarlanır ve boyutları 800x700 piksel olarak belirlenir.

```python
import tkinter as tk
from tkinter import Menu
import tkinter.messagebox as messagebox

# Global olarak seçilen kapı türünü ve özelliklerini saklayacak değişkenler
selected_gate = None
led_checkbox_var = None
gate_properties = {}
gate_instances = []
connection_elements = []

# Türkçe-İngilizce renk eşlemeleri
color_mapping = {
    "Kırmızı": "Red",
    "Yeşil": "Green",
    "Mavi": "Blue",
    "Sarı": "Yellow",
    "Pembe": "Pink",
    "Turuncu": "Orange",
    "Mor": "Purple",
    "Siyah": "Black",
    "Beyaz": "White",
    "Gri": "Gray",
    "Kahverengi": "Brown",
    "Lacivert": "Navy"
}

# Ana pencereyi oluştur
root = tk.Tk()
root.title("Basit Mantık Devre Simülasyonu")
root.geometry("800x700")
```

Sol çerçeve (left_frame), 300 piksel genişliğinde ve lightgray arka plan rengine sahip olup ana pencerenin sol tarafına yerleştirilir. Sağ çerçeve (right_frame), 500 piksel genişliğinde, beyaz arka planlı ve siyah kenarlıklı olup ana pencerenin sağ tarafına genişleyebilir şekilde yerleştirilir. Sağ çerçevede "Uygulama Ekranı" başlıklı bir etiket (app_screen_label) ve bir tuval (canvas) bulunur. Tuval, sağ çerçeve içinde genişleyebilir şekilde ayarlanmıştır. Sağ tıklama menüsü işlevi (show_properties), bir elemanın özelliklerini gösteren bir menü oluşturur ve fare sağ tıklandığında ekranda belirir.

```
# Sol çerçeve (300 px genişliğinde)
left_frame = tk.Frame(root, width=300, bg="lightgray")
left_frame.pack(side=tk.LEFT, fill=tk.Y)

# Sağ çerçeve (500 px genişliğinde)
right_frame = tk.Frame(root, width=500, bg="white", highlightbackground="black", highlightthickness=1)
right_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

# Sağ çerçevede Uygulama Ekranı başlığı
app_screen_label = tk.Label(right_frame, text="Uygulama Ekranı", bg="white", font=("Arial", 16))
app_screen_label.pack(pady=10)

# Sağ çerçevede canvas oluştur
canvas = tk.Canvas(right_frame, bg="white")
canvas.pack(fill=tk.BOTH, expand=True)

# Sağ tıklama menüsü
def show_properties(event, element_type, properties):
    menu = Menu(root, tearoff=0)
    for key, value in properties.items():
        menu.add_command(label=f"{key}: {value}")
    menu.post(event.x_root, event.y_root)
```

Fonksiyon adı create_input_box olup, belirli bir konumda (x, y) ve renk ile başlangıç değeri belirterek giriş kutusu oluşturur. Giriş kutusu, belirlenen konumda bir kare (square) ve ortasında bir metin (text) içerir. Kare ve metin, sağ tıklandığında özelliklerini gösteren bir bağlam menüsü ile donatılmıştır. Kare ve metin, fare ile sürüklenip hareket ettirilebilir ve metin çift tıklanarak "0" ile "1" arasında değiştirilebilir. Giriş kutusunun başlangıç değeri start_value ile belirlenir ve varsayılan olarak "0" olarak ayarlanmıştır. Kare ve metin, gate_instances listesine eklenir. Fonksiyon, kare ve metin öğelerini ve giriş kutusunun değerini yazdıran bir fonksiyonu döner.

```
def create_input_box(x, y, color="red", start_value="0"):
    square_size = 15
    square = canvas.create_rectangle(x, y, x + square_size, y + square_size, outline="black", fill=color)
    text_x = x + square_size / 2
    text_y = y + square_size / 2
    text = canvas.create_text(text_x, text_y, text=start_value, font=("Arial", 11), fill="white", anchor="center")

    def update_properties():
        properties = {"Etiket": "Giriş Kutusu", "Renk": canvas.itemcget(square, "fill"),
        "Başlangıç Değeri": canvas.itemcget(text, "text")}
        return properties

    gate_instances.append((square, text))

    def on_right_click(event):
        properties = update_properties()
        menu = Menu(root, tearoff=0)
        menu.add_command(label="Renk Değiştir", command=lambda: change_color("Giriş Kutusu", square))
        for key, value in properties.items():
            menu.add_command(label=f"{key}: {value}")
        menu.post(event.x_root, event.y_root)

    canvas.tag_bind(square, '<Button-3>', on_right_click)
    canvas.tag_bind(text, '<Button-3>', on_right_click)

    def start_move(event):
        canvas.scan_mark(event.x, event.y)
```

```
    def move_element(event):
        x0, y0 = canvas.coords(square)[:2]
        dx = event.x - x0
        dy = event.y - y0
        canvas.move(square, dx, dy)
        canvas.move(text, dx, dy)

    canvas.tag_bind(square, "<ButtonPress-1>", start_move)
    canvas.tag_bind(square, "<B1-Motion>", move_element)
    canvas.tag_bind(text, "<ButtonPress-1>", start_move)
    canvas.tag_bind(text, "<B1-Motion>", move_element)

    def change_value(event):
        current_value = canvas.itemcget(text, "text")
        new_value = "1" if current_value == "0" else "0"
        canvas.itemconfig(text, text=new_value)

    canvas.tag_bind(text, "<Double-1>", change_value)

    def print_value():
        print(f"Giriş Kutusu Değeri: {canvas.itemcget(text, 'text')}")

    return square, text, print_value
```

Fonksiyon adı create_output_box olup, belirli bir konumda (x, y) ve LED olup olmadığını belirleyen bir parametre ile LED veya çıkış kutusu oluşturur. Çıkış kutusu, belirlenen konumda bir oval (square) ve ortasında bir metin (text) içerir. Oval ve metin, sağ tıklandığında özelliklerini gösteren bir bağlam menüsü ile donatılmıştır. Oval ve metin, gate_instances listesine eklenir.

```
# LED ve Çıkış Kutusu'nu oluşturmak için create_output_box fonksiyonu:
def create_output_box(x, y, is_led=False):
    square_size = 15
    square = canvas.create_oval(x, y, x + square_size, y + square_size, outline="black", fill="red")
    text_x = x + square_size / 2
    text_y = y + square_size / 2
    text = canvas.create_text(text_x, text_y, text="", font=("Arial", 11), fill="white", anchor="center")

    def update_properties():
        etiket = "LED" if is_led else "Çıkış Kutusu"
        properties = {"Etiket": etiket, "Renk": canvas.itemcget(square, "fill"), "Sonuç": canvas.itemcget(text, "text")}
        return properties

    def on_right_click(event):
        properties = update_properties()
        menu = Menu(root, tearoff=0)
        menu.add_command(label="Renk Değiştir", command=lambda: change_color_output_box(square))
        for key, value in properties.items():
            menu.add_command(label=f"{key}: {value}")
        menu.post(event.x_root, event.y_root)

    canvas.tag_bind(square, '<Button-3>', on_right_click)
    canvas.tag_bind(text, '<Button-3>', on_right_click)

    return square, text
```

Fonksiyon adı change_color olup, belirli bir öğenin (item) rengini değiştirmek için bir diyalog penceresi oluşturur. Diyalog penceresi, kullanıcının bir renk seçmesine olanak tanır ve seçilen rengi öğeye uygular.

```
def change_color(element_type, item):
    color_var = tk.StringVar()
    color_var.set("Kırmızı")  # Varsayılan renk

    dialog = tk.Toplevel(root)
    dialog.title("Renk Değiştir")
    dialog.geometry("200x100")

    label = tk.Label(dialog, text=f"Yeni renk seçiniz:")
    label.pack(pady=(5, 3))

    color_menu = tk.OptionMenu(dialog, color_var, *color_mapping.keys())
    color_menu.pack(pady=(0, 5))

    def submit_color():
        color = color_mapping[color_var.get()]  # Türkçe rengi İngilizceye çevir
        canvas.itemconfig(item, fill=color)
        dialog.destroy()

    submit_button = tk.Button(dialog, text="Tamam", command=submit_color)
    submit_button.pack(pady=(1, 10))
```

item: Rengi değiştirilecek tuval öğesi (oval).color_var: Varsayılan olarak "Kırmızı" rengine ayarlanmış bir StringVar nesnesi.color_menu: Kullanıcının Türkçe renk isimlerinden birini seçmesini sağlayan bir açılır menü (OptionMenu).dialog: Ana pencerenin (root) üstünde açılan yeni bir pencere (Toplevel), başlığı "Renk Değiştir" ve boyutları 200x100 piksel.label: Diyalog penceresinde kullanıcıya yeni renk seçmesini söyleyen bir etiket. color_menu: Kullanıcının renk seçimi yapabileceği bir açılır menü.submit_button: Kullanıcı yeni rengi seçtikten sonra diyalog penceresini kapatıp rengi uygulamak için bir düğme.submit_color(): Kullanıcı düğmeye tıkladığında seçilen rengi İngilizceye çevirir ve öğeye (item) uygular (canvas.itemconfig(item, fill=color)), ardından diyalog penceresini kapatır (dialog.destroy()).Bu fonksiyon, kullanıcıya belirli bir çıkış kutusunun rengini seçme ve değiştirme olanağı sağlar.

```
def change_color_output_box(item):
    color_var = tk.StringVar()
    color_var.set("Kırmızı")  # Varsayılan renk

    dialog = tk.Toplevel(root)
    dialog.title("Renk Değiştir")
    dialog.geometry("200x100")

    label = tk.Label(dialog, text=f"Yeni renk seçiniz:")
    label.pack(pady=(5, 3))

    color_menu = tk.OptionMenu(dialog, color_var, *color_mapping.keys())
    color_menu.pack(pady=(0, 5))

    def submit_color():
        color = color_mapping[color_var.get()]  # Türkçe rengi İngilizceye çevir
        canvas.itemconfig(item, fill=color)
        dialog.destroy()

    submit_button = tk.Button(dialog, text="Tamam", command=submit_color)
    submit_button.pack(pady=(1, 10))
```

def create_gate_with_inputs(gate_name, input_count):: Kapı oluşturma işlevini tanımlar, kapının adı ve giriş sayısını alır.gate_rect = canvas.create_rectangle(gate_x, gate_y, gate_x + gate_width, gate_y + gate_height, fill="white", outline="black"): Kapıyı temsil eden dikdörtgeni oluşturur.for i in range(int(input_count)):: Belirtilen giriş sayısı kadar döner.input_box, input_text, print_value = create_input_box(input_box_x, input_box_y): Giriş kutularını oluşturur ve bunları saklar.output_line = canvas.create_line(gate_x + gate_width, gate_y + gate_height / 2, gate_x + gate_width + 20, gate_y + gate_height / 2): Kapıdan çıkan çıkış çizgisini oluşturur.gate_instances.append((gate_rect, gate_text, input_lines, output_line, input_boxes)): Oluşturulan kapı ve ilişkili nesnelerin referanslarını saklar.

```
def create_gate_with_inputs(gate_name, input_count):
    gate_width = 100
    gate_height = 60
    gate_x = 150
    gate_y = 100

    # Dikdörtgen oluştur
    gate_rect = canvas.create_rectangle(gate_x, gate_y, gate_x + gate_width, gate_y + gate_height, fill="white", outline="black")
    gate_text = canvas.create_text(gate_x + gate_width / 2, gate_y + gate_height / 2, text=gate_name)

    # Giriş çizgilerini ve giriş kutularını oluştur
    input_lines = []
    input_boxes = []  # Yeni eklenen giriş kutularını saklamak için liste
    for i in range(int(input_count)):
        line_y = gate_y + (i + 1) * (gate_height / (int(input_count) + 1))
        input_line = canvas.create_line(gate_x - 20, line_y, gate_x, line_y)
        input_lines.append(input_line)

        # Giriş kutusu oluştur ve print_value fonksiyonunu al
        input_box_x = gate_x - 35  # Gate'in solundan 35 piksel solunda
        input_box_y = line_y - 7.5  # Çizgiye göre hizalama
        input_box, input_text, print_value = create_input_box(input_box_x, input_box_y)
        input_boxes.append((input_box, input_text))

    # Çıkış çizgisini oluştur
    output_line = canvas.create_line(gate_x + gate_width, gate_y + gate_height / 2, gate_x + gate_width + 20, gate_y + gate_height / 2)

    gate_instances.append((gate_rect, gate_text, input_lines, output_line, input_boxes))
```

def on_right_click(event):: Sağ tıklamaya yanıt veren işlev.canvas.tag_bind(gate_rect, '<Button-3>', on_right_click): Dikdörtgen kapıya sağ tıklama olayı.canvas.tag_bind(gate_text, '<Button-3>', on_right_click): Kapının metin etiketine sağ tıklama olayı.def start_move(event):: Sürükleme işlemini başlatan işlev.
def move_gate(event):: Kapıyı ve bağlı girişleri sürüklemeyi sağlayan işlev.canvas.tag_bind(gate_rect, "<ButtonPress-1>", start_move): Dikdörtgen kapıya fare tıklama olayı.canvas.tag_bind(gate_rect, "<B1-Motion>", move_gate): Dikdörtgen kapıya fare sürükleme olayı.canvas.tag_bind(gate_text, "<ButtonPress-1>", start_move): Kapının metin etiketine fare tıklama olayı.canvas.tag_bind(gate_text, "<B1-Motion>", move_gate): Kapının metin etiketine fare sürükleme olayı.return print_value: Bağlantı değerlerini içeren değişkeni döndürür.

```
    def on_right_click(event):
        properties = {"Etiket": gate_name, "Bağlantı Giriş Sayısı": input_count}
        show_properties(event, gate_name, properties)

    canvas.tag_bind(gate_rect, '<Button-3>', on_right_click)
    canvas.tag_bind(gate_text, '<Button-3>', on_right_click)

    def start_move(event):
        canvas.scan_mark(event.x, event.y)

    def move_gate(event):
        x, y = canvas.coords(gate_rect)[:2]
        dx = event.x - x
        dy = event.y - y
        canvas.move(gate_rect, dx, dy)
        canvas.move(gate_text, dx, dy)
        for line in input_lines:
            canvas.move(line, dx, dy)
        for box, text in input_boxes:
            canvas.move(box, dx, dy)
            canvas.move(text, dx, dy)
        canvas.move(output_line, dx, dy)

    canvas.tag_bind(gate_rect, "<ButtonPress-1>", start_move)
    canvas.tag_bind(gate_rect, "<B1-Motion>", move_gate)
    canvas.tag_bind(gate_text, "<ButtonPress-1>", start_move)
    canvas.tag_bind(gate_text, "<B1-Motion>", move_gate)

    return print_value
```

def create_io_image(element_type, label, color=None, start_value=None):: Giriş kutusu veya LED oluşturan bir fonksiyon tanımlar.if element_type == "Çıkış Kutusu" or element_type == "LED":: Giriş tipine bağlı olarak farklı işlemler yapar.Giriş tipi "Çıkış Kutusu" veya "LED" ise, yuvarlak bir şekil oluşturur ve rengini ayarlar. def on_right_click(event):: Sağ tıklamaya yanıt veren bir işlev tanımlar. Renk değiştirmek için bir menü görüntüler.def start_move(event): ve def move_element(event):: Sürükleme işlemini başlatan ve sürüklemeyi sağlayan işlevler tanımlar.Diğer durumda, kare bir şekil ve bir metin etiketi oluşturur. Bu kare, giriş kutusunu temsil eder.def on_right_click(event):: Sağ tıklamaya yanıt veren bir işlev tanımlar. Renk değiştirmek için bir menü görüntüler.def start_move(event): ve def move_element(event):: Sürükleme işlemini başlatan ve sürüklemeyi sağlayan işlevler tanımlar.def change_value(event):: Çift tıklamaya yanıt veren bir işlev tanımlar. Metin etiketinin değerini değiştirir.

```python
# Giriş kutusu oluşturma ve sağ tıklama işlevselliği
def create_io_image(element_type, label, color=None, start_value=None):
    if element_type == "Çıkış Kutusu" or element_type == "LED":
        radius = 7.5
        x, y = 50, 50
        circle = canvas.create_oval(x, y, x + 2 * radius, y + 2 * radius, outline="black", fill=color)

        def update_properties():
            properties = {"Etiket": element_type, "Renk": canvas.itemcget(circle, "fill")}
            return properties

        gate_instances.append((circle,))

        def on_right_click(event):
            properties = update_properties()
            menu = Menu(root, tearoff=0)
            menu.add_command(label="Renk Değiştir", command=lambda: change_color(element_type, circle))
            for key, value in properties.items():
                menu.add_command(label=f"{key}: {value}")
            menu.post(event.x_root, event.y_root)

        canvas.tag_bind(circle, '<Button-3>', on_right_click)

        def start_move(event):
            canvas.scan_mark(event.x, event.y)

        def move_element(event):
            x0, y0, _, _ = canvas.coords(circle)
            dx = event.x - x0
            dy = event.y - y0
            canvas.move(circle, dx, dy)

        canvas.tag_bind(circle, "<ButtonPress-1>", start_move)
        canvas.tag_bind(circle, "<B1-Motion>", move_element)
```

```
else:
    square_size = 15
    x, y = 100, 100
    square = canvas.create_rectangle(x, y, x + square_size, y + square_size, outline="black", fill=color)
    text_x = x + square_size / 2
    text_y = y + square_size / 2
    text = canvas.create_text(text_x, text_y, text=start_value, font=("Arial", 11), fill="white", anchor="center")

    def update_properties():
        properties = {"Etiket": element_type, "Renk": canvas.itemcget(square, "fill"), "Başlang□ç Değeri": canvas.itemcget(text, "text")}
        return properties

    gate_instances.append((square, text))

    def on_right_click(event):
        properties = update_properties()
        menu = Menu(root, tearoff=0)
        menu.add_command(label="Renk Değiştir", command=lambda: change_color(element_type, square))
        for key, value in properties.items():
            menu.add_command(label=f"{key}: {value}")
        menu.post(event.x_root, event.y_root)

    canvas.tag_bind(square, '<Button-3>', on_right_click)
    canvas.tag_bind(text, '<Button-3>', on_right_click)

    def start_move(event):
        canvas.scan_mark(event.x, event.y)

    def move_element(event):
        x0, y0 = canvas.coords(square)[:2]
        dx = event.x - x0
        dy = event.y - y0
        canvas.move(square, dx, dy)
```

```
    canvas.tag_bind(square, "<ButtonPress-1>", start_move)
    canvas.tag_bind(square, "<B1-Motion>", move_element)
    canvas.tag_bind(text, "<ButtonPress-1>", start_move)
    canvas.tag_bind(text, "<B1-Motion>", move_element)

    def change_value(event):
        current_value = canvas.itemcget(text, "text")
        new_value = "1" if current_value == "0" else "0"
        canvas.itemconfig(text, text=new_value)

    canvas.tag_bind(text, "<Double-1>", change_value)

    def start_move(event):
        canvas.scan_mark(event.x, event.y)

    def move_element(event):
        x0, y0 = canvas.coords(square)[:2]
        dx = event.x - x0
        dy = event.y - y0
        canvas.move(square, dx, dy)
        canvas.move(text, dx, dy)

    canvas.tag_bind(square, "<ButtonPress-1>", start_move)
    canvas.tag_bind(square, "<B1-Motion>", move_element)
    canvas.tag_bind(text, "<ButtonPress-1>", start_move)
    canvas.tag_bind(text, "<B1-Motion>", move_element)

    def change_value(event):
        current_value = canvas.itemcget(text, "text")
        new_value = "1" if current_value == "0" else "0"
        canvas.itemconfig(text, text=new_value)

    canvas.tag_bind(text, "<Double-1>", change_value)
```

def create_connection_line(color, label):: Bağlantı çizgisi oluşturan bir fonksiyon tanımlar.start_x, start_y = None, None: Başlangıç noktası için değişkenler tanımlar.line = canvas.create_line(0, 0, 0, 0, fill=color, width=2, tags=label): Başlangıçta görünmez bir çizgi oluşturur.def start_draw(event):: Çizim işlemini başlatan bir işlev tanımlar. Başlangıç noktasını günceller.def draw_line(event):: Çizimi gerçekleştiren bir işlev tanımlar. Çizgiyi fare hareketine göre günceller.def update_properties():: Özellikleri güncelleyen bir işlev tanımlar. Şu anki çizginin rengini döndürür.def on_right_click(event):: Sağ tıklamaya yanıt veren bir işlev tanımlar. Renk değiştirme seçeneği sunar.canvas.bind("<Button-1>", start_draw): Fare tıklama olayına çizimi başlatan işlevi bağlar.canvas.bind("<B1-Motion>", draw_line): Fare sürükleme olayına çizimi gerçekleştiren işlevi bağlar. canvas.tag_bind(line, '<Button-3>', on_right_click): Çizgiye sağ tıklama olayı ekler ve sağ tıklamaya yanıt veren işlevi bağlar.def end_draw(event):: Çizim işlemini sonlandıran bir işlev tanımlar. Olay bağlantılarını kaldırır. canvas.bind("<ButtonRelease-1>", end_draw): Fare tıklama bırakma olayına çizimi sonlandıran işlevi bağlar.

```
def create_connection_line(color, label):
    # Çizginin başlangıç ve bitiş noktalarının koordinatları
    start_x, start_y = None, None
    line = canvas.create_line(0, 0, 0, 0, fill=color, width=2, tags=label)

    # Başlangıç noktasını güncelleyen fonksiyon
    def start_draw(event):
        nonlocal start_x, start_y
        start_x, start_y = event.x, event.y

    def draw_line(event):
        nonlocal start_x, start_y
        end_x, end_y = event.x, event.y
        canvas.coords(line, start_x, start_y, end_x, end_y)

    def update_properties():
        properties = {"Etiket": "Çizgi Çizme", "Renk": canvas.itemcget(line, "fill")}
        return properties
```

```
    def on_right_click(event):
        properties = update_properties()
        menu = Menu(root, tearoff=0)
        menu.add_command(label="Renk Değiştir", command=lambda: change_color("Çizgi Çizme", line))
        for key, value in properties.items():
            menu.add_command(label=f"{key}: {value}")
        menu.post(event.x_root, event.y_root)
        canvas.unbind("<Button-1>")
        canvas.unbind("<B1-Motion>")

    canvas.bind("<Button-1>", start_draw)
    canvas.bind("<B1-Motion>", draw_line)

    canvas.tag_bind(line, '<Button-3>', on_right_click)

    def end_draw(event):
        canvas.unbind("<Button-1>")
        canvas.unbind("<B1-Motion>")
        canvas.unbind("<ButtonRelease-1>")

    canvas.bind("<ButtonRelease-1>", end_draw)
```

def create_connection_node(color):: Bağlantı düğümü oluşturan bir fonksiyon tanımlar.x1, y1 = 50, 50 ve x2, y2 = 65, 65: Dikdörtgenin köşelerinin koordinatlarını belirler.points = [x1 + round_radius, y1, x2 - round_radius, y1, x2, y1 + round_radius, x2, y2 - round_radius, x2 - round_radius, y2, x1 + round_radius, y2, x1, y2 - round_radius, x1, y1 + round_radius]: Yuvarlatılmış köşeleri hesaplar.node = canvas.create_polygon(points, outline="black", fill=color): Dikdörtgen düğümü oluşturur.def start_move(event): ve def move_node(event):: Sürükleme işlemini başlatan ve düğümü sürüklemeyi sağlayan işlevler tanımlar.def update_properties():: Özellikleri güncelleyen bir işlev tanımlar. Şu anki düğümün rengini döndürür.def on_right_click(event):: Sağ tıklamaya yanıt veren bir işlev tanımlar. Renk değiştirme seçeneği sunar.canvas.tag_bind(node, '<Button-3>', on_right_click): Düğüme sağ tıklama olayı ekler ve sağ tıklamaya yanıt veren işlevi bağlar.

```python
def create_connection_node(color):
    x1, y1 = 50, 50
    x2, y2 = 65, 65
    round_radius = 3  # Yumuşatma yarıçapı

    # Dikdörtgenin köşelerini yuvarlatmak için köşelerdeki noktaları hesapla
    points = [x1 + round_radius, y1,x2 - round_radius, y1,x2, y1 + round_radius,x2, y2
    - round_radius,x2 - round_radius, y2,x1 + round_radius, y2,x1, y2 - round_radius,x1, y1 + round_radius]

    node = canvas.create_polygon(points, outline="black", fill=color)
    connection_elements.append(node)

    def start_move(event):
        canvas.scan_mark(event.x, event.y)

    def move_node(event):
        x, y = canvas.coords(node)[:2]
        canvas.move(node, event.x - x, event.y - y)

    canvas.tag_bind(node, "<ButtonPress-1>", start_move)
    canvas.tag_bind(node, "<B1-Motion>", move_node)

    def update_properties():
        properties = {"Etiket": "Bağlantı Düğümü", "Renk": canvas.itemcget(node, "fill")}
        return properties

    def on_right_click(event):
        properties = update_properties()
        menu = Menu(root, tearoff=0)
        menu.add_command(label="Renk Değiştir", command=lambda: change_color("Bağlantı Düğümü", node))
        for key, value in properties.items():
            menu.add_command(label=f"{key}: {value}")
        menu.post(event.x_root, event.y_root)

    canvas.tag_bind(node, '<Button-3>', on_right_click)
```

def open_gate_properties_dialog(gate_name):: Kapı özelliklerini girdiği bir diyalog penceresi oluşturan bir fonksiyon tanımlar.dialog = tk.Toplevel(root): Kök pencereden ayrı olarak bir diyalog penceresi oluşturur. label = tk.Label(dialog, text=f"{gate_name} için bağlantı giriş sayısını giriniz:"): Kapı adını içeren bir metin etiketi oluşturur.input_count_label = tk.Label(dialog, text="Bağlantı Giriş Sayısı:"): Bağlantı giriş sayısını belirten bir metin etiketi oluşturur.input_count_entry = tk.Entry(dialog): Bağlantı giriş sayısının girilebileceği bir giriş kutusu oluşturur.led_checkbox_var = tk.BooleanVar(): LED seçeneğinin durumunu depolamak için bir Boolean değişkeni tanımlar.led_checkbox = tk.Checkbutton(dialog, text="LED mi?", variable=led_ checkbox _var): LED seçeneğini temsil eden bir onay kutusu oluşturur.def submit_properties():: Özelliklerin gönderilmesinden sorumlu bir işlev tanımlar.submit_button = tk.Button(dialog, text="Tamam", command= submit_properties): Özelliklerin gönderilmesini sağlayan bir düğme oluşturur.def show_gate_properties(event):: Kapı özelliklerini gösteren bir işlev tanımlar.dialog.bind('<Button-3>', show_gate_properties): Sağ tıklamaya yanıt olarak kapı özelliklerini gösteren işlevi bağlar.

```python
# Kapı özellikleri diyalogları
def open_gate_properties_dialog(gate_name):
    global led_checkbox_var
    dialog = tk.Toplevel(root)
    dialog.title(f"{gate_name} Özellikleri")
    dialog.geometry("300x150")

    label = tk.Label(dialog, text=f"{gate_name} için bağlantı giriş sayısını giriniz:")
    label.pack(pady=(5, 3))

    input_count_label = tk.Label(dialog, text="Bağlantı Giriş Sayısı:")
    input_count_label.pack(pady=(5, 2))
    input_count_entry = tk.Entry(dialog)
    input_count_entry.pack(pady=(0, 5))

    led_checkbox_var = tk.BooleanVar()
    led_checkbox = tk.Checkbutton(dialog, text="LED mi?", variable=led_checkbox_var)
    led_checkbox.pack()

    def submit_properties():
        input_count = input_count_entry.get()
        create_gate_with_inputs(gate_name, input_count)
        dialog.destroy()

    submit_button = tk.Button(dialog, text="Tamam", command=submit_properties)
    submit_button.pack(pady=(1, 10))

    def show_gate_properties(event):
        properties = {"Etiket": gate_name, "Bağlantı Giriş Sayısı": input_count_entry.get()}
        show_properties(event, gate_name, properties)

    dialog.bind('<Button-3>', show_gate_properties)
```

def open_io_properties_dialog(element_type):: Giriş veya çıkış elemanı için özelliklerin girildiği bir diyalog penceresi oluşturan bir fonksiyon tanımlar.dialog = tk.Toplevel(root): Kök pencereden ayrı bir diyalog penceresi oluşturur.color_var = tk.StringVar(dialog): Renk seçeneğini depolamak için bir StringVar oluşturur. color_menu = tk.OptionMenu(dialog, color_var, *color_mapping.keys()): Renk seçimini sağlayan bir menü oluşturur.if element_type == "Giriş Kutusu":: Eleman tipine göre farklı etiketler ve giriş kutuları oluşturur. def submit_properties():: Özelliklerin gönderilmesinden sorumlu bir işlev tanımlar.if start_value not in ["0", "1"]:: Başlangıç değerinin geçerliliğini kontrol eder.submit_button = tk.Button(dialog, text="Tamam", command=submit_properties): Özelliklerin gönderilmesini sağlayan bir düğme oluşturur.

```python
def open_io_properties_dialog(element_type):
    dialog = tk.Toplevel(root)
    dialog.title(f"{element_type} Özellikleri")
    dialog.geometry("300x200")
    if element_type == "Giriş Kutusu":
        label = tk.Label(dialog, text=f"{element_type} için renk ve başlangıç bilgilerini giriniz:")
        label.pack(pady=(5, 3))
    else:
        label = tk.Label(dialog, text=f"{element_type} için renk bilgisini seçiniz:")
        label.pack(pady=(5, 3))

    color_label = tk.Label(dialog, text="Renk:")
    color_label.pack(pady=(5, 2))
    color_var = tk.StringVar(dialog)
    color_var.set("Kırmızı")  # Varsayılan renk
    color_menu = tk.OptionMenu(dialog, color_var, *color_mapping.keys())
    color_menu.pack(pady=(0, 5))

    if element_type == "Giriş Kutusu":
        start_value_label = tk.Label(dialog, text="Başlangıç Değeri:")
        start_value_label.pack(pady=(5, 2))
        start_value_entry = tk.Entry(dialog)
        start_value_entry.pack(pady=(0, 5))

    def submit_properties():
        color = color_mapping[color_var.get()]  # Türkçe rengi İngilizceye çevir
        if element_type == "Giriş Kutusu":
            start_value = start_value_entry.get()
            if start_value not in ["0", "1"]:
                messagebox.showerror("Hata", "Başlangıç değeri yalnızca '0' veya '1' olabilir.")
                return
            create_io_image(element_type, element_type, color, start_value)
        else:
            create_io_image(element_type, element_type, color)
        dialog.destroy()

    submit_button = tk.Button(dialog, text="Tamam", command=submit_properties)
    submit_button.pack(pady=(10, 10))
```

def open_connection_properties_dialog(connection_type):: Bağlantı tipi için özelliklerin girildiği bir diyalog penceresi oluşturan bir fonksiyon tanımlar.dialog = tk.Toplevel(root): Kök pencereden ayrı bir diyalog penceresi oluşturur.colors = ["Kırmızı", "Yeşil", ...]: Renk seçimini sağlamak için kullanılacak renk listesini oluşturur. color_var = tk.StringVar(dialog): Renk seçeneğini depolamak için bir StringVar oluşturur.color_menu = tk.OptionMenu(dialog, color_var, *colors): Renk seçimini sağlayan bir menü oluşturur.def submit_properties():: Özelliklerin gönderilmesinden sorumlu bir işlev tanımlar.color = color_mapping[color_var.get()]: Seçilen rengi İngilizceye çevirir.if connection_type == "Çizgi Çizme": ve elif connection_type == "Bağlantı Düğümü":: Bağlantı tipine göre farklı işlevler çağırır ve pencereyi kapatır.

```python
def open_connection_properties_dialog(connection_type):
    dialog = tk.Toplevel(root)
    dialog.title(f"{connection_type} Özellikleri")
    dialog.geometry("250x150")

    label = tk.Label(dialog, text=f"{connection_type} için renk bilgisini seçiniz:")
    label.pack(pady=(5, 3))

    color_label = tk.Label(dialog, text="Renk:")
    color_label.pack(pady=(5, 2))

    colors = ["Kırmızı", "Yeşil", "Mavi", "Sarı", "Pembe", "Turuncu", "Mor",
    "Siyah", "Beyaz", "Gri", "Kahverengi", "Lacivert"]
    color_var = tk.StringVar(dialog)
    color_var.set(colors[0])
    color_menu = tk.OptionMenu(dialog, color_var, *colors)
    color_menu.pack(pady=(0, 5))

    def submit_properties():
        color = color_mapping[color_var.get()]  # Türkçe rengi İngilizceye çevir
        if connection_type == "Çizgi Çizme":
            create_connection_line(color, connection_elements)
        elif connection_type == "Bağlantı Düğümü":
            create_connection_node(color)
        dialog.destroy()
        dialog.destroy()

    submit_button = tk.Button(dialog, text="Tamam", command=submit_properties)
    submit_button.pack(pady=(1, 10))
```

def run_simulation():: Simülasyonu çalıştıran bir fonksiyon tanımlar.for instance in gate_instances:: Tüm kapı örnekleri üzerinde döner.if len(instance) == 5:: Kapı örneğinin doğru bir biçimde oluşturulduğunu kontrol eder. if "NAND Gate" in canvas.itemcget(instance[1], 'text'):: NAND kapısını kontrol eder.input_boxes = instance[4]: Giriş kutularını alır.input_values = [canvas.itemcget(text, "text") for _, text in input_boxes]: Giriş değerlerini alır.input_values = list(map(int, input_values)): Giriş değerlerini tamsayıya dönüştürür.and_result = all (input_values): Tüm girişlerin mantıksal ve sonucunu alır.nand_result = not and_result: NAND kapısının çıkışını hesaplar.is_led = led_checkbox_var.get() == 1: LED seçeneğini kontrol eder.output_square, output_text = create_output_box(gate_x + 120, gate_y + 22, is_led=is_led): Çıkış kutusunu veya LED'i oluşturur.if is_led:: Eğer LED seçeneği işaretlenmişse:if nand_result == 0:: NAND kapısının çıkışını kontrol eder ve LED rengini ayarlar.else:: LED seçeneği işaretlenmemişse:output_value = "1" if nand_result else "0": Çıkış değerini ayarlar. canvas.itemconfig(output_text, text=output_value): Çıkış kutusuna değeri yazar.canvas.itemconfig(output _square, fill="red"): Çıkış kutusunun rengini ayarlar.output_line = instance[3]: Çıkış hattını alır.canvas.coords (output_line, gate_x + 100, gate_y + 30, gate_x + 120, gate_y + 30): Çıkış hattının koordinatlarını günceller.

```python
def run_simulation():
    global led_checkbox_var
    for instance in gate_instances:
        if len(instance) == 5:  # Gate'ler
            if "NAND Gate" in canvas.itemcget(instance[1], 'text'):
                input_boxes = instance[4]
                input_values = [canvas.itemcget(text, "text") for _, text in input_boxes]
                input_values = list(map(int, input_values))  # String değerleri int'e çevir

                and_result = all(input_values)
                nand_result = not and_result

                # Çıkış kutusunu veya LED'i oluştur
                gate_x, gate_y = canvas.coords(instance[0])[:2]
                is_led = led_checkbox_var.get() == 1  # LED seçiliyse True olacak
                output_square, output_text = create_output_box(gate_x + 120, gate_y + 22, is_led=is_led)

                if is_led:
                    canvas.itemconfig(output_text, text="")
                    if nand_result == 0:
                        canvas.itemconfig(output_square, fill="#fa8072")
                    else:
                        canvas.itemconfig(output_square, fill="red")
                else:
                    output_value = "1" if nand_result else "0"
                    canvas.itemconfig(output_text, text=output_value)
                    canvas.itemconfig(output_square, fill="red")

                # Bağlantı çizgisi oluştur
                output_line = instance[3]
                canvas.coords(output_line, gate_x + 100, gate_y + 30, gate_x + 120, gate_y + 30)
```

```python
elif "AND Gate" in canvas.itemcget(instance[1], 'text'):
    input_boxes = instance[4]
    input_values = [canvas.itemcget(text, "text") for _, text in input_boxes]
    input_values = list(map(int, input_values))  # String değerleri int'e çevir

    # AND gate işlemi
    result = all(input_values)

    # Çıkış kutusunu oluştur
    gate_x, gate_y = canvas.coords(instance[0])[:2]
    is_led = led_checkbox_var.get() == 1  # LED seçiliyse True olacak
    output_square, output_text = create_output_box(gate_x + 120, gate_y + 22, is_led=is_led)

    if is_led:
        canvas.itemconfig(output_text, text="")
        if result == 0:
            canvas.itemconfig(output_square, fill="#fa8072")
        else:
            canvas.itemconfig(output_square, fill="red")
    else:
        output_value = "1" if result else "0"
        canvas.itemconfig(output_text, text=output_value)
        canvas.itemconfig(output_square, fill="red")

    # Bağlantı çizgisi oluştur
    output_line = instance[3]
    canvas.coords(output_line, gate_x + 100, gate_y + 30, gate_x + 120, gate_y + 30)
```

```
elif "XNOR Gate" in canvas.itemcget(instance[1], 'text'):
    input_boxes = instance[4]
    input_values = [canvas.itemcget(text, "text") for _, text in input_boxes]
    input_values = list(map(int, input_values))  # String değerleri int'e çevir

    # XNOR kapısı işlemi
    xor_result = input_values[0] ^ input_values[1]
    result = not xor_result

    # Çıkış kutusunu oluştur
    gate_x, gate_y = canvas.coords(instance[0])[:2]
    is_led = led_checkbox_var.get() == 1  # LED seçiliyse True olacak
    output_square, output_text = create_output_box(gate_x + 120, gate_y + 22, is_led=is_led)

    if is_led:
        canvas.itemconfig(output_text, text="")
        if result == 0:
            canvas.itemconfig(output_square, fill="#fa8072")
        else:
            canvas.itemconfig(output_square, fill="red")
    else:
        output_value = "1" if result else "0"
        canvas.itemconfig(output_text, text=output_value)
        canvas.itemconfig(output_square, fill="red")

    # Bağlantı çizgisi oluştur
    output_line = instance[3]
    canvas.coords(output_line, gate_x + 100, gate_y + 30, gate_x + 120, gate_y + 30)
```

```
elif "NOR Gate" in canvas.itemcget(instance[1], 'text'):
    input_boxes = instance[4]
    input_values = [canvas.itemcget(text, "text") for _, text in input_boxes]
    input_values = list(map(int, input_values))  # String değerleri int'e çevir

    # NOR gate işlemi
    notgate = any(input_values)
    result = not notgate

    # Çıkış kutusunu oluştur
    gate_x, gate_y = canvas.coords(instance[0])[:2]
    is_led = led_checkbox_var.get() == 1  # LED seçiliyse True olacak
    output_square, output_text = create_output_box(gate_x + 120, gate_y + 22, is_led=is_led)

    if is_led:
        canvas.itemconfig(output_text, text="")
        if result == 0:
            canvas.itemconfig(output_square, fill="#fa8072")
        else:
            canvas.itemconfig(output_square, fill="red")
    else:
        output_value = "1" if result else "0"
        canvas.itemconfig(output_text, text=output_value)
        canvas.itemconfig(output_square, fill="red")

    # Bağlantı çizgisi oluştur
    output_line = instance[3]
    canvas.coords(output_line, gate_x + 100, gate_y + 30, gate_x + 120, gate_y + 30)
```

```
elif "XOR Gate" in canvas.itemcget(instance[1], 'text'):
    input_boxes = instance[4]
    input_values = [canvas.itemcget(text, "text") for _, text in input_boxes]
    input_values = list(map(int, input_values))  # String değerleri int'e çevir

    # XOR kapısı işlemi
    result = input_values[0] ^ input_values[1]

    # Çıkış kutusunu oluştur
    gate_x, gate_y = canvas.coords(instance[0])[:2]
    is_led = led_checkbox_var.get() == 1  # LED seçiliyse True olacak
    output_square, output_text = create_output_box(gate_x + 120, gate_y + 22, is_led=is_led)

    if is_led:
        canvas.itemconfig(output_text, text="")
        if result == 0:
            canvas.itemconfig(output_square, fill="#fa8072")
        else:
            canvas.itemconfig(output_square, fill="red")
    else:
        output_value = "1" if result else "0"
        canvas.itemconfig(output_text, text=output_value)
        canvas.itemconfig(output_square, fill="red")

    # Bağlantı çizgisi oluştur
    output_line = instance[3]
    canvas.coords(output_line, gate_x + 100, gate_y + 30, gate_x + 120, gate_y + 30)
```

```
elif "OR Gate" in canvas.itemcget(instance[1], 'text'):
    input_boxes = instance[4]
    input_values = [canvas.itemcget(text, "text") for _, text in input_boxes]
    input_values = list(map(int, input_values))  # String değerleri int'e çevir

    # OR gate işlemi
    result = any(input_values)

    # Çıkış kutusunu oluştur
    gate_x, gate_y = canvas.coords(instance[0])[:2]
    is_led = led_checkbox_var.get() == 1  # LED seçiliyse True olacak
    output_square, output_text = create_output_box(gate_x + 120, gate_y + 22, is_led=is_led)

    if is_led:
        canvas.itemconfig(output_text, text="")
        if result == 0:
            canvas.itemconfig(output_square, fill="#fa8072")
        else:
            canvas.itemconfig(output_square, fill="red")
    else:
        output_value = "1" if result else "0"
        canvas.itemconfig(output_text, text=output_value)
        canvas.itemconfig(output_square, fill="red")

    # Bağlantı çizgisi oluştur
    output_line = instance[3]
    canvas.coords(output_line, gate_x + 100, gate_y + 30, gate_x + 120, gate_y + 30)
```

```
elif "NOT Gate" in canvas.itemcget(instance[1], 'text'):
    input_boxes = instance[4]
    input_values = [canvas.itemcget(text, "text") for _, text in input_boxes]
    input_values = list(map(int, input_values))  # String değerleri int'e çevir

    # NOT gate işlemi
    result = not input_values[0]

    # Çıkış kutusunu oluştur
    gate_x, gate_y = canvas.coords(instance[0])[:2]
    is_led = led_checkbox_var.get() == 1  # LED seçiliyse True olacak
    output_square, output_text = create_output_box(gate_x + 120, gate_y + 22, is_led=is_led)

    if is_led:
        canvas.itemconfig(output_text, text="")
        if result == 0:
            canvas.itemconfig(output_square, fill="#fa8072")
        else:
            canvas.itemconfig(output_square, fill="red")
    else:
        output_value = "1" if result else "0"
        canvas.itemconfig(output_text, text=output_value)
        canvas.itemconfig(output_square, fill="red")

    # Bağlantı çizgisi oluştur
    output_line = instance[3]
    canvas.coords(output_line, gate_x + 100, gate_y + 30, gate_x + 120, gate_y + 30)
```

```
elif "Buffer" in canvas.itemcget(instance[1], 'text'):
    input_boxes = instance[4]
    input_values = [canvas.itemcget(text, "text") for _, text in input_boxes]
    input_values = list(map(int, input_values))  # String değerleri int'e çevir

    # BUFFER gate işlemi
    result = input_values[0]

    # Çıkış kutusunu oluştur
    gate_x, gate_y = canvas.coords(instance[0])[:2]
    is_led = led_checkbox_var.get() == 1  # LED seçiliyse True olacak
    output_square, output_text = create_output_box(gate_x + 120, gate_y + 22, is_led=is_led)

    if is_led:
        canvas.itemconfig(output_text, text="")
        if result == 0:
            canvas.itemconfig(output_square, fill="#fa8072")
        else:
            canvas.itemconfig(output_square, fill="red")
    else:
        output_value = "1" if result else "0"
        canvas.itemconfig(output_text, text=output_value)
        canvas.itemconfig(output_square, fill="red")

    # Bağlantı çizgisi oluştur
    output_line = instance[3]
    canvas.coords(output_line, gate_x + 100, gate_y + 30, gate_x + 120, gate_y + 30)
```

def reset():: Uygulama ekranındaki tüm araçları silebilecek bir fonksiyon tanımlar.canvas.delete("all"): Tuvaldeki tüm öğeleri siler.gate_instances.clear(): Kapı örneklerinin listesini temizler.connection_ elements.clear(): Bağlantı öğelerinin listesini temizler.def stop():: Tüm kapıların çıkış kutularını silebilecek bir fonksiyon tanımlar.for instance in gate_instances:: Tüm kapı örnekleri için döngü oluşturur.if len(instance) == 5:: Kapı örneğinin doğru bir şekilde oluşturulduğunu kontrol eder.gate_x, gate_y = canvas.coords(instance[0])[:2]: Kapının koordinatlarını alır.overlapping_items = canvas.find_overlapping(gate_x + 200, gate_y, gate_x + 130, gate_y + 30): Kapının çıkış kutusu ile çakışan öğeleri bulur.for item in overlapping_items:: Çakışan öğeler için döngü oluşturur.if item != instance[0]:: Öğenin kapı örneği olmadığından emin olur.canvas.delete(item): Öğeyi siler.

```python
#Uygulama ekranında yer alan tüm araçları silen fonksiyon
def reset():
    canvas.delete("all")
    gate_instances.clear()
    connection_elements.clear()


# Tüm kapıların çıkış kutularını silen fonksiyon
def stop():
    for instance in gate_instances:
        if len(instance) == 5:  # Kapılar
            gate_x, gate_y = canvas.coords(instance[0])[:2]
            overlapping_items = canvas.find_overlapping(gate_x + 200, gate_y, gate_x + 130, gate_y + 30)
            for item in overlapping_items:
                if item != instance[0]:
                    canvas.delete(item)
```

tools_label: Araçlar bölümünü tanımlar.logic_gates_label: Mantık kapıları bölümünü tanımlar.
logic_gates_buttons: Mantık kapıları için butonları tanımlar.io_elements_label: Giriş-çıkış elemanları bölümünü tanımlar.io_elements_buttons: Giriş-çıkış elemanları için butonları tanımlar.connection_elements_label: Bağlantı elemanları bölümünü tanımlar.connection_elements_buttons: Bağlantı elemanları için butonları tanımlar.
control_buttons_label: Kontrol düğmeleri bölümünü tanımlar.control_buttons: Kontrol düğmelerini tanımlar.
root.mainloop(): Arayüzün sürekli olarak çalışmasını sağlar. Kullanıcı etkileşimde bulunabilir.

```python
# Sol çerçevede Araçlar başlığı
tools_label = tk.Label(left_frame, text="Araçlar", bg="lightgray", font=("Arial", 16))
tools_label.pack(pady=(10, 0))

# Mantık Kapıları
logic_gates_label = tk.Label(left_frame, text="Mant□k Kap□lar□", bg="lightgray", font=("Arial", 12))
logic_gates_label.pack(pady=(15, 2))

# Mantık Kapıları için butonlar
not_gate_button = tk.Button(left_frame, text="NOT Gate", width=20, command=lambda: open_gate_properties_dialog("NOT Gate"))
not_gate_button.pack(pady=2)

buffer_button = tk.Button(left_frame, text="Buffer", width=20, command=lambda: open_gate_properties_dialog("Buffer"))
buffer_button.pack(pady=2)

and_gate_button = tk.Button(left_frame, text="AND Gate", width=20, command=lambda: open_gate_properties_dialog("AND Gate"))
and_gate_button.pack(pady=2)

or_gate_button = tk.Button(left_frame, text="OR Gate", width=20, command=lambda: open_gate_properties_dialog("OR Gate"))
or_gate_button.pack(pady=2)

nand_gate_button = tk.Button(left_frame, text="NAND Gate", width=20, command=lambda: open_gate_properties_dialog("NAND Gate"))
nand_gate_button.pack(pady=2)

nor_gate_button = tk.Button(left_frame, text="NOR Gate", width=20, command=lambda: open_gate_properties_dialog("NOR Gate"))
nor_gate_button.pack(pady=2)

xor_gate_button = tk.Button(left_frame, text="XOR Gate", width=20, command=lambda: open_gate_properties_dialog("XOR Gate"))
xor_gate_button.pack(pady=2)

xnor_gate_button = tk.Button(left_frame, text="XNOR Gate", width=20, command=lambda: open_gate_properties_dialog("XNOR Gate"))
xnor_gate_button.pack(pady=2)
```

```
# Giriş-Çıkış Elemanları
io_elements_label = tk.Label(left_frame, text="Giriş-Çıkış Elemanları", bg="lightgray", font=("Arial", 12))
io_elements_label.pack(pady=(15,2))

# Giriş-Çıkış Elemanları için butonlar
input_box_button = tk.Button(left_frame, text="Giriş Kutusu", width=20, command=lambda: open_io_properties_dialog("Giriş Kutusu"))
input_box_button.pack(pady=2)

output_box_button = tk.Button(left_frame, text="Çıkış Kutusu", width=20, command=lambda: open_io_properties_dialog("Çıkış Kutusu"))
output_box_button.pack(pady=2)

led_button = tk.Button(left_frame, text="LED", width=20, command=lambda: open_io_properties_dialog("LED"))
led_button.pack(pady=2)

# Bağlantı Elemanları
connection_elements_label = tk.Label(left_frame, text="Bağlantı Elemanları", bg="lightgray", font=("Arial", 12))
connection_elements_label.pack(pady=(15,2))

# Bağlantı Elemanları için butonlar
draw_line_button = tk.Button(left_frame, text="Çizgi Çizme", width=20, command=lambda: open_connection_properties_dialog("Çizgi Çizme"))
draw_line_button.pack(pady=2)

connection_node_button = tk.Button(left_frame, text="Bağlantı Düğümü", width=20, command=lambda: open_connection_properties_dialog("Bağlantı Düğümü"))
connection_node_button.pack(pady=2)
```

```
# Kontrol Tuşları
control_buttons_label = tk.Label(left_frame, text="Kontrol Tuşları", bg="lightgray", font=("Arial", 12))
control_buttons_label.pack(pady=(15, 2))

# Kontrol Tuşları için butonlar
run_button = tk.Button(left_frame, text="Çalıştır", width=20, command=run_simulation)
run_button.pack(pady=2)

reset_button = tk.Button(left_frame, text="Reset", width=20, command=reset)
reset_button.pack(pady=2)

stop_button = tk.Button(left_frame, text="Durdur", width=20, command=stop)
stop_button.pack(pady=2)

# Mainloop
root.mainloop()
```

4.2 Görev Dağılımı

Projenin Kodlanması: Senem ADALAN, Sema Su YILMAZ
Raporun Hazırlanması: Senem ADALAN, Sema Su YILMAZ
Test Kodunun Yazılması: Senem ADALAN

4.3 Karşılaşılan Zorluklar ve Çözüm Yöntemleri

Bu projeyi geliştirirken birçok farklı alanda zorlandığımızı söyleyebiliriz.Devre temelleri ile ilgili çok bilgimiz olmadığından ödevi anlamak ve istenenler için algoritma geliştirmek bizi oldukça zorladı.Yaptığımız araştırmalar sayesinde verilen ödevi daha iyi bir şekilde anlayarak algoritmayı geliştirebildik.Yazdığımız devre simülasyonu Tkinter modülü hakkında kapsamlı bilgi sahibi olmamızı gerektiriyordu.Bunun için uzun araştırmalar yapmamız gerekti ancak Tkinter modülü ile ilgili detaylı bilgilere erişebildik.Ödevde kullanılan kapılar, giriş-çıkış elemanlarının işleyişini tam olarak kavrayamamıştık.Bu devre elemanlarını daha iyi anlayabilmek adına Yazılım Mühendisliğine Giriş ders notlarımızı kullandık.

4.4 Proje İsterlerine Göre Eksik Yönler

Proje isterleri tamamlanmıştır ancak aynı anda birden fazla gate ile işlem yapılması doğrultusunda geliştirilebilir.

5. TEST VE DOĞRULAMA
5.1 Yazılımın Test Süreci

Bu birim testi,"create_gate_with_inputs" adlı bir fonksiyonun doğru çalışıp çalışmadığını kontrol etmek için kullanılır.Fonksiyon, farklı mantıksal kapılar oluşturmayı ve bu kapıların giriş değerlerini yazdıran başka bir fonksiyon döndürmesi gerektiğini test eder. Her test, "assert" ifadesi kullanılarak kontrol edilir. Eğer bir test başarısız olursa, bir hata mesajı görüntülenir. Aksi takdirde, test başarılı olduğunda bir başarı mesajı görüntülenir. Son olarak, "test_create_gate_with_inputs()" fonksiyonu çağrılarak tüm testlerin çalıştırılması sağlanır.

```
def test_create_gate_with_inputs():
    # NOT gate test
    not_gate_print_value = create_gate_with_inputs("NOT Gate", 1)
    assert callable(not_gate_print_value), "Giriş kutusu değerini yazdıran fonksiyon oluşturulamadı!"
    print("Gate Oluşturma Testi Başarılı!")

    # Buffer gate test
    buffer_gate_print_value = create_gate_with_inputs("Buffer", 1)
    assert callable(buffer_gate_print_value), "Giriş kutusu değerini yazdıran fonksiyon oluşturulamadı!"
    print("Gate Oluşturma Testi Başarılı!")

    # AND gate test
    and_gate_print_value = create_gate_with_inputs("AND Gate", 2)
    assert callable(and_gate_print_value), "Giriş kutusu değerini yazdıran fonksiyon oluşturulamadı!"
    print("Gate Oluşturma Testi Başarılı!")

    # OR gate test
    or_gate_print_value = create_gate_with_inputs("OR Gate", 2)
    assert callable(or_gate_print_value), "Giriş kutusu değerini yazdıran fonksiyon oluşturulamadı!"
    print("Gate Oluşturma Testi Başarılı!")
```

```
    #NAND gate test
    nand_gate_print_value = create_gate_with_inputs("NAND Gate", 2)
    assert callable(nand_gate_print_value), "Giriş kutusu değerini yazdıran fonksiyon oluşturulamadı!"
    print("Gate Oluşturma Testi Başarılı!")

    #NOR gate test
    nor_gate_print_value = create_gate_with_inputs("NOR Gate", 2)
    assert callable(nor_gate_print_value), "Giriş kutusu değerini yazdıran fonksiyon oluşturulamadı!"
    print("Gate Oluşturma Testi Başarılı!")

    # XOR gate test
    xor_gate_print_value = create_gate_with_inputs("XOR Gate", 2)
    assert callable(xor_gate_print_value), "Giriş kutusu değerini yazdıran fonksiyon oluşturulamadı!"
    print("Gate Oluşturma Testi Başarılı!")

    #XNOR gate test
    xnor_gate_print_value = create_gate_with_inputs("XNOR Gate", 2)
    assert callable(xnor_gate_print_value), "Giriş kutusu değerini yazdıran fonksiyon oluşturulamadı!"
    print("Gate Oluşturma Testi Başarılı!")
test_create_gate_with_inputs()
```

Bu birim testi mantıksal kapıların doğru bir şekilde çalışıp çalışmadığını kontrol etmek için kullanılır. İlk olarak, ilgili mantıksal kapı create_gate_with_inputs fonksiyonu kullanılarak oluşturulur. Sonra, giriş değerleri create_io_image fonksiyonu kullanılarak belirlenir.Ardından, simülasyon çalıştırılır. Son olarak, elde edilen çıktı değerleri görüntülenir.Bu testler, farklı mantıksal kapılar için ayrı ayrı gerçekleştirilir. Her bir test, beklenen çıktı değerleri ile karşılaştırılarak doğruluğu kontrol edilir. Eğer bir test başarısız olursa, bir hata mesajı görüntülenir.Tüm testlerin tamamlanmasının ardından, test_simulation() fonksiyonu çağrılarak tüm testlerin çalıştırılması sağlanır. Bu şekilde, programın beklenen şekilde çalışıp çalışmadığı kontrol edilir.

```
def test_simulation():
    # NOT kapısı testi
    create_gate_with_inputs("NOT Gate", 1)  # 1 girişli NOT kapısı oluştur
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "1")  # Giriş kutusu
    print("NOT Kapısı Testi:")
    run_simulation()
    print("\n")

    #Buffer kapısı testi
    create_gate_with_inputs("Buffer Gate", 1)  # 1 girişli NOT kapısı oluştur
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "1")  # Giriş kutusu
    print("Buffer Kapısı Testi:")
    run_simulation()
    print("\n")

    # AND kapısı testi
    create_gate_with_inputs("AND Gate", 2)  # 2 girişli AND kapısı oluştur
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "1")  # İlk giriş kutusu
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "0")  # İkinci giriş kutusu
    print("AND Kapısı Testi:")
    run_simulation()
    print("\n")

    #OR kapısı testi
    create_gate_with_inputs("OR Gate", 2)  # 2 girişli OR kapısı oluştur
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "1")  # İlk giriş kutusu
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "0")  # İkinci giriş kutusu
    print("OR Kapısı Testi:")
    run_simulation()
    print("\n")
```

```
    # NAND kapısı testi
    create_gate_with_inputs("NAND Gate", 2)  # 2 girişli NAND kapısı oluştur
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "1")  # İlk giriş kutusu
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "0")  # İkinci giriş kutusu
    print("NAND Kapısı Testi:")
    run_simulation()
    print("\n")

    # NOR kapısı testi
    create_gate_with_inputs("NOR Gate", 2)  # 2 girişli NOR kapısı oluştur
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "1")  # İlk giriş kutusu
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "0")  # İkinci giriş kutusu
    print("NOR Kapısı Testi:")
    run_simulation()
    print("\n")

    # XOR kapısı testi
    create_gate_with_inputs("XOR Gate", 2)  # 2 girişli XOR kapısı oluştur
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "1")  # İlk giriş kutusu
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "0")  # İkinci giriş kutusu
    print("XOR Kapısı Testi:")
    run_simulation()
    print("\n")

    #XNOR kapısı testi
    create_gate_with_inputs("XNOR Gate", 2)  # 2 girişli XOR kapısı oluştur
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "1")  # İlk giriş kutusu
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "red", "0")  # İkinci giriş kutusu
    print("XNOR Kapısı Testi:")
    run_simulation()
    print("\n")
test_simulation()
```

Bu kod giriş/çıkış bileşenlerinin oluşturulması için testler içerir. Bu testler, create_io_image adlı fonksiyonun doğru çalışıp çalışmadığını kontrol eder. **Giriş Kutusu Oluşturma Testi:** Bu adımda, bir giriş kutusu bileşeni oluşturulur. create_io_image fonksiyonu kullanılarak giriş kutusu oluşturulur. Oluşturulurken, bileşenin adı ("Giriş Kutusu"), etiketi ("Giriş Kutusu"), rengi ("Red") ve varsayılan değeri ("0") belirtilir.**Çıkış Kutusu Oluşturma Testi:** Bu adımda, bir çıkış kutusu bileşeni oluşturulur. create_io_image fonksiyonu kullanılarak çıkış kutusu oluşturulur. Bu kez, bileşenin adı ("Çıkış Kutusu"), etiketi ("Çıkış Kutusu") ve rengi ("Green") belirtilir. **LED Oluşturma Testi:** Bu adımda, bir LED bileşeni oluşturulur. create_io_image fonksiyonu kullanılarak LED bileşeni oluşturulur. Bileşenin adı ("LED"), etiketi ("LED") ve rengi ("Purple") belirtilir.Bu testler, her bir bileşenin doğru bir şekilde oluşturulup oluşturulmadığını kontrol eder. Testler, test_create_io() fonksiyonu çağrılarak çalıştırılır. Her bir test, bileşenin doğru bir şekilde oluşturulduğunu göstermek için görüntülenir.

```
def test_create_io():
    # Giriş Kutusu Oluşturma Testi
    create_io_image("Giriş Kutusu", "Giriş Kutusu", "Red", "0")

    # Çıkış Kutusu Oluşturma Testi
    create_io_image("Çıkış Kutusu", "Çıkış Kutusu", "Green")

    #LED Oluşturma Testi
    create_io_image("LED", "LED", "Purple")
test_create_io()
```

Bu kod bağlantı bileşenlerinin oluşturulması için testleri içerir. Bu testler, create_connection_line ve create_connection_node adlı fonksiyonların doğru çalışıp çalışmadığını kontrol eder.**Çizgi Çizme Testi:** Bu adımda, bir çizgi bağlantısı oluşturulur. create_connection_line fonksiyonu kullanılarak bir çizgi çizilir. Çizginin rengi ("Black") belirtilir. Bu test, çizginin doğru bir şekilde çizilip çizilmediğini kontrol eder.**Bağlantı Düğümü Oluşturma Testi:** Bu adımda, bir bağlantı düğümü oluşturulur. create_connection_node fonksiyonu kullanılarak bir bağlantı düğümü oluşturulur. Düğümün rengi ("Blue") belirtilir. Bu test, bağlantı düğümünün doğru bir şekilde oluşturulup oluşturulmadığını kontrol eder.Her bir test, ilgili bileşenin doğru bir şekilde oluşturulduğunu göstermek için bir mesaj görüntüler. Tüm testler, test_create_connection() fonksiyonu çağrılarak çalıştırılır.

```
def test_create_connection():
    # Çizgi Çizme Testi
    create_connection_line("Black", "Çizgi Çizme")

    # Bağlantı Düğümü Oluşturma Testi
    create_connection_node("Blue")
test_create_connection()
```

Bu kod, bir giriş kutusu bileşeninin değerinin değiştirilmesini ve bu değişikliğin nasıl gerçekleştiğini test eder. İlk olarak, create_input_box fonksiyonu kullanılarak bir giriş kutusu oluşturulur ve bu kutunun başlangıç değeri "0" olarak ayarlanır. Daha sonra, canvas.itemconfig fonksiyonu aracılığıyla giriş kutusunun içindeki metin nesnesinin değeri "1" olarak değiştirilir. Bu değişiklik sonrasında, giriş kutusunun yeni değeri "1" olarak doğru bir şekilde yansıtılıp yansıtılmadığını kontrol etmek için "print_value" fonksiyonu çağrılır. Bu test, giriş kutusunun değerinin güncellenmesini ve grafik arayüzünde doğru bir şekilde gösterilmesini doğrular.

```
def test_box_value_change():
    # Giriş kutusu oluşturma ve değerini değiştirme testi
    input_box, input_text, print_value = create_input_box(200, 200, color="blue", start_value="0")
    print_value()  # Başlangıçta değer "0" olarak ayarlandı, bu nedenle çıktı "0" olmalı

    # Değerini değiştirme
    canvas.itemconfig(input_text, text="1")
    print_value()  # Değer "1" olarak değiştirildi, çıktı "1" olmalı
test_box_value_change()
```

5.2 Yazılımın Doğrulanması

- **Mantık Kapısı Oluşturma Testi:**Mantık kapıları için oluşturulan fonksiyonların doğru çalışıp çalışmadığını kontrol eder. Her bir mantık kapısı için bir giriş kutusu değeri yazdıran fonksiyon oluşturulup oluşturulmadığına bakar.
- **Simülasyon Testi:**Simülasyonun doğru çalışıp çalışmadığını kontrol eder. Her bir mantık kapısı için simülasyonun doğru sonuçlar üretip üretmediğini kontrol eder.
- **Giriş/Çıkış Elemanı Oluşturma Testi:**Giriş/çıkış elemanlarının doğru şekilde oluşturulup oluşturulmadığını kontrol eder. Giriş kutusu, çıkış kutusu ve LED'in doğru renklerde oluşturulduğunu test eder.
- **Bağlantı Elemanı Oluşturma Testi:**Bağlantı hatlarının ve bağlantı düğümlerinin doğru şekilde oluşturulup oluşturulmadığını kontrol eder. Çizgi çizme ve bağlantı düğümü oluşturma işlevlerinin doğru renklerde ve doğru şekillerde çalışıp çalışmadığını test eder.
- **Giriş Kutusu Değer Değiştirme Testi:**Giriş kutusu değerinin doğru şekilde değiştirilip değiştirilemediğini kontrol eder. Başlangıçta belirlenen değerin doğru olduğunu ve değerin değiştirildiğinde yeni değerin doğru şekilde yansıtıldığını test eder.

Bu testler, her bir işlemin doğru şekilde gerçekleştirildiğini doğrular. Her bir test, ilgili işlemleri başarıyla tamamladığında bir başarı mesajı yazdırır, aksi takdirde bir hata mesajı yazdırır. Bu şekilde, yazılımın beklenen davranışlarını sergileyip sergilemediği test edilir.

6. GİTHUB LİNKLERİ

7. KAYNAKÇA

https://www.technopat.net/sosyal/konu/en-iyi-4-python-gui-kuetuephanesi.2187532/
https://teslaakademi.com/mantik-devreleri
https://electronics.stackexchange.com/questions/439199/logisim-input-and-terminal-issue
https://www.geeksforgeeks.org/logic-gates-in-python/
https://www.youtube.com/watch?v=W7luvtXeQTA
https://stackoverflow.com/questions/20640418/logical-nand-of-two-variables-in-python
https://www.youtube.com/watch?v=PUpQMHQXPs4