GİRİŞ

Bu proje, OpenCV kullanarak mouse ile kontrol edilen bir “tavşan”ın ekranda rastgele yönde hareket eden bir “havuç”u yakalamasına dayalı basit, etkileşimli bir mini oyun geliştirmeyi amaçlar. Oyunun temel hedefi, fareyi kullanarak tavşanı havuca temas ettirmek ve her yakalamada skoru artırmaktır.

GELİŞME

[1]:
```
%matplotlib qt
```

[2]:
```
import cv2
import numpy as np
import math
import random
```

[3]:
```
# Ekran genişliği ve yüksekliği ayarlama (piksel cinsinden)
W, H = 800, 600                 # ekran boyutu

# Arka plan rengi (B,G,R)
BGCOLOR = (25, 25, 25)

# Havuç ve tavşan çizimleri için yarıçap değerleri (piksel cinsinden)
HAVUC_Y = 16                    # havuç yarıçapı (çarpışma için)
TAVSAN_Y = 22                   # tavşan yarıçapı (çarpışma için)

# Havuç hızının rastgele seçileceği aralık (piksel / kare)
SPEED_MIN, SPEED_MAX = 3.0, 7.0 # havuç hız aralığı (px/frame)

# Ekrana yazı yazarken kullanılacak font
FONT = cv2.FONT_HERSHEY_SIMPLEX

# Havucun konumu, yönü, hızının rastgeleliği için
random.seed()

# Farenin konumu (başlangıçta ekranın ortası olarak ayarlama)
mouse_x, mouse_y = W / 2, H / 2
```

İlk olarak cv2, numpy, math ve random kütüphaneleri içe aktarılır. Pencere boyutları (W=800, H=600) ve koyu bir arka plan oluşturulur. Havuç ve tavşan için çarpışma hesaplarında kullanılacak yarıçap değerleri (HAVUC_Y=16, TAVSAN_Y=22) belirlenir, bu değerler hem görsel boyutu hem de çarpışmayı etkiler. Oyunun dinamik bir yapıda olması için havuç hareketi için minimum ve maksimum hızlar (SPEED_MIN=3.0, SPEED_MAX=7.0) seçilir. Ekrana bilgilendirici yazılar yazmak için OpenCV’nin FONT_HERSHEY_SIMPLEX fontu kullanılır. Fare konumu başlangıçta ekran merkezine yerleştirilir; bu sayede oyun açılsa bile tavşanın konumu bilinir bir noktadan başlar.

[4]:
```
# Fare koordinatları için callback fonksiyonu
def mouse_callback(event, x, y, flags, param):
    # Fonksiyon içinde global değişkenleri değiştireceğimizi belirtme
    global mouse_x, mouse_y
    # Eğer event "MOUSEMOVE" ise, güncel fare koordinatlarını kaydetme
    if event == cv2.EVENT_MOUSEMOVE:
        mouse_x, mouse_y = x, y
```

mouse_callback fonksiyonu, **cv2.EVENT_MOUSEMOVE** olayını yakalayarak fare koordinatlarını sürekli olarak günceller. Bu sayede tavşan, her framede herhangi ek hesaplama gerektirmeden fareyi “takip ediyor” gibi davranır. Global değişkenlerin

kullanılması, calback içinden anlık değer güncellemeyi kolaylaştırır ve ana döngüde tavşanın pozisyonu doğrudan bu değerlerden okunur.

[5]:
```python
# Yeni oluşacak havucun konumu (rastgele konum + rastgele yönde rastgele hız)
def random_havuc():
    # Havucun rastgele konumu
    x = random.randint(HAVUC_Y , W - HAVUC_Y )
    y = random.randint(HAVUC_Y , H - HAVUC_Y )
     # 0 ile 2*pi arasında rastgele bir açı seçme, havucun herhangi bir yöne doğru gidebilmesi için
    angle = random.uniform(0, 2 * math.pi)
    # Hız büyüklüğünü rastgele seçme
    speed = random.uniform(SPEED_MIN, SPEED_MAX)
    # Açıyı hız vektörüne dönüştürme (cos, sin)
    vx = math.cos(angle) * speed # x eksenindeki sağa/sola hız bileşeni
    vy = math.sin(angle) * speed # y eksenindeki yukarı/aşağı  hız bileşeni
    # Konumu ve hızı (float) olarak döndürme
    return [float(x), float(y)], [vx, vy]
```

random_havuc fonksiyonu, ekranda rastgele bir başlangıç konumu üretir ve havuca rasgele bir yön ile hız büyüklüğü atar. Yön, 0- 2pi aralığından seçilen bir açıyla belirlenir; hız ise tanımlanan aralıktan doğrusal rastgelelikle alınır. Açı, hız vektörüne kosinüs ve sinüs bileşenleriyle dönüştürülür. Böylece her yeni havuç, farklı bir doğrultuda ve hızda hareket eder.

[6]:
```python
# Havuç çizimi
def havuc_ciz(img, pos):
    # Konumu tamsayıya çevirme
    x, y = int(pos[0]), int(pos[1])
    #Turuncu gövdesi için
    cv2.circle(img, (x, y), HAVUC_Y, (0, 140, 255), -1)  # gövdesinin turuncu rengi  olması için (0, 140, 255)
    cv2.line(img, (x, y - HAVUC_Y), (x, y - HAVUC_Y - 14), (0, 200, 0), 3) #Başlangıç noktası: (x, y - HAVUC_Y) Havuç
    #Yeşil sapı için
    cv2.line(img, (x - 6, y - HAVUC_Y + 2), (x - 12, y - HAVUC_Y - 8), (0, 180, 0), 2)
    cv2.line(img, (x + 6, y - HAVUC_Y + 2), (x + 12, y - HAVUC_Y - 8), (0, 180, 0), 2)
```

[7]:
```python
# Tavşan çizimi ( Kulakları,başı gövdesi,gözleri,ağzı)
def tavsan_ciz(img, pos):
     # Konumu tamsayıya çevirme
    x, y = int(pos[0]), int(pos[1])
    #Kulakları
    cv2.ellipse(img, (x - 10, y - 28), (7, 18), 0, 0, 360, (245, 245, 245), -1)
    cv2.ellipse(img, (x + 10, y - 28), (7, 18), 0, 0, 360, (245, 245, 245), -1)
    # Baş / gövde
    cv2.circle(img, (x, y), TAVSAN_Y, (255, 255, 255), -1)
    #Gözler
    cv2.circle(img, (x - 7, y - 5), 3, (0, 0, 0), -1)
    cv2.circle(img, (x + 7, y - 5), 3, (0, 0, 0), -1)
    #Burun ve ağız
    cv2.circle(img, (x, y + 3), 2, (0, 0, 255), -1)
    cv2.line(img, (x - 6, y + 10), (x - 1, y + 8), (0, 0, 0), 1)
    cv2.line(img, (x + 6, y + 10), (x + 1, y + 8), (0, 0, 0), 1)
```

Havuç ve tavşan çizmi iki ayrı fonksiyonla yapılmıştır .”**havuc_ciz**”, havucun turuncu gövdesini bir daire ile, yeşil saplarını ise birkaç kısa çizgi ile resmeder. Dairenin yarıçapı, çarpışma hesabıyla uyumlu olacak şekilde HAVUC_Y üzerinden belirlenir. “**tavsan_ciz**”, tavşanın kulaklarını elipslerle, baş ve gövdesini tek bir daire ile, göz ve burun/ağız detaylarını küçük daire ve çizgi şeklinde çizer. Her iki fonksiyon da piksel konumlarını tamsayıya yuvarlar; bu, OpenCV çizim fonksiyonlarının beklediği türle uyumluluk ve net görüntü sağlar.

```
[8]: # İki nokta arası Öklidyen mesafe, Tavşan ile havucun çarpışıp çarpışmadığını kontrol etmek için:
def mesafe(a, b):
    dx = a[0] - b[0]
    dy = a[1] - b[1]
    return math.hypot(dx, dy)
```

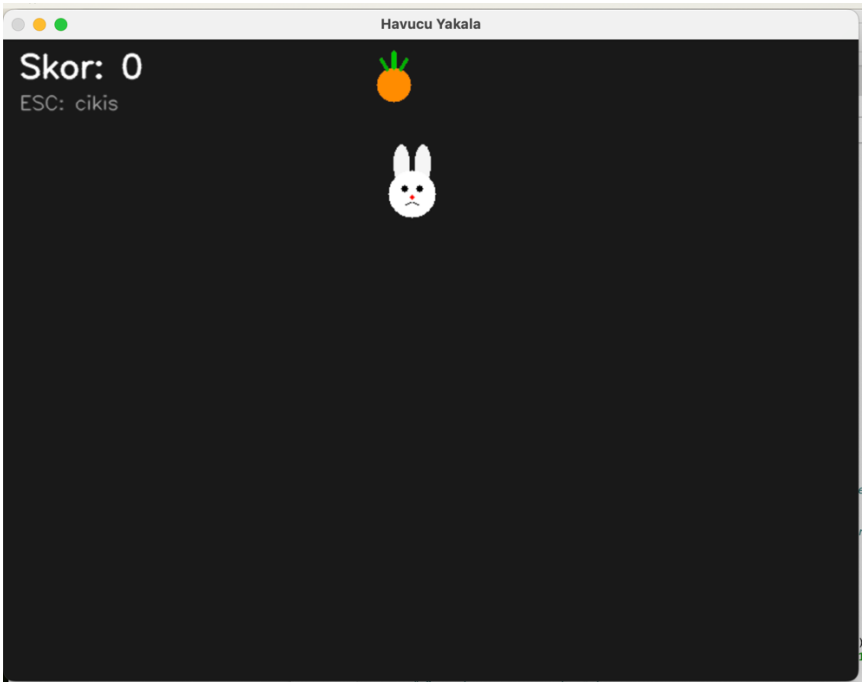
İki nokta arasındaki Öklidyen uzaklık math.hypot ile hesaplanır. Tavşan merkezi ile havuç merkezi arasındaki mesafe, iki yarıçapın toplamına göre kıyaslanır.

```
def main():
    # Pencere oluşturma
    cv2.namedWindow("Havucu Yakala")
    # Fare hareketlerini yakalayacak callback i bağlama
    cv2.setMouseCallback("Havucu Yakala", mouse_callback)

    # İlk havucun konumu ve hızı
    havuc_pos, havuc_vel = random_havuc()
    skor = 0

    # Her karede ekranı güncelleme
    while True:
        # Arka plan
        frame = np.full((H, W, 3), BGCOLOR, dtype=np.uint8)

        # Havuç konumunu hızı kadar değiştirmr
        havuc_pos[0] += havuc_vel[0]
        havuc_pos[1] += havuc_vel[1]

        # Kenarlardan sekme
        if havuc_pos[0] <= HAVUC_Y or havuc_pos[0] >= W - HAVUC_Y:
            havuc_vel[0] *= -1
            havuc_pos[0] = np.clip(havuc_pos[0], HAVUC_Y, W - HAVUC_Y)
        if havuc_pos[1] <= HAVUC_Y or havuc_pos[1] >= H - HAVUC_Y:
            havuc_vel[1] *= -1
            havuc_pos[1] = np.clip(havuc_pos[1], HAVUC_Y, H - HAVUC_Y)

        # Çizimler
        havuc_ciz(frame, havuc_pos)
        tavsan_ciz(frame, (mouse_x, mouse_y))

        # Çarpışma kontrolü
        d = mesafe((mouse_x, mouse_y), havuc_pos)
        # Mesafe, iki yarıçapın toplamından küçük/eşitse "yakalandı" skoru artır
        if d <= (HAVUC_Y + TAVSAN_Y * 0.8):
            skor += 1
            # Yeni havuç üretme (konum+ hız rastgele)
            havuc_pos, havuc_vel = random_havuc()

        # Yazılar
        cv2.putText(frame, f"Skor: {skor}", (15, 35), FONT, 1.0, (255, 255, 255), 2, cv2.LINE_AA)
        cv2.putText(frame, "ESC: cikis", (15, 65), FONT, 0.6, (180, 180, 180), 1, cv2.LINE_AA)

        # Gösterim
        cv2.imshow("Havucu Yakala", frame)


        tus = cv2.waitKey(16) & 0xFF
        if tus == 27:  # ESC
            break

    cv2.destroyAllWindows()
```

main() fonksiyonunda, Program çalıştırıldığında önce bir pencere açılır ve mouse hareketlerini yakalamak için **mouse_callback** fonksiyonu bu pencereye bağlanır. Ardından, ekranda hareket edecek ilk havuç rastgele bir konum ve hızla oluşturulur ve skor sayacı sıfırlanır. Bundan sonra oyun, sürekli tekrarlanan bir döngüye girer. Her döngü adımında, sahne arka planı yeni bir NumPy matrisiyle doldurularak önceki karedeki çizimler temizlenir. Daha sonra havuç, sahip olduğu hız vektörüne göre konumunu günceller. Eğer havuç ekran sınırlarına çarparsa, bu kenardan sekme mantığı ile çözülür: hız bileşeninin işareti ters çevrilir ve konum sınır içine sıkıştırılarak duvara gömülmesi engellenir. Bu işlem, havucun ekranda sürekli zıplayarak dolaşmasını sağlar.

Bir sonraki adımda çizim yapılır: Önce havuç, ardından da mouse konumunu temsil eden tavşan ekrana çizilir. Tavşan doğrudan fare koordinatlarına bağlı olduğundan oyuncu anlık bir kontrol deneyimi yaşar. Ardından yakalama kontrolü yapılır. Tavşan ile havucun merkezleri arasındaki mesafe, yarıçapların toplamına eşit veya daha küçükse bir yakalama gerçekleşmiş sayılır. Bu durumda skor 1 artırılır ve ekranda yeni bir havuç rastgele konum ve hızla belirir. Oyun böylece akıcı bir şekilde devam eder. Sonraki aşamada skor bilgisi ve çıkış yönergesi ekrana yazılır. Çerçeve cv2.imshowile görüntülenir ve cv2.waitKey(16) fonksiyonu kullanılarak yaklaşık 60 kare/saniye hızında bir akış sağlanır. Kullanıcı ESC tuşuna bastığında döngü sonlanır ve pencere kapatılır.

ÇIKTI:



SONUÇ

Kod, OpenCV ve NumPy ile etkileşimli bir oyun iskeletini net biçimde ortaya koymuştur. Mouse ile kontrol, rastgele yön/hızla hareket eden hedef, skor takibi gibi temel oyun becerileri sade ve anlaşılır bir şekilde uygulanmıştır. Bu yapı, daha gelişmiş özellikler için sağlam bir temel sunar.

KAYNAKÇA

https://docs.opencv.org/4.x/d6/d6e/group__imgproc__draw.html

https://www-geeksforgeeks-org.translate.goog/python/python-opencv-cv2-line-method/?_x_tr_sl=en&_x_tr_tl=tr&_x_tr_hl=tr&_x_tr_pto=tc

https://docs.opencv.org/4.x/db/d5b/tutorial_py_mouse_handling.html