

**YAPAY ZEKA VE İLKELERİ**

**BREADTH FİRST SEARCH İLE YILAN OYUNU**

BİLGİSAYAR MÜHENDİSLİĞİ

HAFİZE CEMİLE ÖĞÜT

B210109058

EZGİ GÜL

B210109004

**SAKARYA UYGULAMALI BİLİMLER ÜNİVERSİTESİ**

26 Mayıs 2023

**Proje Özeti: Yılan Oyunu**

Bu proje, genişlik öncelikli arama (BFS) algoritmasını kullanarak bir yılan oyununun geliştirilmesini amaçlamaktadır. Yılan oyunu, klasik bir video oyunudur, oyuncunun yılanı kontrol ederek yiyecekleri yemesi ve yılanın büyümesini sağlaması gerekmektedir.

**Projede verilen görevin açıklanması, gerçeklenme adımları:**

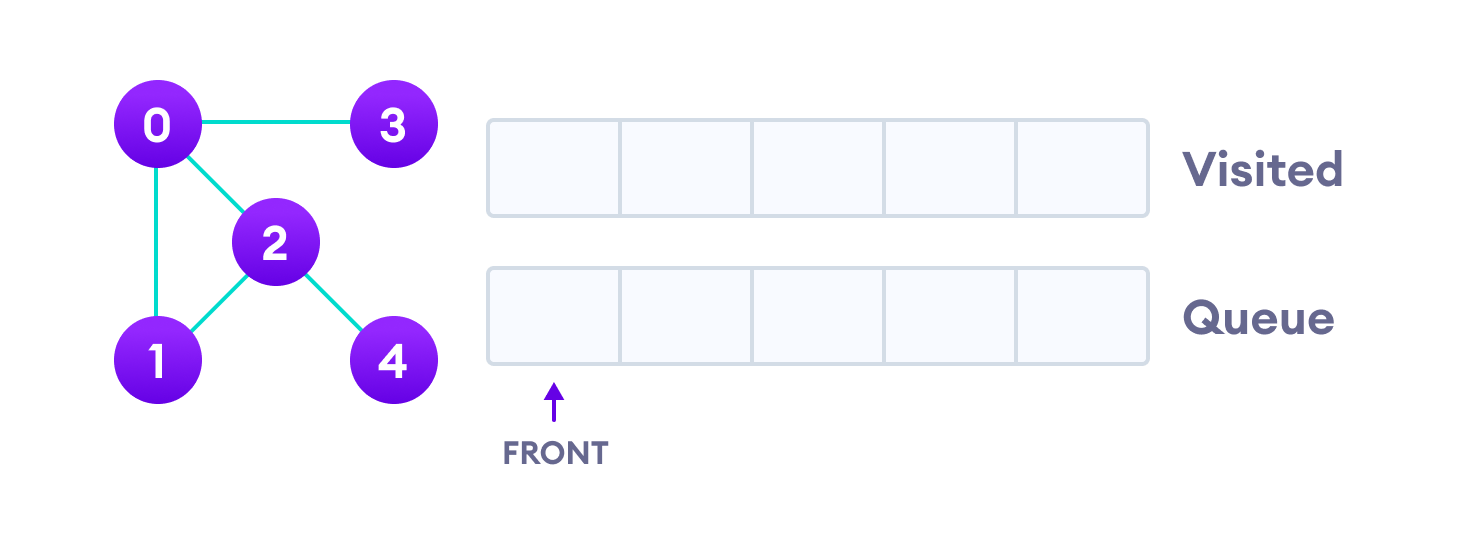
1. Oyun tahtasının oluşturulması:
   * Oyun tahtası, kare şeklinde bir alandır ve yılanın hareket edebileceği hücreleri ve yiyecekleri içerir.
   * Oyun tahtasının boyutunu belirlemeniz gerekmektedir. Örneğin, 10x10 veya 20x20 gibi bir boyut seçebilirsiniz.
   * Oyun tahtasını uygun bir veri yapısı kullanarak temsil etmelisiniz. Matris (dizi içeren dizi) veya benzeri bir veri yapısı kullanabilirsiniz.
2. Yılanın hareketini temsil eden veri yapısının oluşturulması:
   * Yılanın konumunu ve uzunluğunu takip etmek için bir veri yapısı kullanmanız gerekmektedir.
   * Yılanın vücudu genellikle bir dizi veya kuyruk şeklinde temsil edilir. Başlangıçta yılanın uzunluğu belirli bir değerde olabilir.
   * Yılanın konumunu ve vücut parçalarını güncelleyebilmek için veri yapısını uygun şekilde yönetmelisiniz.
3. Yiyeceklerin yerleştirilmesi:
   * Oyun tahtasına yılanın yiyeceğini temsil eden nesneleri (genellikle yiyecekler) yerleştirmeniz gerekmektedir.
   * Yiyecekler rastgele bir konuma yerleştirilebilir. Bu, yılanın hedefi ve yiyeceği yemesi gerektiği için önemlidir.
   * Yeni bir yiyecek yerleştirildiğinde, yılanın uzunluğunu güncellemeniz gerekmektedir.
4. BFS algoritmasının uygulanması:
   * BFS algoritması, genişlik öncelikli bir arama algoritmasıdır ve en kısa yolu bulmak için kullanılır.
   * Başlangıç konumu (yılanın başı) ve hedef (yiyecek) konumu alarak BFS algoritmasını uygulayarak en kısa yolu bulmanız gerekmektedir.
   * BFS algoritmasını kullanarak yılanın hareket edebileceği hücreleri keşfetmelisiniz.
5. Yılanın hareketinin güncellenmesi:
   * En kısa yol bulunduktan sonra, yılanın hareketini güncellemelisiniz.
   * Yılanın başını en kısa yol üzerinde hareket ettirin ve kuyruğunu güncelleyin.
   * Yılan yiyeceği yediğinde, kuyruk uzunluğunu artırmanız gerekmektedir.
6. Oyunun devamının sağlanması:
   * Oyunun devam etmesi için çeşitli kontroller yapmalısınız.
   * Yılanın yiyeceği yemesi durumunda, yeni bir yiyecek yerleştirilmeli ve yılanın uzunluğu güncellenmelidir.
   * Yılan tahtanın kenarına veya kendi vücuduna çarptığında oyun sonlanmalıdır.

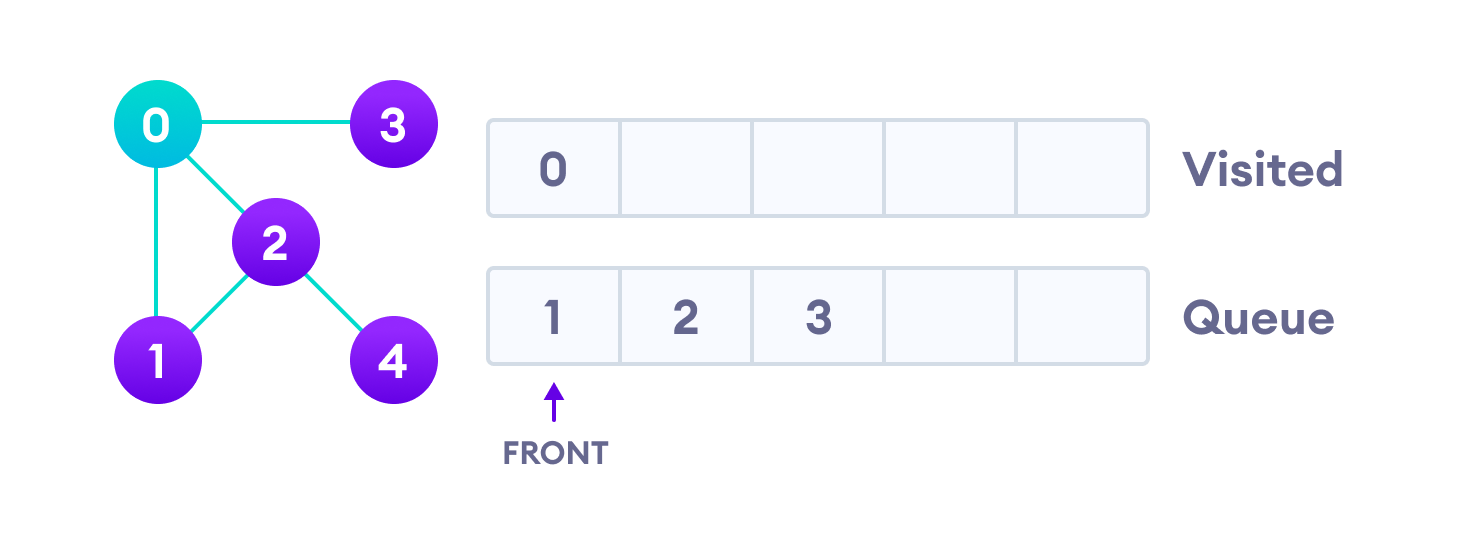
**Kullanılan algoritma:**

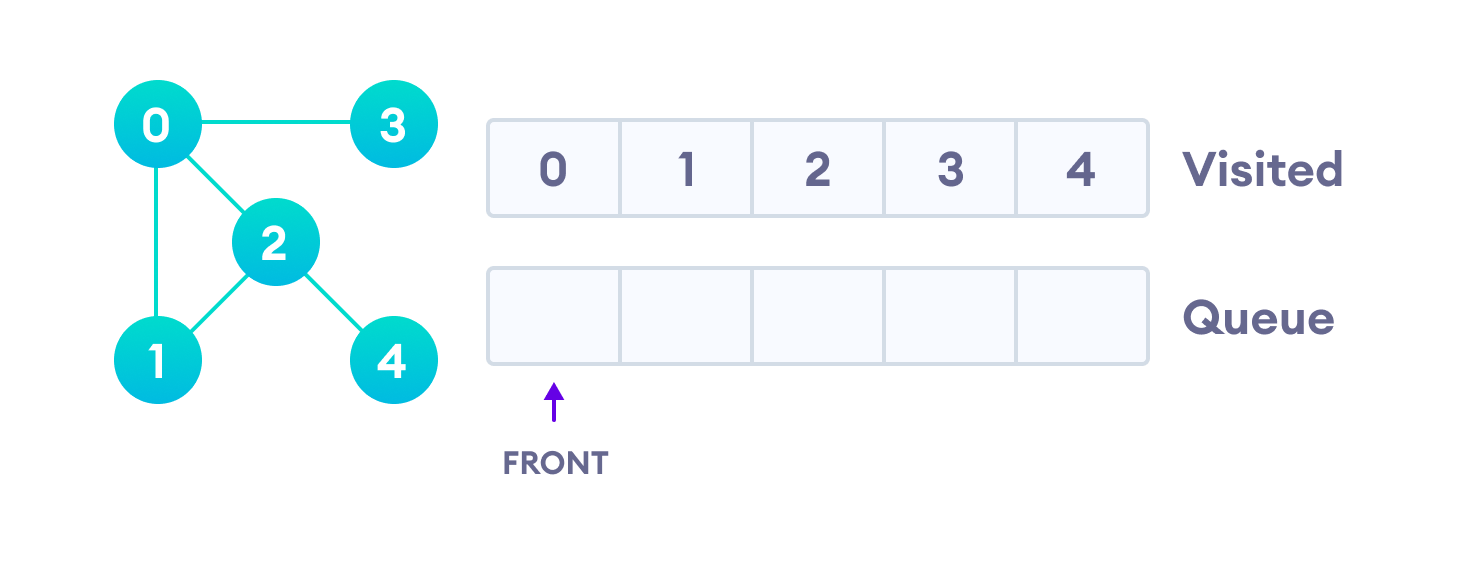
**Breadth First Search: Genişlik Öncelikli Arama**

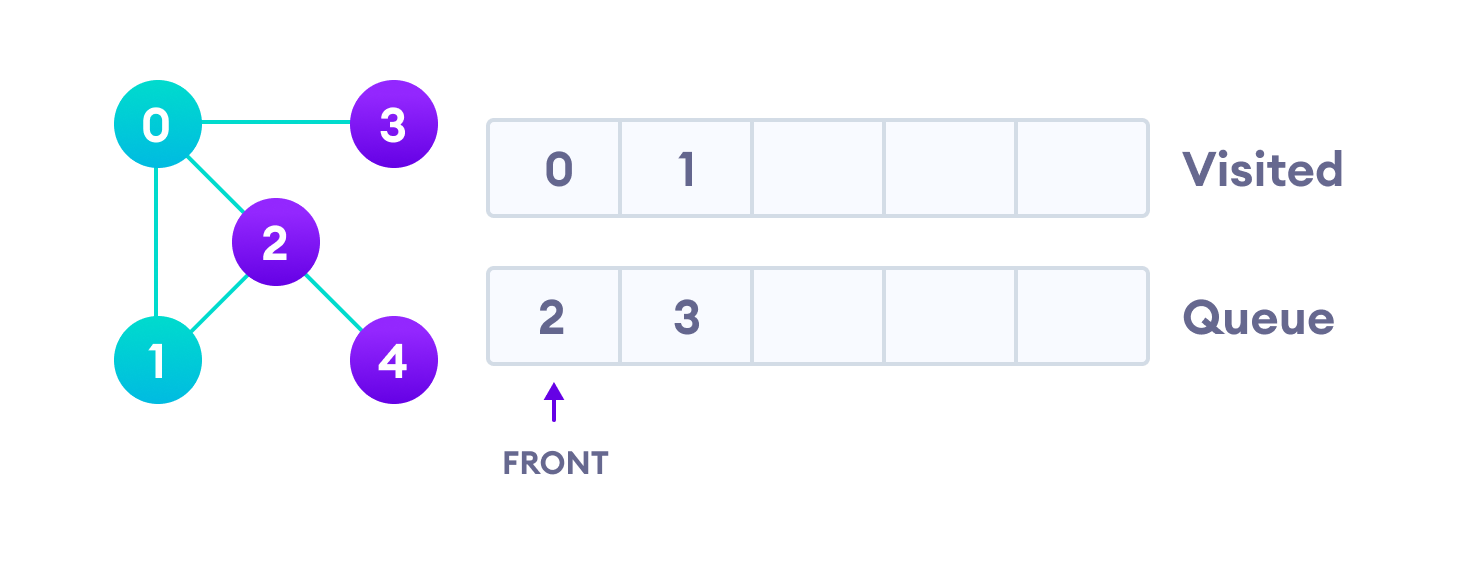
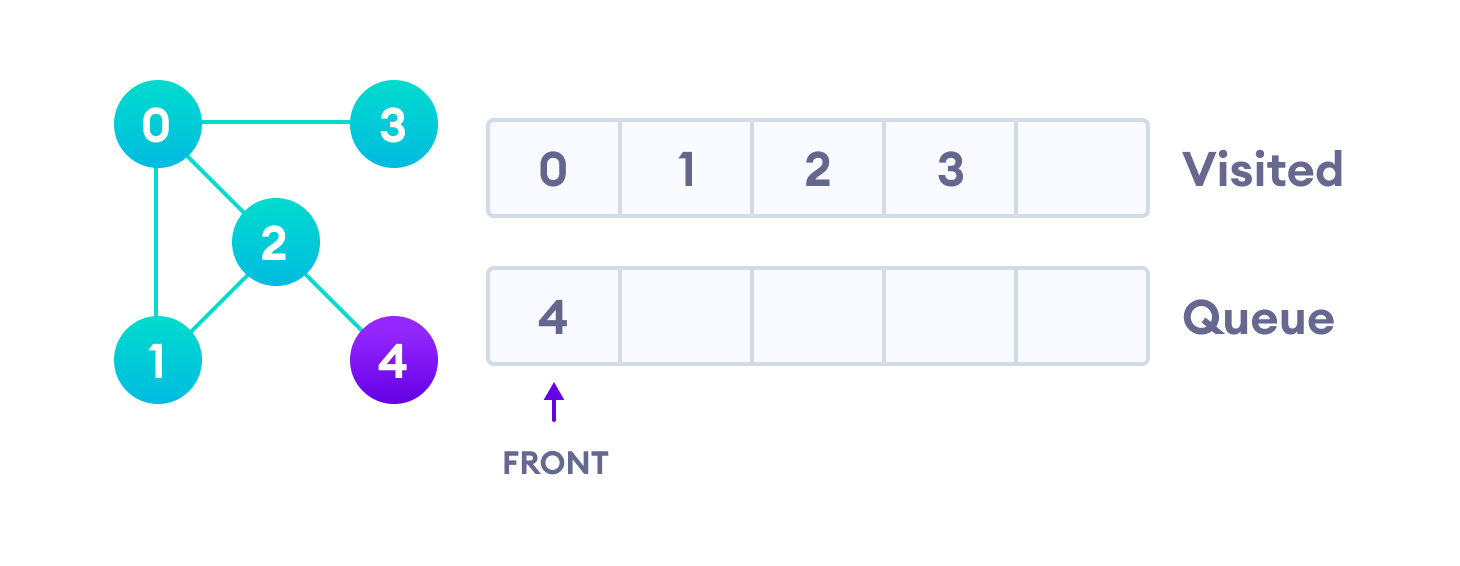
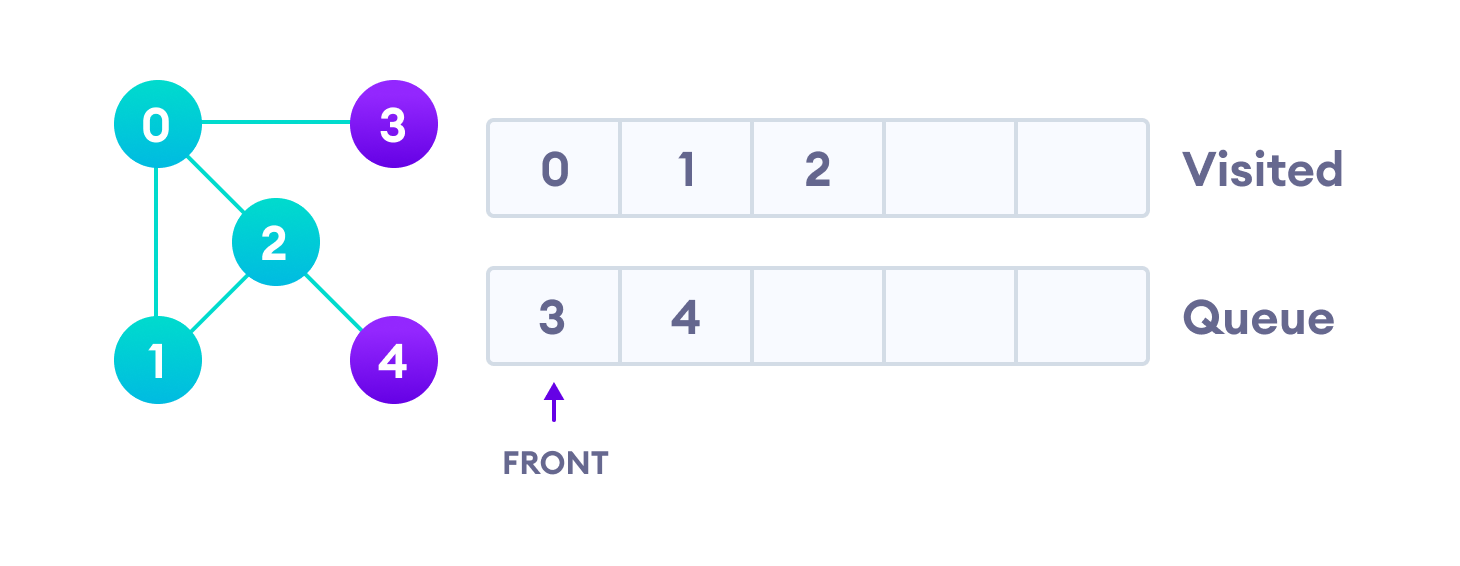
BFS (Genişlik Öncelikli Arama), graf tabanlı problemlerde en kısa yolu veya minimum maliyetli yolu bulmak için kullanılan bir arama algoritmasıdır.

* Başlangıç düğümünden başlar ve tüm komşu düğümleri genişlik önceliğiyle keşfeder.
* Bir kuyruk veri yapısı kullanarak düğümleri keşfetme sırasını tutar.
* Her düğümü keşfettiğinde, komşu düğümleri kuyruğa ekler ve işaretler.
* Kuyruk boşaldığında veya hedef düğüm bulunduğunda durur.
* BFS algoritması, keşfetme sırasına göre en kısa yolu garanti eder.
* Graf tabanlı problemlerde en uygun seçenek olabilir, özellikle minimum maliyetli yolu bulmak istendiğinde.
* Yılan oyunu gibi matris tabanlı problemlerde de BFS algoritması kullanılabilir, yılanın hareket edebileceği hücreleri keşfeder ve hedefe en kısa yolu bulur.









**Genişlik Öncelikli Arama Genel Akış Şeması:**

**metin, ekran görüntüsü, mektup, harf, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

import collections

# BFS algoritması

def bfs(graph, root):

    visited, queue = set(), collections.deque([root])

    visited.add(root)

    while queue:

        # Kuyruktan bir tepe noktasını çıkarıldı

        vertex = queue.popleft()

        print(str(vertex) + " ", end="")

        # Ziyaret edilmediyse, ziyaret edildi olarak işaretleyin

        # enqueue it: kuyruğa alındı

        for neighbour in graph[vertex]:

            if neighbour not in visited:

                visited.add(neighbour)

                queue.append(neighbour)



BFS İLE SNAKE GAME UYGULAMASININ GENEL ALGORİTMASI:

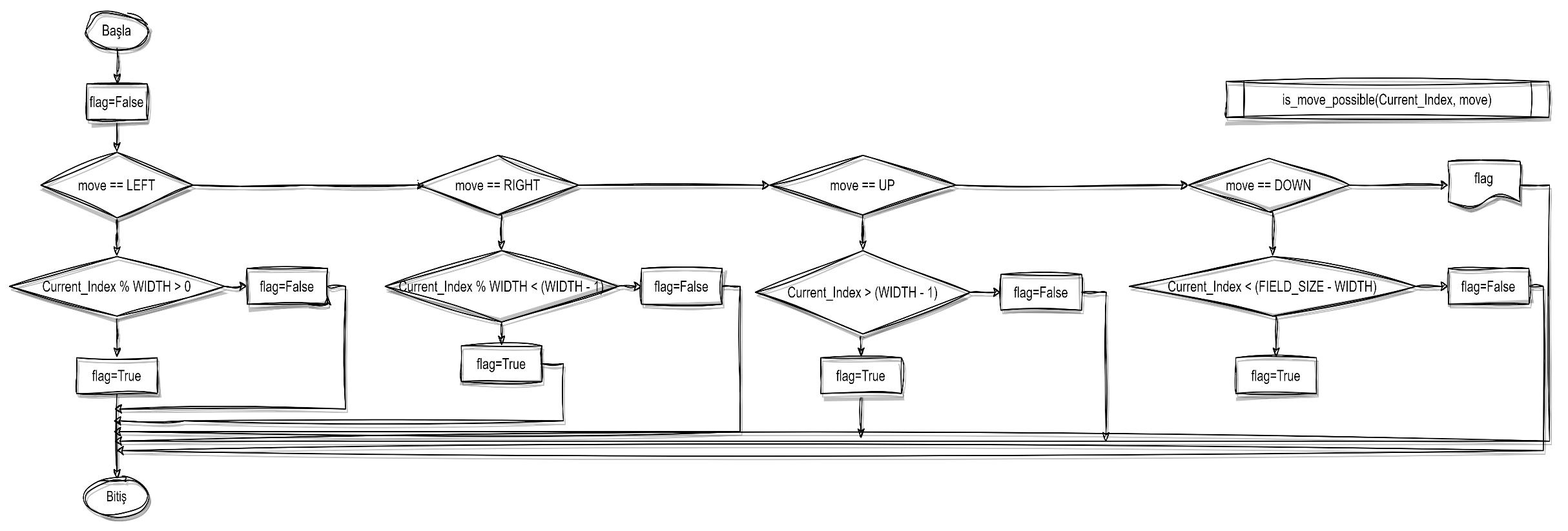
1. Oyun tahtasının oluşturulması: Genişlik (WIDTH) ve yükseklik (HEIGHT) değerleri belirlenir. Bu değerlere göre bir oyun tahtası oluşturulur.
2. Oyunun başlangıç durumu: Yılanın başlangıç konumu, boyutu, yiyecek konumu ve en iyi hareket değişkeni gibi değişkenler tanımlanır.
3. Oyun penceresinin ayarlanması: Pygame kütüphanesi kullanılarak oyun penceresi oluşturulur.
4. Skorun görüntülenmesi için fonksiyonlar tanımlanır.
5. Yiyeceklerin yerleştirilmesi: Oyun tahtası üzerinde rastgele bir konuma yiyecek yerleştirilir.
6. Bir hareketin mümkün olup olmadığını kontrol etme fonksiyonu tanımlanır.
7. Bir hücrenin boş olup olmadığını kontrol etme fonksiyonu tanımlanır.
8. Breadth-First Search (BFS) algoritmasını kullanarak en kısa yolun bulunması sağlanır.
9. Yılanın hareketini temsil eden veri yapısının oluşturulması sağlanır.
10. Yılanın belirtilen yöne hareket etmesini sağlayan fonksiyon tanımlanır.
11. Yılanın kuyruğunun ulaşılabilir olup olmadığını kontrol eden fonksiyon tanımlanır.
12. Yılanın kuyruğunu takip ederek en uzun güvenli hareketi bulan fonksiyon tanımlanır.
13. Yılanın yiyeceğe olan en kısa yolunu bulan ve geçici veri yapılarını güncelleyen fonksiyon tanımlanır.
14. Mevcut konum için en kısa güvenli hareketi seçen fonksiyon tanımlanır.
15. Mevcut konum için en uzun güvenli hareketi seçen fonksiyon tanımlanır.
16. Oyunun devamını sağlayan fonksiyon (gameLoop) tanımlanır.
17. Oyun durumunu günceller ve olayları kontrol eder.

metin, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Akış diyagramı uzun olması sebebi ile görüntü net değil fakat dosyamı projemin içeriğine ekledim. Drawio uygulaması ile online açabilirsiniz. <https://app.diagrams.net/> uygulama dosyamdan sürükleyip bırakmanız yeterli.





Hareket fonksiyonun Akış diyagramı da çalışmada mevcuttur.





import pygame

import random

pygame.init()

#1-Oyun tahtasının oluşturulma:

#Renkler Oyun tahtasında kullanılacak arka plan ve nesnelerin renklerinde kullanılması için renk değerleri oluşturuldu

white = (255, 255, 255)

yellow = (255, 255, 102)

black = (0, 0, 0)

red = (240, 50, 80)

green = (0, 255, 0)

blue = (50, 153, 213)

#Yükseklik ve genişlik değerleri verilerek bir alan boyutu oluşturuldu.

HEIGHT = 30

WIDTH = 40

FIELD\_SIZE = HEIGHT \* WIDTH

HEAD = 0 #Yılanın başının temsil eden head oluşturuldu. Baş değeri 0 olarak atanır.

FOOD = 0 #Yılanın yiyecek ile etkileşimini temsil eden değeri belirtir.

UNDEFINED = (HEIGHT + 1) \* (WIDTH + 1)# tanımlanmamış hücreleri temsil etmek için tanımlanmamış bir alan oluşturuldu.

SNAKE = 2 \* UNDEFINED#yılanın vücudunu temsil etmesi adına bir snake değişkeni oluşturuldu.

LEFT = -1 #Sola hareket

RIGHT = 1 #Sağa hareket

UP = -WIDTH #Yukarı hareket

DOWN = WIDTH #Aşağı hareket

ERR = -2333 # Hata durumu ERR ile temsil edildi.

MOV = [LEFT, RIGHT, UP, DOWN] # yılanın hareket edebileceği yönlere ilişkin sabit bir liste oluşturur.

SNAKE\_BLOCK = 20# Yılanın her bir parçasının boyutu oluşturuldu.

SNAKE\_SPEED = 5000 # Yılanınn hızı belirlendi.

FONT\_STYLE = pygame.font.SysFont("arial", 18) #Your Score yazısının stili belirlendi.

SCORE\_FONT = pygame.font.SysFont("georgia", 20, italic=True, bold=True)

# Oyunun başlangıç durumu

def initial\_game():

    global board, snake, snake\_size, tmpboard, tmpsnake, tmpsnake\_size, food, best\_move

    board = [0] \* FIELD\_SIZE  # [0,0,0,……]Oyun tahtasını temsil eden bir liste oluşturuşdu.

    snake = [0] \* (FIELD\_SIZE + 1)#Yılanın vücudunu temsil eden bir liste oluşturuldu.

    snake[HEAD] = 1 \* WIDTH + 1 #Yılanın başının oyun tahtasındaki konumunu belirlendi.

    snake\_size = 1

    tmpboard = [0] \* FIELD\_SIZE#Geçici bir oyun tahtası oluşturuldu.

    #Bu tahta, oyun tahtasının aynı boyutuna sahip olup, başlangıçta tüm hücrelerini boş (0) olarak ayarlar.

    tmpsnake = [0] \* (FIELD\_SIZE + 1) # Geçici yılanın vücudunu temsil eden bir dizi oluşturuldu.

    tmpsnake[HEAD] = 1 \* WIDTH + 1#Geçici yılanın başının konumu ayarlandı.

    tmpsnake\_size = 1 #Geçici yılanın boyutu 1 olarak ayarlandı, çünkü başlangıçta yılan sadece bir hücreden oluşur.

    food = 4 \* WIDTH + 7 #Yiyeceğin konumu belirlendi.

    best\_move = ERR

# Oyun penceresini ayarla

display\_Board = pygame.display.set\_mode((SNAKE\_BLOCK \* WIDTH, SNAKE\_BLOCK \* HEIGHT))

clock = pygame.time.Clock()

#Breadth-First Snake Game,Your Score yazılarının stilleri belirlendi.

def Your\_score(score):

    game\_title = "Breadth-First Snake Game"

    score\_text = "Your Score: " + str(score)

    game\_title\_surface = SCORE\_FONT.render(game\_title, True, red)

    score\_surface = SCORE\_FONT.render(score\_text, True, red)

    #Yazıların konumu belirlendi.

    display\_Board.blit(game\_title\_surface, [180, 0])

    display\_Board.blit(score\_surface, [250, 40])

#Yılanın yediği yiyeceğin yeni bir konuma yerleştirilmesini sağlayan parametresiz bir fonksiyon oluşturuldu.

def new\_food():

    global food, snake\_size, display\_Board

    cell\_free = False

    while not cell\_free:

        #Rastgele bir konum oluşturuldu.

        w = random.randint(0, WIDTH - 1)

        h = random.randint(0, HEIGHT - 1)

        food = WIDTH \* h + w

        cell\_free = is\_cell\_free(food, snake\_size, snake)

#Oyun tahtasında yılanın ve yiyeceğin görsel temsilini çizecek parametresiz bir fonksiyon oluşturuldu.

def draw():

    global SNAKE\_BLOCK, snake, snake\_size, food

    for Current\_Index in snake[:snake\_size]:

        pygame.draw.rect(display\_Board, black,

                         [SNAKE\_BLOCK \* (Current\_Index % WIDTH), SNAKE\_BLOCK \* (Current\_Index // WIDTH), SNAKE\_BLOCK, SNAKE\_BLOCK])

    pygame.draw.rect(display\_Board, green,

                     [SNAKE\_BLOCK \* (food % WIDTH), SNAKE\_BLOCK \* (food // WIDTH), SNAKE\_BLOCK, SNAKE\_BLOCK])

# Mesajları göstermek için  yazılacak mesajı ve rengini ifade eden iki farklı parametreli bir fonksiyon oluşturuldu.

def message(msg, color):

    mesg = FONT\_STYLE.render(msg, True, color)

    display\_Board.blit(mesg, [WIDTH \* SNAKE\_BLOCK / 6, HEIGHT \* SNAKE\_BLOCK / 3])

# Bir hareketin mümkün olup olmadığını kontrol etmek için index ve hareket değerininden oluşan iki farklı parametreden oluşan fonksiyon oluşturuldu.

def is\_move\_possible(Current\_Index, move):

    flag = False

    if move == LEFT:

        flag = True if Current\_Index % WIDTH > 0 else False

    elif move == RIGHT:

            flag = True if Current\_Index % WIDTH < (WIDTH - 1) else False

    elif move == UP:

          flag = True if Current\_Index > (WIDTH - 1) else False

    elif move == DOWN:

         flag = True if Current\_Index < (FIELD\_SIZE - WIDTH) else False

    return flag

#Belirli bir hücrenin yılanın vücudunda olup olmadığını kontrol eden fonksiyon oluşturuldu.

def is\_cell\_free(Current\_Index, ssize, Indices\_Of\_snake\_Body):

    return not (Current\_Index in Indices\_Of\_snake\_Body[:ssize])

#Current\_Index parametresi, kontrol edilecek hücrenin indeksini temsil eder.

# ssize parametresi, yılanın mevcut boyutunu belirtir.

# Indices\_Of\_snake\_Body ise yılanın vücudunun indekslerini içeren bir liste ya da dizi olarak düşünülebilir.

#Breadth-First Search (BFS) algoritmasını kullanarak yılanın yiyeceğe ulaşabileceği en kısa yolun hesaplanmasını sağlayan BFS fonksiyonu oluşturuldu.

def BFS(Ifood,Indices\_Of\_snake\_Body , GameBoard):

    #Ifood parametresi:yiyeceğin indeksini ,#Indices\_Of\_snake\_Body yılanın vücudunun indekslerini içeren bir liste ,#GameBoard: oyun tahtasıdır

    queue = []

    queue.append(Ifood)

    inqueue = [0] \* FIELD\_SIZE# Oyun tahtasındaki hücreler ziyaret edil dimi? Bunun takibin yapıldığı bir liste.

    found = False #Yiyeceğe ulaşıldı mı?

    # BFS Düşünce Yapısı:

    # 1\* İlk düğümü kuyruktan çıkarın ve hedefe ulaşıldı mı kontrol edin? Evet --> tamamlandı : Hayır --> çocukları ekle

    # 2\* Çıkar

    # 3\* Hedef mi kontrol et?

    # 4\* Hedefe ulaşıldıysa çözümü döndür

    # 5\* Hedefe ulaşılamadıysa çocukları ekle

    # 6\* .....

    while len(queue) != 0:

        # İlk elemanı çıkar

        Current\_Index = int(queue.pop(0))

        # Kontrol et

        if inqueue[Current\_Index] == 1:

            continue

        inqueue[Current\_Index] = 1

        # Komşuları üzerinde döngüye gir (dört olası yönlü hareket)

        for i in range(4):

            if is\_move\_possible(Current\_Index, MOV[i]):

                # Eğer komşu yılanın başına karşılık geliyorsa -- Tamamlandı

                if Current\_Index + MOV[i] == Indices\_Of\_snake\_Body[HEAD]:

                    found = True

                # Eğer komşu boş bir alana karşılık geliyorsa

                if GameBoard[Current\_Index + MOV[i]] < SNAKE:

                    if GameBoard[Current\_Index + MOV[i]] > GameBoard[Current\_Index] + 1:

                        GameBoard[Current\_Index + MOV[i]] = GameBoard[Current\_Index] + 1

                    if inqueue[Current\_Index + MOV[i]] == 0:

                        queue.append(Current\_Index + MOV[i])

    return found

# Yılanın Kuyruğunun Ulaşılabilir Olup Olmadığını Kontrol Et

def is\_tail\_reachable():

    global tmpboard, tmpsnake, food, tmpsnake\_size

    tmpboard[tmpsnake[tmpsnake\_size - 1]] = FOOD

    tmpboard[food] = SNAKE

    result = BFS(tmpsnake[tmpsnake\_size - 1], tmpsnake, tmpboard)

    for i in range(4):

        if is\_move\_possible(tmpsnake[HEAD], MOV[i]) and tmpsnake[HEAD] + MOV[i] == tmpsnake[

            tmpsnake\_size - 1] and tmpsnake\_size > 3:

            result = False

    return result

#2. Yılanın hareketini temsil eden veri yapısının oluşturulması:

# Yılanı belirtilen yöne hareket ettirir

def make\_move(move):

    global snake, board, snake\_size, score

    shift\_array(snake, snake\_size)

    snake[HEAD] += move

    p = snake[HEAD]

    # Oyun tahtasını günceller

    if snake[HEAD] == food:

        board[snake[HEAD]] = SNAKE

        snake\_size += 1

        if snake\_size < FIELD\_SIZE:

            new\_food()

    else:

        board[snake[HEAD]] = SNAKE

        board[snake[snake\_size]] = UNDEFINED

#3. Yiyeceklerin yerleştirilmesi:

# Oyun tahtasını yiyecek yerleştirerek sıfırlar

def board\_reset(Indices\_Of\_snake\_Body, ssize, GameBoard):

    for i in range(FIELD\_SIZE):

        if i == food:

            GameBoard[i] = FOOD

        # Yılan vücudunun bir parçası olmayan tüm hücreleri Tanımsız (UNDEFINED) olarak sıfırlar

        elif is\_cell\_free(i, ssize, Indices\_Of\_snake\_Body):

            GameBoard[i] = UNDEFINED

        else:

            GameBoard[i] = SNAKE

# Yılan için en kısa güvenli hareketi BFS kullanarak bulur

def find\_safe\_way():

    global snake, board

    safe\_move = ERR

    virtual\_shortest\_move()

    if is\_tail\_reachable():

        return choose\_shortest\_safe\_move(snake, board)

    safe\_move = follow\_tail()

    return safe\_move

# Bir diziyi bir birim sağa kaydırır

def shift\_array(arr, size):

    for i in range(size, 0, -1):

        arr[i] = arr[i - 1]

# Yılanın mevcut konum için en iyi olası hareketi bulur, güvenli hareketler yoksa

def any\_possible\_move():

    global food, snake, snake\_size, board

    best\_move = ERR

    # Her bir olası hareket için yiyeceğe olan minimum mesafeyi bulmak için BFS kullanılır

    board\_reset(snake, snake\_size, board)

    BFS(food, snake, board)

    min = SNAKE

    # En kısa mesafeye sahip olan en iyi hareketi seçer

    for i in range(4):

        if is\_move\_possible(snake[HEAD], MOV[i]) and board[snake[HEAD] + MOV[i]] < min:

            min = board[snake[HEAD] + MOV[i]]

            best\_move = MOV[i]

    return best\_move

# Yılanın kuyruğunu takip ederek en uzun güvenli hareketi bulur, kendisine çarpmadan

def follow\_tail():

    global tmpboard, tmpsnake, food, tmpsnake\_size

    tmpsnake\_size = snake\_size

    tmpsnake = snake[:]

    board\_reset(tmpsnake, tmpsnake\_size, tmpboard)

    tmpboard[tmpsnake[tmpsnake\_size - 1]] = FOOD

    tmpboard[food] = SNAKE

    BFS(tmpsnake[tmpsnake\_size - 1], tmpsnake, tmpboard)

    tmpboard[tmpsnake[tmpsnake\_size - 1]] = SNAKE

    return choose\_longest\_safe\_move(tmpsnake, tmpboard)

# Yılanın yiyeceğe olan en kısa yolunu bulur ve ardından tmpsnake ve tmpboard dizilerini günceller

def virtual\_shortest\_move():

    global snake, board, snake\_size, tmpsnake, tmpboard, tmpsnake\_size, food

    tmpsnake\_size = snake\_size

    tmpsnake = snake[:]

    tmpboard = board[:]

    board\_reset(tmpsnake, tmpsnake\_size, tmpboard)

#4. BFS algoritmasının uygulanması:

    food\_eated = False

    while not food\_eated:

        BFS(food, tmpsnake, tmpboard)

        move = choose\_shortest\_safe\_move(tmpsnake, tmpboard)

        shift\_array(tmpsnake, tmpsnake\_size)

        tmpsnake[HEAD] += move

        if tmpsnake[HEAD] == food:

            tmpsnake\_size += 1

            board\_reset(tmpsnake, tmpsnake\_size, tmpboard)

            tmpboard[food] = SNAKE

            food\_eated = True

        else:

            tmpboard[tmpsnake[HEAD]] = SNAKE

            tmpboard[tmpsnake[tmpsnake\_size]] = UNDEFINED

# Mevcut konum için en kısa güvenli hareketi seçer

def choose\_shortest\_safe\_move(psnake, pboard):

    best\_move = ERR

    min = SNAKE

    for i in range(4):

        if is\_move\_possible(psnake[HEAD], MOV[i]) and pboard[psnake[HEAD] + MOV[i]] < min:

            min = pboard[psnake[HEAD] + MOV[i]]

            best\_move = MOV[i]

    return best\_move

# Mevcut konum için en uzun güvenli hareketi seçer

def choose\_longest\_safe\_move(psnake, pboard):

    best\_move = ERR

    max = -1

    for i in range(4):

        if is\_move\_possible(psnake[HEAD], MOV[i]) and pboard[psnake[HEAD] + MOV[i]] < UNDEFINED and pboard[

            psnake[HEAD] + MOV[i]] > max:

            max = pboard[psnake[HEAD] + MOV[i]]

            best\_move = MOV[i]

    return best\_move

#5. Yılanın hareketinin güncellenmesi:

# 6.    Oyunun devamının sağlanması:

# Oyun durumunu günceller ve herhangi bir olayı kontrol eder ve nihayetinde Oyun Tahtasını gösterir

def gameLoop():

    game\_over = False

    game\_close = False

    initial\_game()

    while not game\_over:

        while game\_close == True:

            display\_Board.fill(blue)

            message("You Lost! Press C-Play Again or Q-Quit", red)

            Your\_score(snake\_size - 1)

            pygame.display.update()

            for event in pygame.event.get():

                if event.type == pygame.KEYDOWN:

                    if event.key == pygame.K\_q or event.key == pygame.K\_ESCAPE:

                        game\_over = True

                        game\_close = False

                    if event.key == pygame.K\_c:

                        gameLoop()

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                game\_over = True

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_ESCAPE:

                    game\_over = True

        board\_reset(snake, snake\_size, board)

        if BFS(food, snake, board):

            best\_move = find\_safe\_way()

        else:

            best\_move = follow\_tail()

        if best\_move == ERR:

            best\_move = any\_possible\_move()

        if best\_move != ERR:

            make\_move(best\_move)

        else:

            game\_close = True

        display\_Board.fill(blue)

        draw()

        Your\_score(snake\_size - 1)

        pygame.display.update()

        clock.tick(SNAKE\_SPEED)

    pygame.quit()

    quit()

gameLoop()



ekran görüntüsü, yazılım, multimedya yazılımı, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü, yazılım, metin, bilgisayar içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü, diyagram, metin, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu