Uma imagem com texto, Tipo de letra, logótipo, Azul elétrico

Descrição gerada automaticamente

2023/2024

Pedro Henrique Coelho Rodrigues 25982

Filipe Gonçalves Araújo 25981

Licenciatura em Engenharia em

Desenvolvimento de Jogos Digitais

Introdução à Visão por Computador

Trabalho Prático 1

Índice

[Introdução 2](#_Toc149921094)

[Objetivo 3](#_Toc149921095)

[Software Fonte Desenvolvido 4](#_Toc149921096)

[camara.py 4](#_Toc149921097)

[segmentation.py 5](#_Toc149921098)

[breakout.py 8](#_Toc149921099)

[Desenvolvimento 14](#_Toc149921100)

[Escolha do jogo 14](#_Toc149921101)

[Inicio e primeiros problemas 14](#_Toc149921102)

[Desenvolvimento do ficheiro camara.py 15](#_Toc149921103)

[Desenvolvimento do ficheiro segmentation.py 16](#_Toc149921104)

[Modificações realizadas no código original 18](#_Toc149921105)

[Movimento do Player 18](#_Toc149921106)

[Criação de um game loop 18](#_Toc149921107)

[Conclusão 20](#_Toc149921108)

# Introdução

Com este trabalho, pretende-se dar aos alunos a oportunidade de desenvolver um sistema de User Interface (UI) para um jogo com recurso a técnicas de Visão por Computador. O jogo base escolhido pelo nosso grupo é o Break Out.

A ênfase dos projetos a desenvolver será colocada nas técnicas de Visão por Computador, pelo que será usada uma implementação base do jogo. A linguagem de programação a usar nos projetos é o Python.

# Objetivo

O objetivo do trabalho é que o jogador controle o objeto através da câmara.

O controlo deve ser baseado em algoritmos de segmentação. Poderá ser aplicado qualquer algoritmo de segmentação de entre os abordados na aula. O objeto a ser segmentado na imagem poderá ser um ou dois objetos de cor predefinida. Se o(s) objeto(s) a segmentar forem as mãos ou a cara do jogador, a segmentação poderá basear-se na cor da pele. O controlo deve basear-se na posição do(s) objeto(s) segmentados na imagem.

# Software Fonte Desenvolvido

## camara.py

import cv2  
import segmentation  
  
*#captura da camara*cap = cv2.VideoCapture()  
  
*#loop da camara*def camara\_loop():  
 if not cap.isOpened():  
 cap.open(0)  
 \_, image = cap.read()  
 *#cv2.imshow("Image", image)* else:  
 ret, image = cap.read()  
  
 if not ret:  
 print("Error")  
  
 else:  
 image = image[:, ::-1, :] *#inverter camara* cv2.imshow("Image", image)  
 window\_size = cv2.getWindowImageRect("Image")  
 image\_hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV) *#Passar para hsv* mask = segmentation.update\_segmentation(image\_hsv) *#Criar a mascara atraves da funcao segmentacao* center = segmentation.find\_center(mask) *#Encontrar o centro* if center is not None:  
 center\_x = center[0]  
  
 image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
  
 cv2.circle(image, center, 5, (0, 255, 0), -1) *#Marcar o centro com um ponto* cv2.imshow("Result", image)  
 return center\_x

## segmentation.py

import cv2  
import numpy as np  
  
hmin = 43  
hmax = 99  
smin = 61  
smax = 160  
vmin = 44  
vmax = 141  
  
  
*#funcao responsavel pela segmentacao*def update\_segmentation(image\_hsv):  
 if hmin < hmax:  
 ret, mask\_hmin = cv2.threshold(src=image\_hsv[:, :, 0], thresh=hmin-1,  
 maxval=1, type=cv2.THRESH\_BINARY)  
 ret, mask\_hmax = cv2.threshold(src=image\_hsv[:, :, 0], thresh=hmax,  
 maxval=1, type=cv2.THRESH\_BINARY\_INV)  
 mask\_h = mask\_hmin \* mask\_hmax  
 else:  
 ret, mask\_hmin = cv2.threshold(src=image\_hsv[:, :, 0], thresh=hmin,  
 maxval=1, type=cv2.THRESH\_BINARY)  
 ret, mask\_hmax = cv2.threshold(src=image\_hsv[:, :, 0], thresh=hmax-1,  
 maxval=1, type=cv2.THRESH\_BINARY\_INV)  
 mask\_h = cv2.bitwise\_or(mask\_hmin, mask\_hmax)  
  
  
 ret, mask\_smin = cv2.threshold(src=image\_hsv[:, :, 1],  
 thresh=smin, maxval=1, type=cv2.THRESH\_BINARY)  
 ret, mask\_smax = cv2.threshold(src=image\_hsv[:, :, 1],  
 thresh=smax, maxval=1, type=cv2.THRESH\_BINARY\_INV)  
 mask\_s = mask\_smin \* mask\_smax  
  
  
 ret, mask\_vmin = cv2.threshold(src=image\_hsv[:, :, 2],  
 thresh=vmin, maxval=1, type=cv2.THRESH\_BINARY)  
 ret, mask\_vmax = cv2.threshold(src=image\_hsv[:, :, 1],  
 thresh=vmax, maxval=1, type=cv2.THRESH\_BINARY\_INV)  
 mask\_v = mask\_vmin \* mask\_vmax  
  
  
 *# cv2.imshow("Mask H min", mask\_hmin\*255)  
 # cv2.imshow("Mask H max", mask\_hmax \* 255)  
 #cv2.imshow("Mask H", mask\_h \* 255)  
 #cv2.imshow("Mask S min", mask\_smin \* 255)  
 #cv2.imshow("Mask S max", mask\_smax \* 255)  
 #cv2.imshow("Mask S", mask\_s \* 255)  
 # cv2.imshow("Mask V min", mask\_vmin \* 255)  
 # cv2.imshow("Mask V max", mask\_vmax \* 255)  
 #cv2.imshow("Mask V", mask\_v \* 255)* mask = mask\_s \* mask\_h \* mask\_v  
 *#cv2.imshow("Mask", mask \*255)* contours, hierarchy = cv2.findContours(image=mask,  
 mode=cv2.RETR\_TREE,  
 method=cv2.CHAIN\_APPROX\_NONE)  
  
 mask\_filtered = np.zeros(mask.shape, np.uint8)  
 for i in range (len(contours)):  
 contour = contours[i]  
 contour\_area = cv2.contourArea(contour)  
 if contour\_area > 100:  
 cv2.drawContours(image=mask\_filtered, contours=contours,  
 contourIdx=i, color=1,thickness=-1)  
 cv2.imshow("Mask Filtered", mask\_filtered \* 255)  
 return mask\_filtered  
  
  
*#Encontrar o centro da mascara*def find\_center(mask):  
  
 contours, hierarchy = cv2.findContours(image=mask,  
 mode=cv2.RETR\_TREE,  
 method=cv2.CHAIN\_APPROX\_NONE)  
 if len(contours) > 0:  
 contour = max(contours, key=cv2.contourArea)  
  
 M = cv2.moments(contour)  
 if M['m00'] != 0:  
 Cx = int(np.round(M['m10'] / M['m00']))  
 Cy = int(np.round(M['m01'] / M['m00']))  
 return Cx, Cy  
 return None  
  
*#Criacao de trackbars*def trackbar():  
 def on\_change\_hmin(val):  
 global hmin  
 hmin = val  
  
 def on\_change\_hmax(val):  
 global hmax  
 hmax = val  
  
 def on\_change\_smin(val):  
 global smin  
 smin = val  
  
 def on\_change\_smax(val):  
 global smax  
 smax = val  
  
 def on\_change\_vmin(val):  
 global vmin  
 vmin = val  
  
 def on\_change\_vmax(val):  
 global vmax  
 vmax = val  
  
 cv2.namedWindow("Image")  
 cv2.createTrackbar("Hmin", "Image", hmin, 180, on\_change\_hmin)  
 cv2.createTrackbar("Hmax", "Image", hmax, 180, on\_change\_hmax)  
 cv2.createTrackbar("Smin", "Image", smin, 255, on\_change\_smin)  
 cv2.createTrackbar("Smax", "Image", smax, 255, on\_change\_smax)  
 cv2.createTrackbar("Vmin", "Image", vmin, 255, on\_change\_vmin)  
 cv2.createTrackbar("Vmax", "Image", vmax, 255, on\_change\_vmax)

## breakout.py

*"""  
 Sample Breakout Game  
  
 Sample Python/Pygame Programs  
 Simpson College Computer Science  
 http://programarcadegames.com/  
 http://simpson.edu/computer-science/  
"""  
  
# --- Import libraries used for this program*import math  
  
import cv2  
import pygame  
  
import camara  
import segmentation  
  
*# Define some colors*black = (0, 0, 0)  
white = (255, 255, 255)  
blue = (0, 0, 255)  
  
*# Size of break-out blocks*block\_width = 23  
block\_height = 15  
  
  
class Block(pygame.sprite.Sprite):  
 *"""This class represents each block that will get knocked out by the ball  
 It derives from the "Sprite" class in Pygame """* def \_\_init\_\_(self, color, x, y):  
 *""" Constructor. Pass in the color of the block,  
 and its x and y position. """  
  
 # Call the parent class (Sprite) constructor* super().\_\_init\_\_()  
  
 *# Create the image of the block of appropriate size  
 # The width and height are sent as a list for the first parameter.* self.image = pygame.Surface([block\_width, block\_height])  
  
 *# Fill the image with the appropriate color* self.image.fill(color)  
  
 *# Fetch the rectangle object that has the dimensions of the image* self.rect = self.image.get\_rect()  
  
 *# Move the top left of the rectangle to x,y.  
 # This is where our block will appear..* self.rect.x = x  
 self.rect.y = y  
  
  
class Ball(pygame.sprite.Sprite):  
 *""" This class represents the ball  
 It derives from the "Sprite" class in Pygame """  
  
 # Speed in pixels per cycle* speed = 4.0  
  
 *# Floating point representation of where the ball is* x = 0.0  
 y = 180.0  
  
 *# Direction of ball (in degrees)* direction = 200  
  
 width = 10  
 height = 10  
  
 *# Constructor. Pass in the color of the block, and its x and y position* def \_\_init\_\_(self):  
 *# Call the parent class (Sprite) constructor* super().\_\_init\_\_()  
  
 *# Create the image of the ball* self.image = pygame.Surface([self.width, self.height])  
  
 *# Color the ball* self.image.fill(white)  
  
 *# Get a rectangle object that shows where our image is* self.rect = self.image.get\_rect()  
  
 *# Get attributes for the height/width of the screen* self.screenheight = pygame.display.get\_surface().get\_height()  
 self.screenwidth = pygame.display.get\_surface().get\_width()  
  
 def bounce(self, diff):  
 *""" This function will bounce the ball  
 off a horizontal surface (not a vertical one) """* self.direction = (180 - self.direction) % 360  
 self.direction -= diff  
  
 def update(self):  
 *""" Update the position of the ball. """  
 # Sine and Cosine work in degrees, so we have to convert them* direction\_radians = math.radians(self.direction)  
  
 *# Change the position (x and y) according to the speed and direction* self.x += self.speed \* math.sin(direction\_radians)  
 self.y -= self.speed \* math.cos(direction\_radians)  
  
 *# Move the image to where our x and y are* self.rect.x = self.x  
 self.rect.y = self.y  
  
 *# Do we bounce off the top of the screen?* if self.y <= 0:  
 self.bounce(0)  
 self.y = 1  
  
 *# Do we bounce off the left of the screen?* if self.x <= 0:  
 self.direction = (360 - self.direction) % 360  
 self.x = 1  
  
 *# Do we bounce of the right side of the screen?* if self.x > self.screenwidth - self.width:  
 self.direction = (360 - self.direction) % 360  
 self.x = self.screenwidth - self.width - 1  
  
 *# Did we fall off the bottom edge of the screen?* if self.y > 600:  
 return True  
 else:  
 return False  
  
  
class Player(pygame.sprite.Sprite):  
 *""" This class represents the bar at the bottom that the  
 player controls. """* def \_\_init\_\_(self):  
 *""" Constructor for Player. """  
 # Call the parent's constructor* super().\_\_init\_\_()  
  
 self.speed = 5  
 self.width = 75  
 self.height = 15  
 self.image = pygame.Surface([self.width, self.height])  
 self.image.fill((white))  
  
 *# Make our top-left corner the passed-in location.* self.rect = self.image.get\_rect()  
 self.screenheight = pygame.display.get\_surface().get\_height()  
 self.screenwidth = pygame.display.get\_surface().get\_width()  
  
 self.rect.x = 0  
 self.rect.y = self.screenheight - self.height  
 def update(self, center):  
 *""" Update the player position. """  
  
 #mexer o player com as setas do teclado* keys = pygame.key.get\_pressed()  
 if keys[pygame.K\_LEFT]:  
 self.rect.x -= self.speed  
 if keys[pygame.K\_RIGHT]:  
 self.rect.x += self.speed  
  
 *#mexer o player com a camara* if center is not None:  
 if center < self.screenwidth\*(1/3):  
 self.rect.x -= self.speed *# mexer para a esquerda se o ponto(centro) estiver à esquerda* elif center > self.screenwidth\*(2/3):  
 self.rect.x += self.speed *# mexer para a direita se o ponto(centro) estiver à direita  
 # Obrigar o player a ficar dentro dos limites* if self.rect.x < 0:  
 self.rect.x = 0  
 elif self.rect.x + self.width > self.screenwidth:  
 self.rect.x = self.screenwidth - self.width  
  
*# Call this function so the Pygame library can initialize itself*pygame.init()  
  
*#segmentacao*segmentation.trackbar()  
  
*# Create an 800x600 sized screen*screen = pygame.display.set\_mode([800, 600])  
  
*# Set the title of the window*pygame.display.set\_caption('Breakout')  
  
*# Enable this to make the mouse disappear when over our window*pygame.mouse.set\_visible(0)  
  
*# This is a font we use to draw text on the screen (size 36)*font = pygame.font.Font(None, 36)  
  
*# Create a surface we can draw on*background = pygame.Surface(screen.get\_size())  
  
*# Create sprite lists*blocks = pygame.sprite.Group()  
balls = pygame.sprite.Group()  
allsprites = pygame.sprite.Group()  
  
*# Create the player paddle object*player = Player()  
allsprites.add(player)  
  
*# Create the ball*ball = Ball()  
allsprites.add(ball)  
balls.add(ball)  
  
*# The top of the block (y position)*top = 80  
  
*# Number of blocks to create*blockcount = 32  
  
*# --- Create blocks  
  
# Five rows of blocks*for row in range(5):  
 *# 32 columns of blocks* for column in range(0, blockcount):  
 *# Create a block (color,x,y)* block = Block(blue, column \* (block\_width + 2) + 1, top)  
 blocks.add(block)  
 allsprites.add(block)  
 *# Move the top of the next row down* top += block\_height + 2  
  
*# Clock to limit speed*clock = pygame.time.Clock()  
  
*# Is the game over?*game\_over = False  
  
*# Exit the program?*exit\_program = False  
  
def game\_loop():  
 global exit\_program, game\_over, player, ball, blocks, screen, clock  
  
 *# Main program loop* while not exit\_program:  
 *# chamar o loop da camara e encontrar o centro* center = camara.camara\_loop()  
  
 *# Limit to 30 fps* clock.tick(30)  
  
 *# Clear the screen* screen.fill(black)  
  
 *# Process the events in the game* for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 exit\_program = True  
  
 *# Update the ball and player position as long  
 # as the game is not over.* if not game\_over:  
 *# Update the player and ball positions* player.update(center)  
 game\_over = ball.update()  
  
 *# If we are done, print game over* if game\_over:  
 game\_over = False  
 text = font.render("Game Over", True, white)  
 textpos = text.get\_rect(centerx=background.get\_width() / 2)  
 textpos.top = 300  
 screen.blit(text, textpos)  
  
 *# See if the ball hits the player paddle* if pygame.sprite.spritecollide(player, balls, False):  
 *# The 'diff' lets you try to bounce the ball left or right  
 # depending where on the paddle you hit it* diff = (player.rect.x + player.width / 2) - (ball.rect.x + ball.width / 2)  
  
 *# Set the ball's y position in case  
 # we hit the ball on the edge of the paddle* ball.rect.y = screen.get\_height() - player.rect.height - ball.rect.height - 1  
 ball.bounce(diff)  
  
 *# Check for collisions between the ball and the blocks* deadblocks = pygame.sprite.spritecollide(ball, blocks, True)  
  
 *# If we actually hit a block, bounce the ball* if len(deadblocks) > 0:  
 ball.bounce(0)  
  
 *# Game ends if all the blocks are gone* if len(blocks) == 0:  
 game\_over = True  
  
 *# Draw Everything* allsprites.draw(screen)  
  
 *# Flip the screen and show what we've drawn* pygame.display.flip()  
  
*# Call the game loop function to start the game*game\_loop()  
cv2.destroyWindow()  
pygame.quit()

# Desenvolvimento

Este projeto foi desenvolvido usando as bliblotecas math, OpenCV (cv2), numpy e pygame

## Escolha do jogo

O primeiro passo para começar a desenvolver o projeto foi a escolha do jogo, pelo qual optamos pelo Break Out e uma versão bastante simples do mesmo e desenvolvido em pygame.

O código escolhido foi pode ser encontrado neste link:

<http://programarcadegames.com/python_examples/show_file.php?file=breakout_simple.py>

## Inicio e primeiros problemas

De seguida, começamos com a implementação do código. Inicialmente começamos por enfrentar alguns problemas uma vez que não estávamos a conseguir inicializar a câmara e o jogo em simultâneo.

O jogo apenas estava a iniciar após a câmara ser fechada uma vez que a implementação da câmara no ciclo de jogo estava a ser feita de forma errada

Para ultrapassar esta dificuldade, decidimos, diferente do que foi desenvolvido em aula, separar o código da câmara, do jogo, e da segmentação em ficheiros diferentes.

## Desenvolvimento do ficheiro camara.py

Começamos por inicializar um objeto de captura de vídeo.

Função camara\_loop:

* Esta função é um loop que captura continuamente frames da câmara e executa varias funções sobre a mesma.
* Verifica se a câmara não está aberta.
* Se não estiver aberta, a câmara é aberta
* Em seguida, lê um quadro da câmara e inverte horizontalmente a imagem para que a sua exibição seja correta
* Converte a imagem de BGR para o espaço de cores HSV
* Chama uma função responsável pela segmentação de cores para criar uma máscara que será explicada mais a frente.
* Em seguida, é chamada uma função para encontrar o centro do objeto na máscara.
* Se um centro for encontrado, ele é marcado na imagem com um círculo verde.
* Finalmente, a função retorna as coordenadas do centro encontrado, que serão utilizadas noutra parte do código.

## Desenvolvimento do ficheiro segmentation.py

Este código é usado para isolar uma determinada faixa de cores em uma imagem e encontrar o centro da região de interesse após a segmentação

Definição de valores mínimos e máximos para os canais HSV:

* hmin, hmax, smin, smax, vmin, e vmax são variáveis que definem os limites da faixa de cores que você deseja segmentar na imagem. Esses valores representam os limites mínimos e máximos dos canais.

Função update\_segmentation:

* Esta função recebe uma imagem no espaço de cores HSV.
* Ela cria máscaras binárias para cada canal (H, S e V) com base nos valores mínimos e máximos definidos.
* As máscaras são aplicadas uns aos outros usando operações lógicas, resultando em uma única máscara final que representa a região de interesse.
* Em seguida, encontra os contornos na máscara final e cria uma nova máscara, uma máscara filtrada, que mantém apenas os contornos com uma área maior que 100 pixels.
* A função retorna a máscara filtrada

Função find\_center:

* Esta função recebe uma máscara binária, que será a máscara filtrada resultante da função anterior.
* Ela encontra os contornos na máscara e identifica o maior deles com base na sua área.
* Em seguida, calcula o centro usando os momentos da imagem.
* Retorna as coordenadas x e y do centro ou None se nenhum contorno for encontrado.

Função trackbar:

* Esta função cria uma janela de exibição de imagem onde você pode ajustar os valores mínimos e máximos para os canais HSV de forma manual usando trackbars.
* Cada trackbar está associado a uma função de callback que atualiza os valores das variáveis hmin, hmax, smin, smax, vmin e vmax conforme o movimento dos trackbars.

## Modificações realizadas no código original

### Movimento do Player

O código original apresenta apenas uma forma de mexer o player na função update: através tracking do cursor rato. Nós apagamos essa parte do código, colocamos o ponto “center” como argumento, e adicionámos a possibilidade de controlar com as setas do teclado e a possibilidade de controlar usando a câmara.

Como uma parte do código original continha tracking de um ponto, neste caso, o cursor do rato, a fazer a transformação para o uso da câmara começamos por fazer algo semelhante, usando, desta forma, o ponto central da máscara como ponto referência em vez do curso do rato.

Posteriormente, fizemos aquilo que de facto acabou por ser o produto final e era pretendido desde o início q era originalmente pretendido. Se o ponto central estivesse no lado esquerdo da câmara, o player mexe-se para o lado esquerdo, se estiver do lado direito, mexe-se para o lado direito

### Criação de um game loop

O código original não apresenta um game loop. O jogo é desenvolvido apenas dentro de um ciclo While. A criação do game loop surgiu com a intenção de ser possível poder recomeçar o jogo após perder, algo que não é possível no código original, e o jogo não começar imediatamente assim que iniciado, que também acontece no código original

Apesar de termos criado o game loop, acabamos por não melhorar esta parte do projeto por falta de tempo. Decidimos focarmo-nos na parte de Visão em vez da jogabilidade do jogo em si.

Juntamente com a inicialização de variáveis já feitas no código original, iniciamos também a função segmentation.trackbar().

Para a criação do game\_loop, utilizamos o código base do jogo que estava dentro do ciclo While, sendo este mesmo ciclo usado na função game\_loop, transformando as variáveis usadas em variáveis globais

No início do ciclo While, inicializamos a variável center, que, pela função camara\_loop, irá conter as coordenadas do ponto que servirá de referência para movimentar o player

Este center é utilizado como argumento na função update

# Conclusão

O jogo não se encontra no melhor estado possível para ser jogado, uma vez que não apresenta menu nem forma de recomeçar, o que acaba por prejudicar um pouco a demonstração do trabalho desenvolvido usando a técnica de Visão por Computador.

Contudo, o objetivo principal do trabalho, controlar o jogo com a câmara, foi conseguido.

Acreditamos que poderíamos ter realizado um trabalho mais completo se tivéssemos começado a trabalhar um pouco mais cedo.