INTRODUÇÃO À VISÃO POR COMPUTADOR

Trabalho Prático 3

Licenciatura em Engenharia em Desenvolvimento de Jogos Digitais

Pedro Henrique Coelho Rodrigues 25982
Filipe Gonçalves Araújo 25981

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

2023/2024

Índice

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 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.

Na fase 3 o controlo deve ser baseado em algoritmos de tracking ou deteção de movimento. Pode ser aplicado qualquer algoritmo de tracking ou deteção de movimento de entre os abordados na aula. O controlo deverá basear-se no movimento presente nas imagens frames do vídeo ao longo do tempo.

O algoritmo utilizado foi o TrackerCSRT.

Software Fonte Desenvolvido

tracker.py

```
import cv2

cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
tracker = cv2.TrackerCSRT_create()

x, y, w, h = 300, 240, 100, 100
bbox = (x, y, w, h)

_, frame = cap.read()

img = cv2.rectangle(img=frame, pt1=(x, y), pt2=(x+w, y+h), color=255,
thickness=2)
img = cv2.flip(img, 1)  # Invert the camera horizontally
cv2.imshow("Image", img)
tracker.init(frame, bbox)

def tracking():
    global cap  # Use the global cap object

    ret, frame = cap.read()

    if ret:
        track_ok, bbox = tracker.update(frame)
        if track_ok:
            x, y, w, h = bbox
            image_show = cv2.rectangle(img=frame, pt1=(x, y), pt2=(x +
w, y + h), color=255, thickness=2)
            center_x = x + w / 2
        else:
            image_show = frame.copy()
            cv2.putText(img=image_show,
                        text="Tracking failed",
                        org=(5, 35),
                        fontFace=cv2.FONT_HERSHEY_SIMPLEX,
                        fontScale=0.5,
                        color=(0, 0, 255),
                        thickness=2)
        cv2.imshow(winname="Image", mat=image_show)
    return center_x

def camara_loop():
    if not cap.isOpened():
        cap.open(0)
        _, image = cap.read()
    else:
        ret, image = cap.read()
        if not ret:
            print("Error")
        else:
            image = cv2.flip(image, 1)  # Invert the camera
horizontally
            cv2.imshow("Image", image)
```

```
            center = tracking()
            if center is not None:
                image_out = image.copy()
                center_x = int(center)
                flipped_center_x = image.shape[1] - center_x  # 
Calculate flipped center
                cv2.circle(image_out, center=(flipped_center_x, 
int(image.shape[0] / 2)), radius=3,
                           color=(0, 255, 0), thickness=-1)
                cv2.imshow("Result", image_out)
                return flipped_center_x
```

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 tracker

# 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

        if abs(self.rect.x - center) > self.speed:
            if self.rect.x < center:
                self.rect.x += self.speed  # Move right
            else:
                self.rect.x -= self.speed  # Move left

            # 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()

# 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 start_screen():
```

```
    waiting = True
    while waiting:
        screen.fill(black)
        # Display start message
        start_text = font.render("Press any key to start", True,
white)
        textpos = start_text.get_rect(center=(screen.get_width() / 2,
screen.get_height() / 2))
        screen.blit(start_text, textpos)
        pygame.display.flip()

        # Wait for a key press to start the game
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                exit()
            if event.type == pygame.KEYDOWN:
                waiting = False

start_screen()

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 = tracker.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)
            game_loop()

        # 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) e pygame.

Escolha do jogo

Para desenvolver o projeto foi decido manter a escolha do jogo do trabalho prático nº1, ou seja, o Break Out e uma versão bastante simples do mesmo e desenvolvido em pygame.

O código escolhido 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.

Usando como base o código feito no trabalho anterior, o código usado para realizar a segmentação foi trocado pelo código utilizado para fazer o tracking de objetos.

Para a realização deste código, usamos como base o trackerCSRT desenvolvido na aula.

Inicialmente, o programa não estava a funcionar. O programa ficava preso num loop infinito o que impossibilitava a correta utilização do tracking.

Para resolver este problema, foi modificada a estrutura do projeto, e, em vez de 3 ficheiros de código, foram utilizados apenas 2, colocando o código responsável pelo tracking e pela câmara no mesmo ficheiro.

Desenvolvimento do ficheiro tracker.py

De forma geral:

- Este código utiliza a biblioteca OpenCV para rastrear um objeto em um fluxo de vídeo da câmara;
- Ele usa o algoritmo CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability) para realizar o rastreamento do objeto;
- A ideia principal é identificar um objeto na tela e rastreá-lo conforme ele se move na câmara.

Partes do Código:

- Inicialização da Câmara e do Tracker:
  - Usa a função cv2.VideoCapture() para aceder à câmara;
  - Cria um tracker usando cv2.TrackerCSRT_create().

- Definição da Região de Interesse:
  - Define uma região retangular na imagem inicial que será rastreada;
  - Inicializa o rastreador com essa região.

- Função de Rastreamento ("tracking()"):
  - Obtém um novo quadro da câmara;
  - Atualiza o rastreamento usando o método update() do rastreador;
  - Desenha um retângulo na posição atual do objeto rastreado;
  - Se o rastreamento falhar, exibe uma mensagem na tela.

- Loop Principal (“camera_loop()”):
    - Verifica se a câmara está aberta; se não, a abre;
    - Lê um novo frame da câmara e inverte horizontalmente;
    - Mostra o frame na tela;
    - Chama a função de tracking (“tracking()");
    - Se o rastreamento for bem-sucedido, mostra um círculo no centro posição do objeto rastreado, considerando a inversão horizontal.

Observações:

O tracking é realizado nos frames captados originalmente, sem serem invertidos.

Posteriormente, quer a imagem da câmara, quer a região retangular que identifica a zona a ser rastreada, quer o circulo que marca o centro são invertidas horizontalmente, ou seja, as coordenadas que são retornadas da função camara_loop e que são usadas para controlar a câmara, são as coordenadas inversas no eixo do x do ponto que é originalmente rastreado.

Modificações realizadas no código original

Movimento do Player

O código original apresenta apenas uma forma de mexer o player na função update: através do 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 usamos algo semelhante, uma vez que a posição em que se encontra o ponto central da área retangular rastreada vai ditar a posição do player durante o jogo (no eixo dos x).

Criação de um start screen

Foi criado um simples start screen com um fundo preto e a frase “Press any key to start”.

Assim que qualquer tecla for pressionada, a câmara ligará e o jogo começará.

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.

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

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 que pertence á classe player.

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. A janela do jogo não abre em primeiro plano, ficando atrás das janelas da câmara, o que também prejudica o começo do jogo, e, por isso, é utilizado uma velocidade baixa para a bola para que seja possível ver o objetivo principal, que é controlar o player.

Apesar de não estar nas condições mais jogáveis, o objetivo principal foi cumprido.