



INSTITUTO  
POLITÉCNICO  
DO CÁVADO  
E DO AVE



ESCOLA  
SUPERIOR  
DE TECNOLOGIA  
IPCA

# Algoritmos de deteção de Objetos

## Trabalho Prático – Fase 3

Introdução à Visão por Computador

Ruben Faria – nº17010

Ademar Valente – nº23155

**Docente** José Henrique Brito

Licenciatura em Engenharia em Desenvolvimento de Jogos Digitais

*Dezembro 2022*



## Índice de Conteúdos

Índice de figuras.....	2
1 Introdução .....	3
Contextualização.....	3
Estrutura do relatório.....	4
2 Desenvolvimento .....	5
Matriz teórica aplicada: implementação.....	5
Estrutura de dados.....	7
Testes efetuados na fase de implementação .....	9
3 Conclusão .....	11
Avaliação de objetivos propostos alcançados .....	11
Análise critico-reflexiva .....	11
4 Bibliografia.....	13
5 Anexos .....	14
Anexo I – Código com implementação da solução - Fase 1.....	14

## Índice de figuras

Figura 1 - Capa do jogo Super Breakout - Atari2600 (1976).....	3
Figura 2 - Features identificadores. ....	6
Figura 3 - Modelos de treinamento do haarcascade. ....	7
Figura 4 -Controlador do Paddle.....	9
Figura 5 - Resultado final.....	10
Figura 6 - Objetivos propostos no enunciado.....	11

# 1 Introdução

## Contextualização

O presente relatório procura novamente descrever de forma clara e estruturada o trabalho prático desenvolvido para responder ao terceiro desafio proposto na disciplina Introdução à Visão por Computador, integrada no 2º ano do curso de Engenharia de Desenvolvimento de Jogos Digitais. Procura apresentar a solução final com sucesso, bem como o caminho escolhido para lá chegar; e as dificuldades sentidas durante todo o processo.

Tal como o descrito no enunciado do trabalho prático, fornecido pelo docente, a resolução e implementação assenta na linguagem de programação Python, e é usado o OpenCV e as ferramentas aqui para presentes para levar o processamento e análise de imagem de encontro à resolução do problema.

O trabalho prático circunda à volta de um jogo de computador: o *Breakout*, um clássico dos Videojogos que apresenta o seu auge por volta da década de 70 pelas mãos da Atari, lançado para as máquinas de Arcade e para a consola Atari 2600.

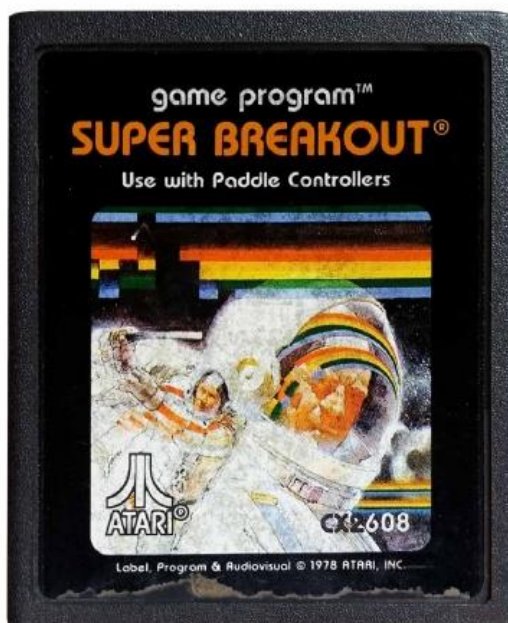


Figura 1 - Capa do jogo Super Breakout - Atari2600 (1976).

Para implementar os desafios propostos pela docência seguimos o código-fonte presente numa das hiperligações do conteúdo programático da disciplina (IVC – 0.1 – DevEnv – slide 11), abaixo indicado:

<https://www.studytonight.com/tkinter/brick-breaker-game-using-tkinter-python-project>

## Estrutura do relatório

O relatório é assim composto por 3 partes:

- Introdução, com nota inicial para contextualização do tema principal, metodologia e linguagem utilizada;
- Desenvolvimento, com implementação dos procedimentos que dão solução para o problema proposto passo a passo, estrutura de dados adotada para a solução e sua integração no código-fonte, e testes efetuados ao longo do processo de resolução do problema;
- Conclusão, com abordagem aos objetivos que foram propostos e alcançados, e análise crítico-reflexiva sobre as dificuldades sentidas e aspetos a melhorar.

Para além dos passos apresentados, o presente relatório tem também componentes estruturais relevantes para a organização do projeto escrito e fluidez da leitura do mesmo; que são o índice, a bibliografia e os anexos.

## 2 Desenvolvimento

### Matriz teórica aplicada: implementação

Neste capítulo procuraremos mais uma vez demonstrar a aplicabilidade dos conteúdos teóricos dados em aula para a conclusão deste trabalho prático. Para concluir o desafio dado para este trabalho, fomos buscar ferramentas utilizadas anteriormente para a realização das outras fases do trabalho prático, e introduzir novas, relacionadas com deteção de objetos. Mais uma vez, e após uma análise das ferramentas que nos foram disponibilizadas para concluir com sucesso o desafio, e estudar as hipóteses que nos são disponibilizadas pela biblioteca OpenCV; foram escolhidas as apresentadas abaixo para realizar o desafio da melhor forma.

#### **Captação e tratamento de imagem, e criação de janela de controlo (cv2.VideoCapture, cv2.flip)**

Este primeiro passo é sempre fundamental para que possamos ligar o controlo do nosso *paddle* com a imagem captada em tempo real pela *webcam*. Com a implementação da imagem captada pela câmara podemos executar as operações de controlo do *paddle* do jogo através de um objeto escolhidos por nós.

Associado a este passo procedemos à criação de uma janela que terá a imagem captada pela câmara (à qual demos o nome de “Play Smiling!”). A janela será inicializada em simultâneo com o jogo após um clique no rato.

Passado este passo, e por forma a agilizar e harmonizar a jogabilidade, iremos aplicar uma inversão total de píxeis na horizontal, por forma a que quando movimentarmos o nosso objeto para a esquerda, ele se mova para a direita, e vice-versa.

#### **Formatação de imagem e espaços de cor: melhorar a eficiência no cálculo de deteção de objetos (cv2.cvtColor)**

No passo seguinte procuramos simplificar a imagem na sua leitura e obtemos isso convertendo a imagem para uma escala de cinzentos. Desta forma, a formulação de cálculos será mais simples, mais fácil, mais rápida e eficiente.

Em alguma da literatura encontrada este passo é um pouco controverso, mas é usado na maioria das vezes na deteção de objetos, alegando que suaviza a imagem.

**Deteção de Objetos: usar uma face como controlador e identificar em tempo real (cv2.CascadeClassifier e cv2.Rectangle)**

Finalmente, o último bloco de processos deste trabalho insere a proposta feita no enunciado: a de controlar o nosso *paddle* do jogo com um objeto específico, bem identificado e delineado. Depois de estudar as possibilidades que nos foram dadas na sala de aula da disciplina, optamos por escolher a face como controlador principal nesta fase 3.

A deteção de objetos usada, o método de Haar; consiste em utilizar um conjunto enorme de *features* (ou pontos de interesse de uma imagem) retirado de uma função que é treinada para identificar uma face através do treinamento.

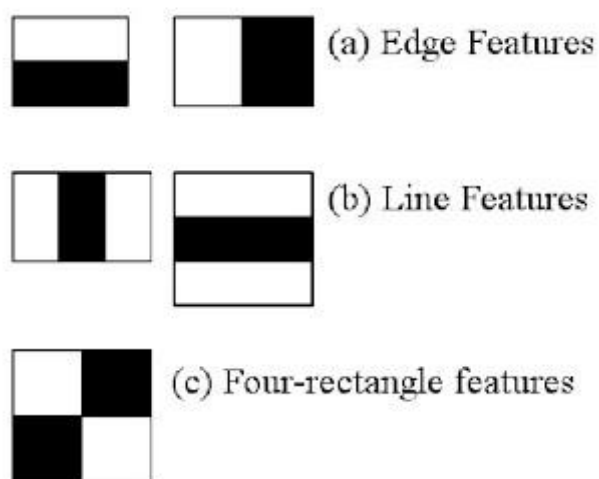


Figura 2 - Features identificadores.

O que isto quer dizer é que este método, proposto por Paul Viola e Michael Jones em 2001, trouxe uma ferramenta que foi treinada através de leitura de centenas de milhar de imagens com e sem faces, retira delas pontos de interesse chave, e obtém informação suficiente para depois em tempo real identificar com relativa rapidez uma face (ou outro objeto dentro dos que a biblioteca nos oferece). Este processo de identificação tem inúmeras funcionalidades no



nosso quotidiano, e pode até ser implementado individualmente, com treinamento através de uma recolha de imagens do que se quer depois na implementação detetar (Nota: quanto maior o número de imagens recolhidas, maior o número de *features*, melhor a identificação).

> Este PC > Windows (C:) > Utilizadores > adema > anaconda3 > envs > IVC > Library > etc > haarcascades

Nome	Data de modificação	Tipo	Tamanho
haarcascade_eye.xml	05/06/2022 16:32	Documento XML	334 KB
haarcascade_eye_tree_eyeglasses.xml	05/06/2022 16:32	Documento XML	588 KB
haarcascade_frontalcatface.xml	05/06/2022 16:32	Documento XML	402 KB
haarcascade_frontalcatface_extended.xml	05/06/2022 16:32	Documento XML	374 KB
haarcascade_frontalface_alt.xml	05/06/2022 16:32	Documento XML	661 KB
haarcascade_frontalface_alt_tree.xml	05/06/2022 16:32	Documento XML	2 627 KB
haarcascade_frontalface_alt2.xml	05/06/2022 16:32	Documento XML	528 KB
haarcascade_frontalface_default.xml	05/06/2022 16:32	Documento XML	909 KB
haarcascade_fullbody.xml	05/06/2022 16:32	Documento XML	466 KB
haarcascade_lefteye_2splits.xml	05/06/2022 16:32	Documento XML	191 KB
haarcascade_licence_plate_rus_16stages.x...	05/06/2022 16:32	Documento XML	47 KB
haarcascade_lowerbody.xml	05/06/2022 16:32	Documento XML	387 KB
haarcascade_profileface.xml	05/06/2022 16:32	Documento XML	810 KB
haarcascade_righteye_2splits.xml	05/06/2022 16:32	Documento XML	192 KB
haarcascade_russian_plate_number.xml	05/06/2022 16:32	Documento XML	74 KB
haarcascade_smile.xml	05/06/2022 16:32	Documento XML	185 KB
haarcascade_upperbody.xml	05/06/2022 16:32	Documento XML	768 KB

Figura 3 - Modelos de treinamento do haarcascade.

No final da identificação passamos a implementar um processo que demonstre isso mesmo, e que possa ser o veículo para comandar o nosso *paddle* no jogo. Delineamos a imagem com um retângulo, e definimos o seu centro como controlador.

## Estrutura de dados

Para a correta implementação das funcionalidades acima descritas, fizemos introduções no código-fonte do jogo que nos foi fornecido pela docência. A estrutura escolhida procura obedecer à estrutura original, não rompendo com a tentativa de harmonizar a leitura do código e da linguagem utilizada.

De um modo geral, fizemos a introdução de uma classe para os procedimentos acima enunciados (**class FaceJoystick**), e outra para enumerar os processos relacionados com a

orientação do *paddle* após a implementação desses mesmos procedimentos (**class Screen (Enum)**).

Dentro da classe FaceJoystick foram criados os seguintes métodos e variáveis:

- **def \_\_init\_\_(self);** em que é criada uma variável “*cap*” responsável pelas ações responsáveis pela captação de imagem, e cria a janela de controlo (“Play Smiling!”);
- **def open\_window;** que inicializa a utilização da câmara. Por seu lado, **def destroy\_window** finaliza a variável “*cap*”;
- **def face\_on\_detection;** onde se invertem todos os pixéis da imagem na horizontal (cv2.flip) e onde se insere a implementação das funcionalidades de **Formatação de imagem e espaços de cor** e **Deteção de Objetos** enunciados em cima;
- **def get\_box;** que nos dá a área da caixa que deteta a face.

Para além disso, está também inserido o código responsável por associar a deteção do movimento da(s) face(s) encontrada e delineada, como podemos observar na imagem abaixo.

```
bounding_box_center = [0, 0]
bounding_box = self.get_box(faces)

if len(bounding_box) == 0:
    return Screen.CENTER

x = bounding_box[0]
y = bounding_box[1]
width = bounding_box[2]
height = bounding_box[3]

x_point_o = width / 2
y_point_o = height / 2
bounding_box_center[0] = x + x_point_o
bounding_box_center[1] = y + y_point_o

start_point = (x, y)
end_point = (x + width, y + height)

cv2.rectangle(image, start_point, end_point, (0, 0, 0), 2)

pixel_threshold = 20
image_x_center = image.shape[1] / 2

cv2.imshow(self.window_name, image)
```

```
if image_x_center - pixel_threshold > bounding_box_center[0]:  
    return Screen.LEFT  
elif image_x_center + pixel_threshold < bounding_box_center[0]:  
    return Screen.RIGHT  
else:  
    return Screen.CENTER
```

Figura 4 -Controlador do Paddle.

### Testes efetuados na fase de implementação

Ao longo do processo de execução para resolver o problema que nos foi proposto, foram precisas verificações e afinações até ao resultado ser o pretendido. Tivemos aqui várias afinações que foram sendo feitas, até que se atingisse um resultado. Tivemos muitas dificuldades ao longo da realização do trabalho, e inclusivamente tentamos outros conjuntos de *features* lecionados, tendo optado pela leitura de faces que nos dá o ficheiro “/haarcascade\_frontalface\_alt.xml”.

Mais uma vez, para a realização do trabalho procuramos potenciar a leitura da imagem com ferramentas de melhoria das funcionalidades introduzidas. Neste caso a utilização da conversão de cores procurou facilitar os cálculos necessários para a deteção das faces, e a redução para duas dimensões procurou isso mesmo. Após alguns testes, verificamos (apesar de não ser claramente) que usar uma escala de cinzentos potenciou a leitura do nosso controlador.

Passando este momento, e tendo as bases dos trabalhos anteriores, tornou-se mais simples obter um resultado favorável. A aplicabilidade dos exemplos propostos em aula foi praticamente direta, apenas se procurou encontrar o melhor método possível (foram feitos testes com olhos e faces, e escolhida a última), identificar a sua funcionalidade; e executar.

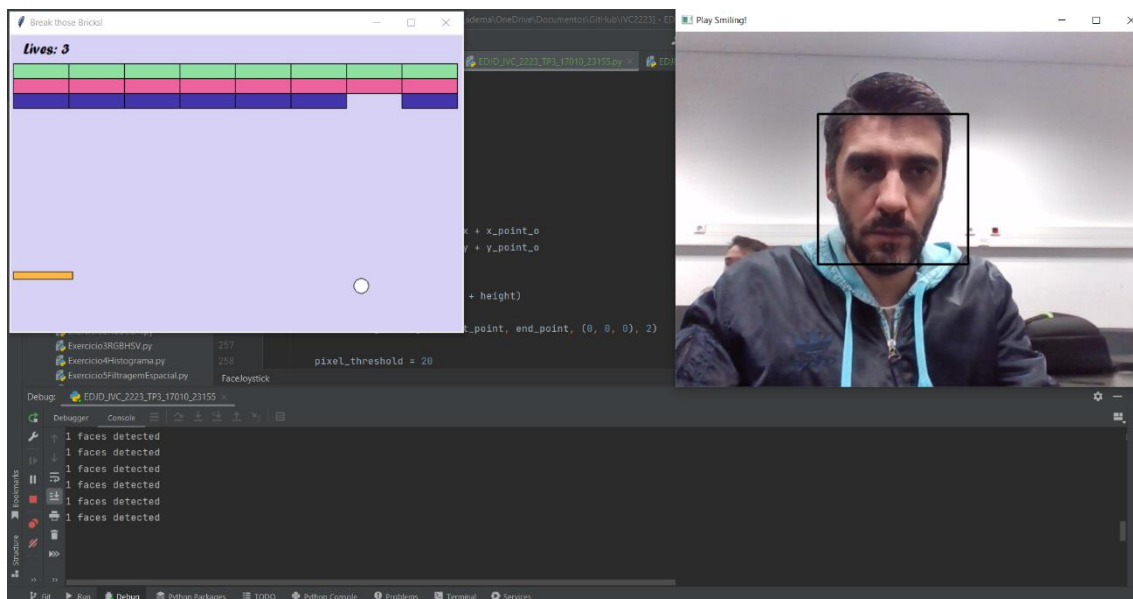


Figura 5 - Resultado final.

### 3 Conclusão

#### Avaliação de objetivos propostos alcançados

Finalizado o trabalho, é importante avaliar se todos os momentos propostos pela docência foram realizados. Assim, de forma muito sucinta, analisando o enunciado do projeto podemos retirar as seguintes ilações de forma esquematizada:







Fase 3 – resultados	
Controlar o <i>paddle</i> através da câmara.	
Apresentar a janela do jogo e outra janela de imagem.	
Usar o rato para iniciar os algoritmos utilizados.	
O controlo do <i>paddle</i> deve ser baseado em algoritmos de deteção de objetos.	
Pode ser aplicado um dos algoritmos de deteção de objetos abordado em aula.	
O controlo do <i>paddle</i> deve basear-se na posição do objeto detetado na imagem.	

Figura 6 - Objetivos propostos no enunciado.

#### Análise critico-reflexiva

Podemos concluir que os pressupostos para esta fase do trabalho foram concluídos com sucesso. Mais uma vez a análise profunda dos conceitos teóricos apreendidos nas aulas, bem como todo o volume prático efetuado nas mesmas; foi crucial para que o resultado tivesse uma aplicabilidade bem-sucedida.

A implementação da deteção de Objetos e o projeto de Viola e Jones, dadas pelo OpenCV, deram-nos oportunidade para programar uma solução sólida, eficaz, funcional e simples de ser entendida, mesmo por trás de todas as equações matemáticas de difícil perceção que a teoria encerra.

Verificamos que o `cv2.CascadeClassifier` nos dá um controlo aceitável, desde que não existam mais do que uma face no raio da nossa câmara. Este foi talvez o resultado menos desejável, que não nos permite dizer que existe um controlo total do *paddle*.

No final, estamos mais uma vez entusiasmados com o resultado que apresentamos. A aplicabilidade visual neste tipo de trabalhos é bastante apelativa, e abre portas para conceitos que na área dos videojogos se revela bastante importante. Estamos preparados para continuar a apreender novos conceitos e melhorar o que já foi feito, e conseguimos perceber no conjunto de todos os trabalhos realizados que a construção evolutiva neste projeto nos dá uma aprendizagem bastante apoiada; quer pela repetição de conceitos (que nos permite um maior controlo dos mesmos), quer pelo crescendo gradual da complexidade acontecer a um ritmo de entendimento confortável e seguro.

## 4 Bibliografia

<https://www.studytonight.com/tkinter/brick-breaker-game-using-tkinter-python-project>

[https://docs.opencv.org/4.x/df/d54/tutorial\\_py\\_features\\_meaning.html](https://docs.opencv.org/4.x/df/d54/tutorial_py_features_meaning.html)

[https://docs.opencv.org/4.x/db/d28/tutorial\\_cascade\\_classifier.html](https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html)

<https://www.geeksforgeeks.org/face-detection-using-python-and-opencv-with-webcam/>

<https://realpython.com/face-detection-in-python-using-a-webcam/>

Material fornecido pelo docente (Moodle)

## 5 Anexos

### Anexo I – Código com implementação da solução - Fase 1

No presente anexo apresentamos o código elaborado, anexado ao código-fonte.

```
import tkinter as tk
from enum import Enum
import cv2

class GameObject(object):
    def __init__(self, canvas, item):
        self.canvas = canvas
        self.item = item

    def get_position(self):
        return self.canvas.coords(self.item)

    def move(self, x, y):
        self.canvas.move(self.item, x, y)

    def delete(self):
        self.canvas.delete(self.item)

class Ball(GameObject):
    def __init__(self, canvas, x, y):
        self.radius = 10
        self.direction = [1, -1]
        # increase the below value to increase the speed of ball
        self.speed = 5
        item = canvas.create_oval(x - self.radius, y - self.radius,
                                   x + self.radius, y + self.radius,
                                   fill='white')
        super(Ball, self).__init__(canvas, item)

    def update(self):
        coords = self.get_position()
        width = self.canvas.winfo_width()
        if coords[0] <= 0 or coords[2] >= width:
            self.direction[0] *= -1
        if coords[1] <= 0:
            self.direction[1] *= -1
        x = self.direction[0] * self.speed
        y = self.direction[1] * self.speed
        self.move(x, y)

    def collide(self, game_objects):
        coords = self.get_position()
        x = (coords[0] + coords[2]) * 0.5
        if len(game_objects) > 1:
            self.direction[1] *= -1
        elif len(game_objects) == 1:
```



```

        game_object = game_objects[0]
        coords = game_object.get_position()
        if x > coords[2]:
            self.direction[0] = 1
        elif x < coords[0]:
            self.direction[0] = -1
        else:
            self.direction[1] *= -1

    for game_object in game_objects:
        if isinstance(game_object, Brick):
            game_object.hit()

class Paddle(GameObject):
    def __init__(self, canvas, x, y):
        self.width = 80
        self.height = 10
        self.ball = None
        item = canvas.create_rectangle(x - self.width / 2,
                                       y - self.height / 2,
                                       x + self.width / 2,
                                       y + self.height / 2,
                                       fill='#FFB643')
        super(Paddle, self).__init__(canvas, item)

    def set_ball(self, ball):
        self.ball = ball

    def move(self, offset):
        coords = self.get_position()
        width = self.canvas.wininfo_width()
        if coords[0] + offset >= 0 and coords[2] + offset <= width:
            super(Paddle, self).move(offset, 0)
            if self.ball is not None:
                self.ball.move(offset, 0)

class Brick(GameObject):
    COLORS = {1: '#4535AA', 2: '#ED639E', 3: '#8FE1A2'}

    def __init__(self, canvas, x, y, hits):
        self.width = 75
        self.height = 20
        self.hits = hits
        color = Brick.COLORS[hits]
        item = canvas.create_rectangle(x - self.width / 2,
                                       y - self.height / 2,
                                       x + self.width / 2,
                                       y + self.height / 2,
                                       fill=color, tags='brick')
        super(Brick, self).__init__(canvas, item)

    def hit(self):
        self.hits -= 1
        if self.hits == 0:
            self.delete()
        else:
            self.canvas.itemconfig(self.item,
                                   fill=Brick.COLORS[self.hits])

```

```
class Game(tk.Frame):
    def __init__(self, master):
        super(Game, self).__init__(master)
        self.lives = 3
        self.width = 610
        self.height = 400
        self.canvas = tk.Canvas(self, bg='#D6D1F5',
                                width=self.width,
                                height=self.height, )

        self.canvas.pack()
        self.pack()

        self.items = {}
        self.ball = None
        self.movement_detection = FaceJoystick()
        self.paddle = Paddle(self.canvas, self.width / 2, 326)
        self.items[self.paddle.item] = self.paddle
        # adding brick with different hit capacities - 3,2 and 1
        for x in range(5, self.width - 5, 75):
            self.add_brick(x + 37.5, 50, 3)
            self.add_brick(x + 37.5, 70, 2)
            self.add_brick(x + 37.5, 90, 1)

        self.hud = None
        self.setup_game()
        self.canvas.focus_set()

    def setup_game(self):
        self.movement_detection.destroy_window()
        self.add_ball()
        self.update_lives_text()
        self.text = self.draw_text(300, 200,
                                    'Click mouse to start')
        self.canvas.bind('<Button-1>', lambda _: self.start_game())

    def add_ball(self):
        if self.ball is not None:
            self.ball.delete()
        paddle_coords = self.paddle.get_position()
        x = (paddle_coords[0] + paddle_coords[2]) * 0.5
        self.ball = Ball(self.canvas, x, 310)
        self.paddle.set_ball(self.ball)

    def add_brick(self, x, y, hits):
        brick = Brick(self.canvas, x, y, hits)
        self.items[brick.item] = brick

    def draw_text(self, x, y, text, size='40'):
        font = ('Forte', size)
        return self.canvas.create_text(x, y, text=text,
                                       font=font)

    def update_lives_text(self):
        text = 'Lives: %s' % self.lives
        if self.hud is None:
            self.hud = self.draw_text(50, 20, text, 15)
        else:
            self.canvas.itemconfig(self.hud, text=text)
```

```
def start_game(self):
    self.movement_detection.open_window()
    self.canvas.unbind('<Button-1>')
    self.canvas.delete(self.text)
    self.paddle.ball = None
    self.game_loop()

def game_loop(self):

    screen_move = self.movement_detection.face_on_detection

    if screen_move == Screen.LEFT:
        self.paddle.move(-10)
    elif screen_move == Screen.RIGHT:
        self.paddle.move(10)

    self.check_collisions()
    num_bricks = len(self.canvas.find_withtag('brick'))
    if num_bricks == 0:
        self.ball.speed = None
        self.draw_text(300, 200, 'You win! You the Breaker of
Bricks.')
```

```
    elif self.ball.get_position()[3] >= self.height:
        self.ball.speed = None
        self.lives -= 1
        if self.lives < 0:
            self.movement_detection.destroy_window()
            self.draw_text(300, 200, 'You Lose! Game Over!')
        else:
            self.after(1000, self.setup_game)
    else:
        self.ball.update()
        self.after(50, self.game_loop)

def check_collisions(self):
    ball_coords = self.ball.get_position()
    items = self.canvas.find_overlapping(*ball_coords)
    objects = [self.items[x] for x in items if x in self.items]
    self.ball.collide(objects)

class FaceJoystick:

    def __init__(self):
        self.cap = cv2.VideoCapture()
        self.window_name = "Play Smiling!"

    def open_window(self):
        if not self.cap.isOpened():
            self.cap.open(0)

    @property
    def face_on_detection(self):
        cap = self.cap
        ret, image = cap.read()
        image = cv2.flip(image, 1)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        face_cascade =
```

```

cv2.CascadeClassifier('C:/Users/adema/anaconda3/envs/IVC/Library/etc/h
aarcascades'

'/haarcascade_frontalface_alt.xml')
    faces = face_cascade.detectMultiScale(gray)
    print("{} faces detected".format(len(faces)))

    bounding_box_center = [0, 0]
    bounding_box = self.get_box(faces)

    if len(bounding_box) == 0:
        return Screen.CENTER

    x = bounding_box[0]
    y = bounding_box[1]
    width = bounding_box[2]
    height = bounding_box[3]

    x_point_o = width / 2
    y_point_o = height / 2
    bounding_box_center[0] = x + x_point_o
    bounding_box_center[1] = y + y_point_o

    start_point = (x, y)
    end_point = (x + width, y + height)

    cv2.rectangle(image, start_point, end_point, (0, 0, 0), 2)

    pixel_threshold = 20
    image_x_center = image.shape[1] / 2

    cv2.imshow(self.window_name, image)

    if image_x_center - pixel_threshold > bounding_box_center[0]:
        return Screen.LEFT
    elif image_x_center + pixel_threshold <
bounding_box_center[0]:
        return Screen.RIGHT
    else:
        return Screen.CENTER

    @staticmethod
    def get_box(faces):
        max_area = 0
        bounding_box = []

        for (x, y, width, height) in faces:

            current_area = width * height
            if max_area > current_area:
                continue

            bounding_box = [x, y, width, height]

        return bounding_box

    def destroy_window(self):
        cap = self.cap

        if cap.isOpened():

```

```
cap.release()
cv2.destroyAllWindows(self.window_name)

class Screen(Enum):
    LEFT = 0
    RIGHT = 1
    CENTER = 2

if __name__ == '__main__':
    root = tk.Tk()
    root.title('Break those Bricks!')
    game = Game(root)
    game.mainloop()
```