



INSTITUTO  
POLITÉCNICO  
DO CÁVADO  
E DO AVE



ESCOLA  
SUPERIOR  
DE TECNOLOGIA  
IPCA

# Algoritmos de Segmentação

## **Trabalho Prático – Fase 1**

Introdução à Visão por Computador

Ruben Faria – nº17010

Ademar Valente – nº23155

**Docente** José Henrique Brito

**Licenciatura em Engenharia em Desenvolvimento de Jogos Digitais**

*Novembro 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 .....	8
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 - Equação dos valores Kernell utilizados. ....	6
Figura 3 - Representação de Segmentação por Thresholding.....	7
Figura 4 - Equação da funcionalidade Momentos em (x,y).....	8
Figura 5 – Resultado final vs. Threshold usado (mask). ....	9
Figura 6 – Resultado final vs. Filtro Gaussiano utilizado (blur).....	10
Figura 7 - Objetivos propostos no enunciado. ....	11

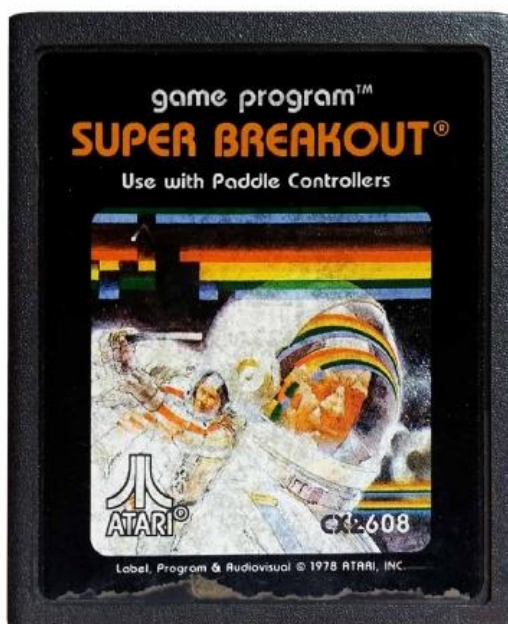
# 1 Introdução

## Contextualização

O presente relatório procura descrever de forma clara, harmoniosa e estruturada o trabalho prático desenvolvido para responder ao desafio proposto na disciplina Introdução à Visão por Computador, integrada no 2ºano 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 e a bibliografia.

## 2 Desenvolvimento

### Matriz teórica aplicada: implementação

Chegados à fase de elaboração do trabalho, torna-se importante rever os conteúdos teóricos dados e praticados em aula; por forma a tomar os nossos processos de decisão para a elaboração de uma solução sólida, rápida, pratica e eficaz. 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)**

O primeiro passo a implementar é a introdução da imagem captada pela câmara para poder executar as operações de controlo do paddle do jogo através de um objeto.

Associado a este passo procedemos à criação de uma janela que terá a imagem captada pela câmara (à qual demos o nome de “Play!”). 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 pixéis na horizontal, por forma a que quando movimentarmos o nosso objeto para a esquerda, ele se mova para a direita, e vice-versa.

#### **Filtragem: redução de ruído indesejado (cv2.Gaussianblur)**

Em seguida iremos tentar limpar a imagem, começando por isolar antecipadamente pontos que possam provocar ruído ao threshold que vamos de seguida escolher para isolar o objeto pretendido por cor.

Para esse efeito, utilizamos um filtro passa-baixo (mais concretamente gaussiano) com o objetivo de eliminar as altas frequências. O filtro vai proceder ao varrimento da imagem por um *kernell* bidimensional, no que resultará a diminuição/eliminação do ruído existente, que para o efeito é indesejável.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Figura 2 - Equação dos valores Kernell utilizados.

Na equação  $x$  e  $y$  representam o índice da localização, o  $x$  a distância do pixel central no eixo horizontal e o  $y$  a distância no eixo vertical. O  $\sigma$  representa o desvio padrão, controlando a dimensão do efeito de suavização em torno de um pixel.

Os valores utilizados na função Gaussiana foram escolhidos após uma fase de testes, e tidos como suficientes para a eliminação de ruído manter a movimentação do *paddle* limpa.

**Segmentação: conversão para HSV e a escolha de um threshold (cv2.cvtColor, cv2.inRange)**

O processo de segmentação de uma imagem consiste em dividir uma imagem em vários segmentos, segmentos esses constituídos por pixéis com características comuns.

Antes deste processo, passamos a imagem que recebemos em formato RBG para o formato HSV (Hue, Saturation, Value). Este espaço de cor é mais simples para trabalhar, e torna-se mais intuitivo para obter o espectro de cores que vamos definir como mínimo e máximo.

Após este passo, definimos um espectro mínimo e máximo que vão segmentar a nossa imagem em dois planos: o de fundo e o objeto. Definida uma cor (no nosso caso, azul; cor única da peça de Lego® que serve como objeto-controlador), e usando a função cv2.inRange() para o efeito, todos os valores vão ser de valor 0 fora da cor azul (aparecendo na visualização do threshold a preto), e 255 dentro dos valores de azul escolhidos (aparecendo na visualização a branco). Geramos assim o pretendido, uma imagem com dois planos distintos, de características binárias; que fica patente através das seguintes equações:



- $B(x,y) = 1$  (ou 255) , se  $I(x,y) > \text{Threshold}$
- $B(x,y) = 0$  , se  $I(x,y) \leq \text{Threshold}$

Figura 3 - Representação de Segmentação por Thresholding.

**Contornos: deteção de objeto-controlador após encontrar os seus contornos (cv2.findContours, cv2.drawContours, cv2.contourArea)**

Nesta parte do processo fomos à procura de isolar os contornos da imagem pretendida para o nosso controlador do *paddle*.

Os contornos são os pontos de uma imagem em que existe maior variação de intensidade. Após termos segmentado a imagem para os objetos de cor azul, é através dos contornos que procederemos ao isolamento de um único objeto capaz de ser o controlador.

Por fases, usamos a funcionalidade cv2.findContours para encontrar os contornos dos objetos dentro dos limites do threshold, em seguida desenhámos os contornos com a funcionalidade cv2.drawContours (com preenchimento para melhor observarmos o controlador, bem como a existência ou não de ruído na fase de testes).

No final isolamos dentro dos contornos obtidos pelo threshold aquele que tem maior área. Assim todos os elementos mais pequenos que possam surgir na nossa imagem serão excluídos da possibilidade de serem controlador.

**Etiquetagem: encontrar e usar o centro de massa (cv2.moments)**

Finalmente, utilizamos uma funcionalidade para correlacionar o nosso objeto com o controlo direto da orientação do *paddle* durante o jogo.

Para tal utilizamos a funcionalidade cv2.moments que permite encontrar o centro de massa do objeto que queremos utilizar, para posteriormente utilizarmos este ponto como o orientador na direção do nosso jogo.

Esta funcionalidade encontra a média da intensidade dos pixéis de uma imagem, neste caso de um contorno anteriormente definido, logo; o seu centro.

<p><i>Momentos</i>(<math>m_{pq}</math>)</p> $m_{pq} = \sum_i \sum_j x_i^p y_j^q f(i, j)$	<p><i>CentroMassa</i></p> $Cx = \frac{m_{10}}{m_{00}}; Cy = \frac{m_{01}}{m_{00}}$
--	--

Figura 4 - Equação da funcionalidade Momentos em (x,y).

## 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 ObjectJoystick**), e outra para enumerar os processos relacionados com a orientação do *paddle* após a implementação desses mesmos procedimentos (**class Screen(Enum)**).

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

- **def \_\_init\_\_(self)**; em que é criada uma variável “*cap*” responsável pelas ações responsáveis pela câmara (abrir, ler e inverter na horizontal, e fechar a câmara);
- **def open\_window** e **def destroy\_window**; que inicializa e termina a utilização da câmara respetivamente. O método open\_window é chamado à classe start\_game
- **def get\_mask**; onde estão implementadas as funcionalidades de **Filtragem** e **Segmentação** acima descritas;
- **def detect\_camera\_object** e **def get\_contourIdx**; onde estão implementadas as funcionalidades de **Contornos** acima descritas. A função detect\_camera\_object é também chamada à classe game\_loop (já presente no código-fonte) para detetar o lado em que se encontra o *paddle*;
- **def get\_contour\_center**; onde estão implementadas as funcionalidades de **Etiquetagem** acima descritas;
- **def get\_screen\_position**; onde se define o posicionamento do *paddle* através do movimento do objeto-controlador.

## 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. Neste capítulo em concreto tivemos dois momentos cuja experimentação (quer por pesquisa bibliográfica, quer por método tentativa e erro) foi mais exaustiva.

Num primeiro momento destacamos a escolha dos valores HSV para elaboração do threshold. Apesar de existir uma vasta documentação sobre os valores *Hue* a adotar para o azul na conversão de RGB para HSV, bem como uma funcionalidade para imprimir os valores de conversão (`np.uint8()`); tivemos que, por tentativa e erro, alterar os valores de saturação (S) para assim diminuir o espectro de cinzentos presente na variação de azuis.

Após vários testes, o threshold que consideramos como final e aceitável foi o abaixo apresentado:

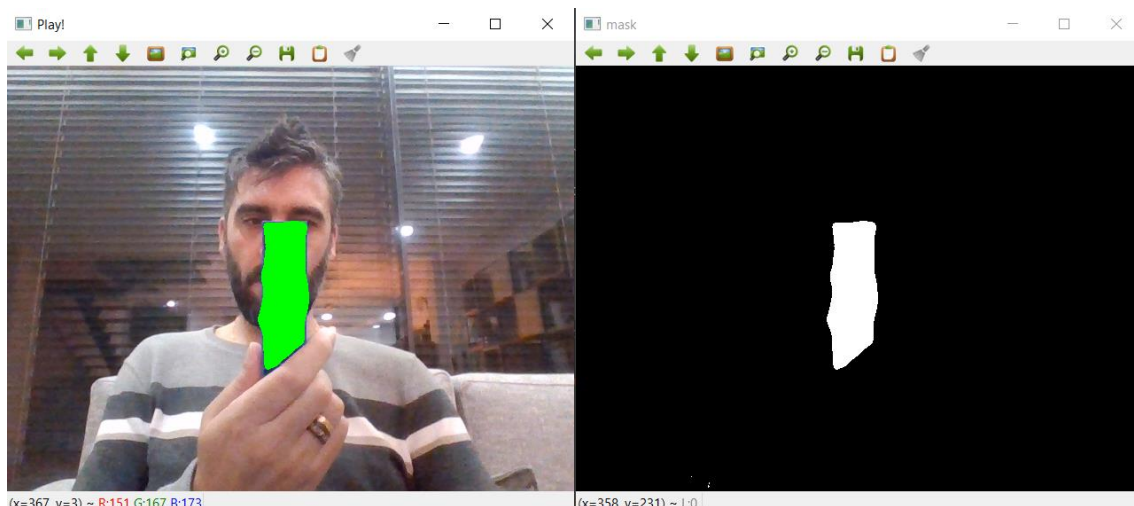
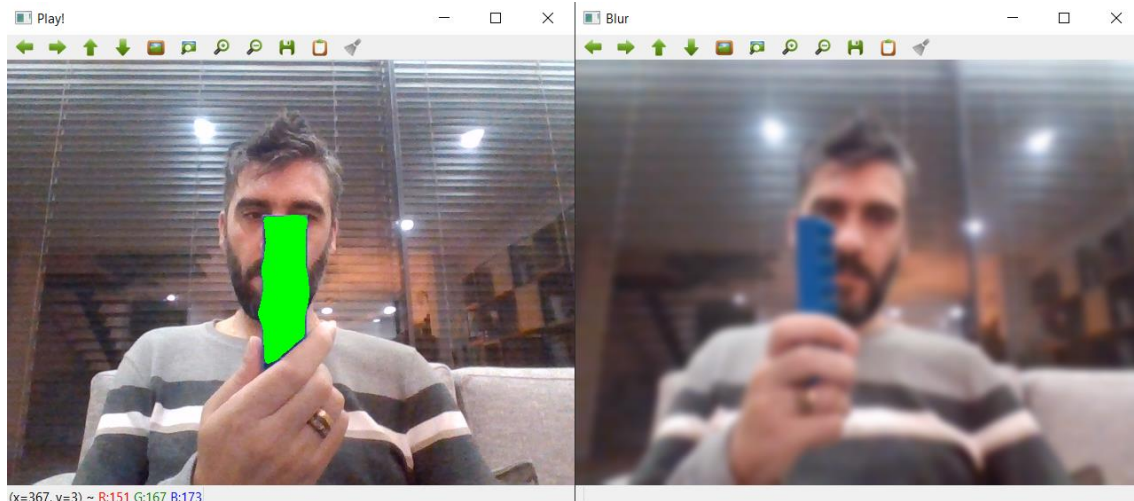


Figura 5 – Resultado final vs. Threshold usado (mask).

O outro ponto de teste importante de referir diz respeito à filtragem da imagem e a sua convolução pelo filtro passa-baixo (Gaussiano) utilizado. Este ponto foi muito importante para a eliminação do ruído da imagem, e foi sendo aumentado até ficarmos satisfeitos com o resultado; que foi o de 25x25.



*Figura 6 – Resultado final vs. Filtro Gaussiano utilizado (blur).*

### 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:







<b>Fase 1 - resultados</b>	
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 segmentação	
O objeto a ser segmentado na imagem pode ser um ou dois objetos de cor predefinida	
O controlo do <i>paddle</i> deve basear-se na posição do objeto segmentado na imagem	

Figura 7 - Objetivos propostos no enunciado.

#### Análise crítico-reflexiva

Em suma, podemos concluir que os pressupostos para esta fase do trabalho foram concluídos com sucesso. Após análise profunda dos conceitos teóricos apreendidos nas aulas, bem como todo o volume prático efetuado nas mesmas; foi bastante enriquecedor ver que a conjugação de todos os conceitos programáticos pode ter uma aplicabilidade tão marcada no propósito do nosso curso.

A conjugação da implementação das funções de Filtragem, Segmentação, Contornos e Etiquetagem que nos são dadas pelo OpenCV permitiram que tenhamos conseguido uma

solução sólida e eficaz, capaz de se manter funcional mesmo alterando o fundo da imagem adotada, com uma resposta ágil e praticamente equiparada ao controlador por teclado.

Houve, no entanto, várias dificuldades ao longo do processo de implementação das funcionalidades. Questões relacionadas com a eliminação de ruído (no que diz respeito ao acerto dos valores para convolução da imagem), encontrar os melhores valores HSV mínimos e máximos para elaboração do threshold de segmentação, e conseguir isolar através dos contornos uma só imagem que pudesse ser o controlador foram alguns dos problemas que surgiram. Estes problemas acabaram por ser debelados satisfatoriamente.

No final, resta-nos assumir que a elaboração deste projeto foi altamente enriquecedora e entusiasmante, por abrir as portas da programação a uma aplicabilidade tão visual como foi esta. Estamos preparados para continuar a apreender novos conceitos e melhorar o que já foi feito, sabendo desde já que esta primeira fase do trabalho é uma base bastante sólida de conhecimento para as fases vindouras.

## 4 Bibliografia

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

<https://machinelearningknowledge.ai/image-segmentation-in-python-opencv/>

<https://realpython.com/python-opencv-color-spaces/>

[https://docs.opencv.org/4.x/d4/d13/tutorial\\_py\\_filtering.html](https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html)

[https://docs.opencv.org/4.x/df/d9d/tutorial\\_py\\_colorspaces.html](https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html)

[https://docs.opencv.org/3.4/dd/d49/tutorial\\_py\\_contour\\_features.html](https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html)

<https://www.youtube.com/watch?v=aTC-Rc4Io0&start=693>

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. Desta forma, expomos as alterações feitas ao código-fonte a azul.

```
import tkinter as tk
from enum import Enum

import cv2
import cv2 as cv
import numpy as np

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.winfo_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.paddle = Paddle(self.canvas, self.width / 2, 326)  
        self.items[self.paddle.item] = self.paddle  
  
        self.objectDetection = ObjectJoystick()  
  
        # 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()

# self.canvas.bind('<Left>', lambda _: self.paddle.move(-10))

# self.canvas.bind('<Right>', lambda _: self.paddle.move(10))

def setup_game(self):

    self.objectDetection.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.objectDetection.open_window()

    self.canvas.unbind('<Button-1>')

    self.canvas.delete(self.text)

    self.paddle.ball = None

    self.game_loop()


def game_loop(self):

    self.objectDetection.detect_camera_object()

    screen_side = self.objectDetection.side

    if screen_side == Screen.LEFT:

        self.paddle.move(-15)

    elif screen_side == Screen.RIGHT:

        self.paddle.move(15)

    self.check_collisions()

    num_bricks = len(self.canvas.find_withtag('brick'))

    if num_bricks == 0:
```

```
        self.objectDetection.destroy_window()

        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.objectDetection.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 ObjectJoystick:
```

```
    def __init__(self):

        self.cap = cv2.VideoCapture()

        self.side = Screen.MIDDLE

        self.windowName = "Play!"

    def open_window(self):
```

```
        if not self.cap.isOpened():
            self.cap.open(0)

    def get_mask(self, image):

        blur = cv2.GaussianBlur(image, (25, 25), 0)
        # cv2.imshow("Blur", blur)

        hsv = cv2.cvtColor(blur, cv.COLOR_BGR2HSV)

        # threshold blue

        # blue testing

        # blueBGR = np.uint8([[[255, 0, 0]]])
        # hsv_blue = cv2.cvtColor(blueBGR, cv2.COLOR_BGR2HSV)
        # print(hsv_blue)

        light_blue = np.array([90, 80, 0])
        dark_blue = np.array([130, 255, 255])

        # mask = cv2.inRange(hsv, light_blue, dark_blue)
        # cv2.imshow("mask", mask)

        return cv2.inRange(hsv, light_blue, dark_blue)

    def detect_camera_object(self):
        cap = self.cap
        _, image = cap.read()
```



```
image = cv2.flip(image, 1)

mask = self.get_mask(image)

contours, _ = cv2.findContours(mask, cv.RETR_TREE, cv.CHAIN_APPROX_NONE)
contourIdx = self.get_contourIdx(contours)

cv2.drawContours(image=image, contours=contours, contourIdx=contourIdx,
color=(0, 255, 0), thickness=-1)

cv2.imshow(self.windowName, image)

if contourIdx > -1:
    self.side = self.get_screen_position(image, contours[contourIdx])
else:
    self.side = Screen.MIDDLE

def get_contourIdx(self, contours):

    countourIdx = -1
    max_area = 0

    for i in range(len(contours)):

        current_contour = contours[i]
        countour_area = cv2.contourArea(current_contour)

        if countour_area > max_area:
            max_area = countour_area
            countourIdx = i
```

```
        return countourIdx

def get_contour_center(self, contour):

    moment = cv2.moments(contour)

    x = int(moment["m10"] / moment["m00"])

    y = int(moment["m01"] / moment["m00"])

    return [x, y]

def get_screen_position(self, image, contour):

    side = Screen.MIDDLE

    contour_center = self.get_contour_center(contour)

    image_point_o = image.shape[0] / 2

    if image_point_o - 25 > contour_center[0]:

        side = Screen.LEFT

    elif image_point_o - 25 < contour_center[0]:

        side = Screen.RIGHT

    return side

def destroy_window(self):

    cap = self.cap

    if cap.isOpened():

        cap.release()

        cv2.destroyAllWindows(self.windowName)
```

```
class Screen(Enum):  
    LEFT = 0  
    MIDDLE = 1  
    RIGHT = 2  
  
if __name__ == '__main__':  
    root = tk.Tk()  
    root.title('Break those Bricks!')  
    game = Game(root)  
    game.mainloop()
```