Relatório Projeto IVC –

Fase 1

**EDJD 2022/2023**

*Este projeto serviu de prepósito a um projeto para a cadeira de Introdução a visão por computador, em python dividido em três fases.*

# Índice

[Introdução 1](#_Introdução) [Propósitos e Objetivos 2](#_Propósitos_e_Objetivos) [Fase 1 3](#_Fase_1)  [Fase 2 4](#_Fase_2) [Fase 3 5](#_Fase_3) [Desenvolvimento da Fase 1 6](#_Desenvolvimento_da_Fase)

[Conclusão 7](#_Conclusão)

# Índice de imagens

Figura 1 - Importação das libraries utilizadas para a segmentação 7

Figura 2 - Declaração do threshold inicial e início da captura de vídeo 7

Figura 3 – Função cam() parte 1 8

Figura 4 - Função cam() parte 2 8

Figura 5 - Função cam() parte 3 9

Figura 6 - Função cam() parte 4 9

Figura 7 - Função cam() parte 5 10

Figura 8 - Função cam() parte 6 10

Figura 9 - Função cam() parte 7 10

Figura 11 - formula do centroide (https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/) 11

Figura 12 - formula do x do centroide (https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/) 11

Figura 13 - formula do y do centroide (https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/) 11

Figura 14 - Função cam() parte 9 11

Figura 16 - Função cam() parte 11 11

Figura 17 - Função mouse\_get\_threshold() parte 1 12

Figura 18 - Função mouse\_get\_threshold() parte 2 12

Figura 19 - Função game\_loop() 13

## Introdução

Este trabalho foi proposto pelo professor José Brito, no contexto da unidade curricular Introdução à Visão por Computador, integrada no primeiro semestre do 2º ano de licenciatura de Desenvolvimento de Jogos Digitais, que visa o reforço e a aplicação dos conhecimentos adquiridos ao logo do semestre.

## Propósitos e Objetivos

No jogo, o jogador controla um paddle de forma que a bola destrua os tijolos da parede. O objetivo do trabalho é que o jogador controle o paddle através da câmara. O trabalho tem 3 fases. Em cada fase o controlo do paddle deve basear-se em diferentes técnicas de Visão por Computador. Em todas as fases deve ser apresentada a janela com o jogo e pelo menos outra janela com a imagem captada pela câmara e a visualização do resultado dos algoritmos de Visão por Computador. Poderão ser apresentadas várias janelas com resultados intermédios dos algoritmos de Visão por Computador. Em todas as fases, poderá ser usado o rato para inicializar os algoritmos usados. Na fase 1 o controlo do paddle 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 do paddle deve basear-se na posição do(s) objeto(s) segmentados na imagem.

### Fase 1

Na fase 1 o controlo do paddle 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 do paddle deve basear-se na posição do(s) objeto(s) segmentados na imagem.

### Fase 2

Na fase 2 o controlo do paddle deve ser baseado em algoritmos de deteção de movimento. Poderá ser aplicado qualquer algoritmo de deteção de movimento de entre os abordados na aula. O controlo do paddle deverá basear-se no movimento dos pixels da imagem.

### Fase 3

Na fase 3 o controlo do paddle deve ser baseado em algoritmos de deteção de objetos. Poderá ser aplicado qualquer algoritmo de deteção de objetos de entre os abordados na aula. O controlo do paddle deve basear-se na posição do(s) objeto(s) detetados na imagem.

## Desenvolvimento da Fase 1

Primeiramente para a resolução desta fase precisamos de uma library que nos permita utilizar e manipular a câmara, para isto usamos a library de openCV chamada cv2. Importámos também a library numpy para o uso de matrizes.

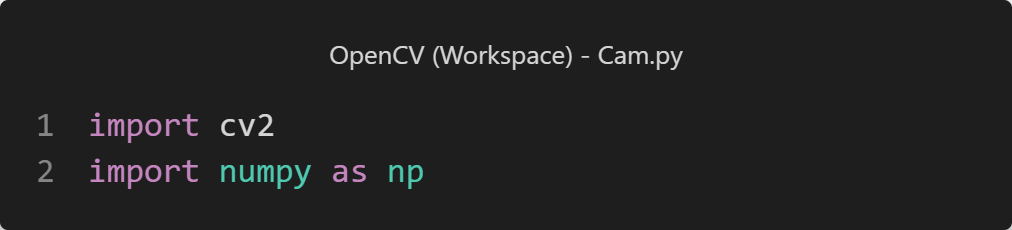


Figura 1 - Importação das libraries utilizadas para a segmentação

Começamos por definir thresholds globais e iniciá-los com um valor predefinido para a segmentação de verde.

Iniciamos também a captura de vídeo associando a mesma a uma variável global para permitir a suma manipulação.

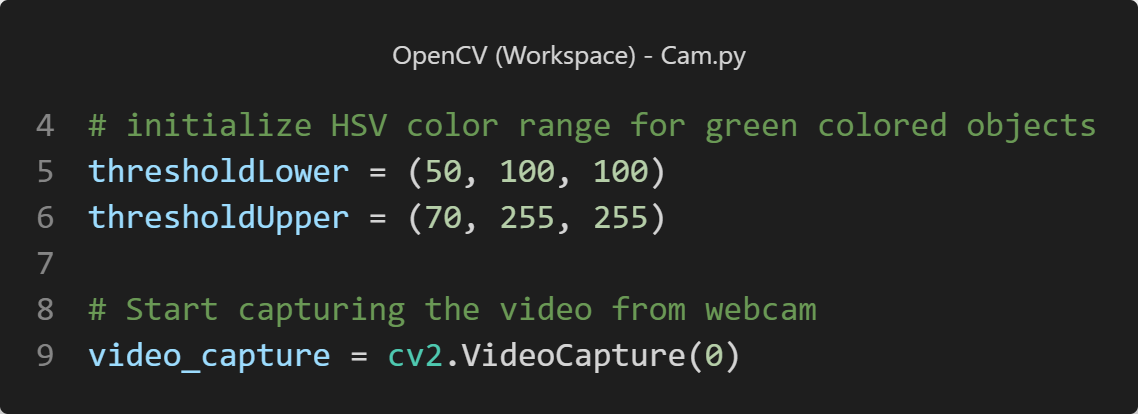


Figura 2 - Declaração do threshold inicial e início da captura de vídeo

Definimos uma função chamada cam(), esta começa por verificar se a vídeo\_capture está aberta, caso não abre a mesma. Gurdamos o frame atual na variável frame e usamos a função flip da library cv2 para evitar o efeito da imagem estar espelhada.

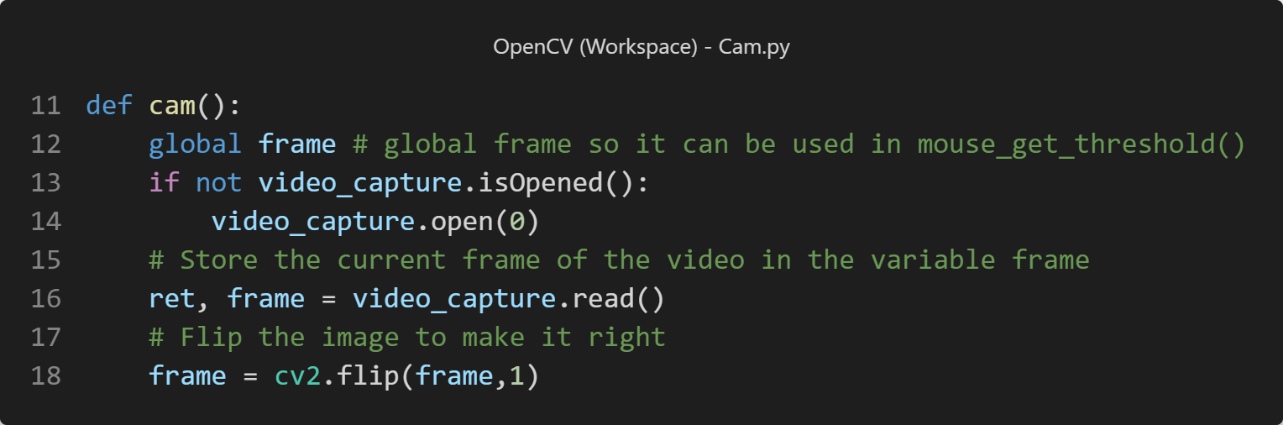


Figura 3 – Função cam() parte 1

Damos uso à função GaussianBlur da library cv2 com o kernel de tamanho 5 para remover ruído excessivo no frame. Convertemos também o frame para o formato HSV com recurso ao método cvtColor da library cv2 permitindo assim uma segmentação mais eficaz.

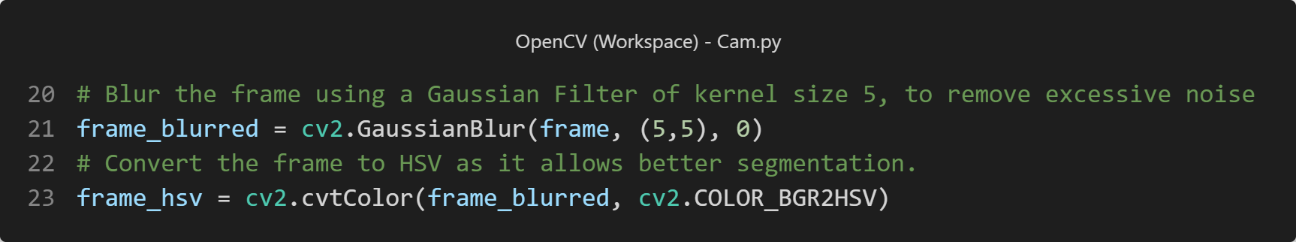


Figura 4 - Função cam() parte 2

Segmentamos o frame já convertido para HSV com a função inRange da library cv2 baseada nos valores atuais dos thresholds. Obtendo assim uma imagem onde a cor segmentada está representada a branco e as restantes a preto.

Após segmentado, erodimos o frame\_segmented para eliminar o ruído restante, obtendo assim o frame\_eroded. Este é então dilatado dando origem ao frame\_masked.

Por fim, após esta filtração, mostramos o frame\_masked na janela “Masked Output” com a intenção de dar uma visualização da segmentação ao utilizador.

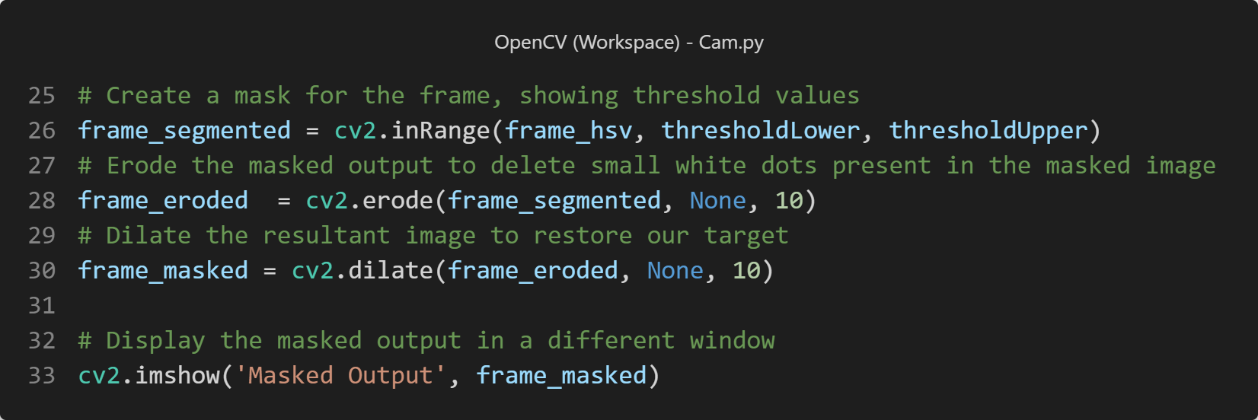


Figura 5 - Função cam() parte 3

Com a função findContours da library cv2 obtemos todas as áreas da cor segmentada e guardamos o output na variável contours.

Declaramos também o centro do objeto como noneType, para caso não haver um objeto da cor segmentada a função devolver none.

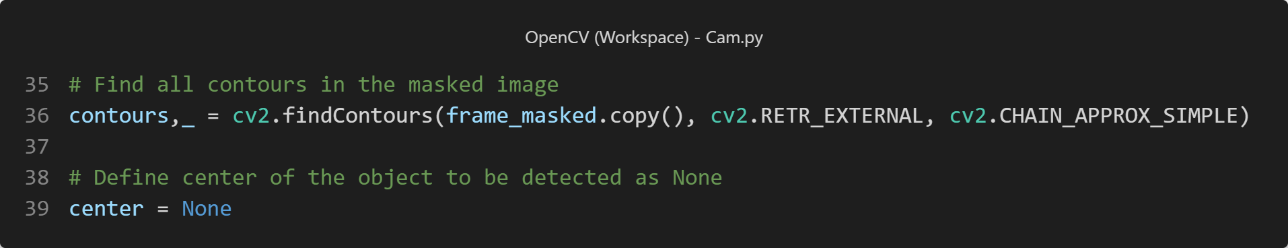


Figura 6 - Função cam() parte 4

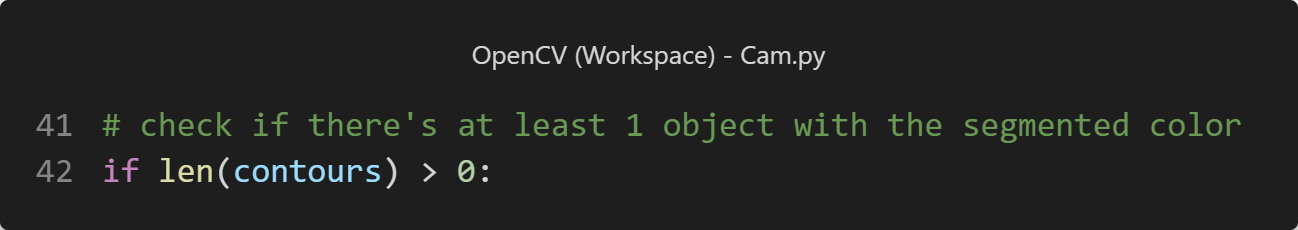
Verificamos se o comprimento da variável contours é superior a 0, ou seja, que existe um ou mais de um objeto de cor igual à da segmentação. 

Figura 7 - Função cam() parte 5

Caso não exista mostramos o frame na janela “Frame”, passamos também o output da função setMouseCallback da library cv2 para a função mouse\_get\_threshold e devolvemos o center com o valor de none.

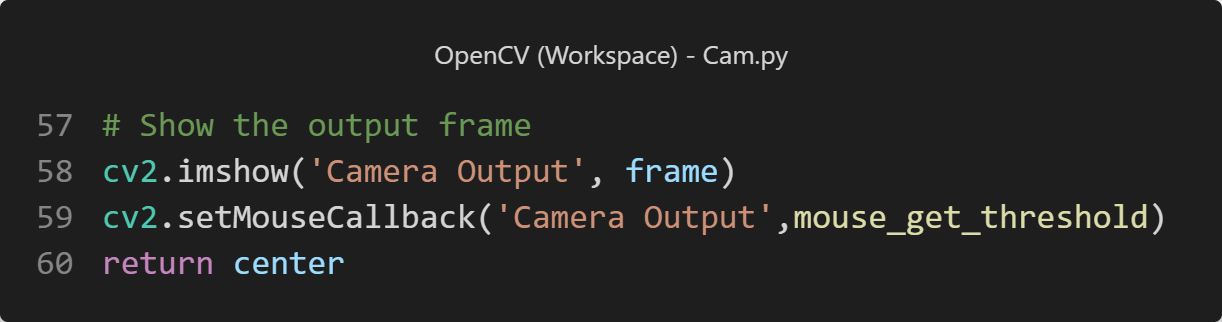


Figura 8 - Função cam() parte 6

Caso exista calculamos o objeto de maior área recorrendo à função max na variável contours, com o método contourArea da library cv2 como chave de busca para analisar apenas as áreas dos contornos.

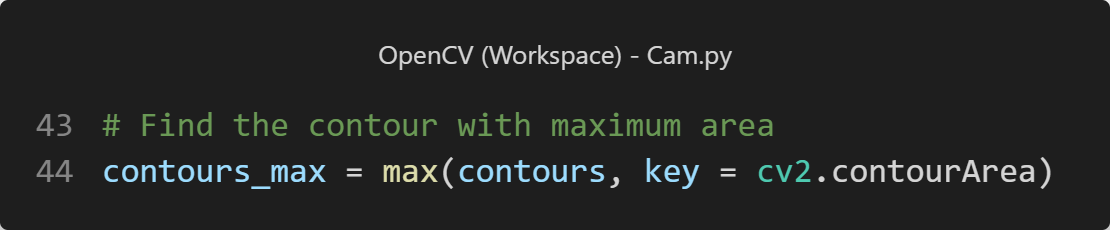


Figura 9 - Função cam() parte 7

Determinamos o retângulo de menor área que contorna a nossa área máxima, calculamos os vértices desse retângulo com recurso ao método boxPoints de modo a ter os pontos necessários para desenhar o retângulo no frame utilizando a função drawContours. Isto para ter uma referência visual que indique o objeto ou área que está a ser segmentada.Uma imagem com texto

Descrição gerada automaticamente

Após usarmos o método moments da library cv2, utilizamos o output na fórmula para encontrar o centro da área segmentada.

\[\mathbf{c} = \frac{1}{n} \sum^n_{i=1} \mathbf{x}_i\]

Figura 11 - formula do centroide (https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/)

\[C_x = \cfrac{M_{10}}{M_{00}}\]

Figura 12 - formula do x do centroide (https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/)

\[C_y = \cfrac{M_{01}}{M_{00}}\]

Figura 13 - formula do y do centroide (https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/)

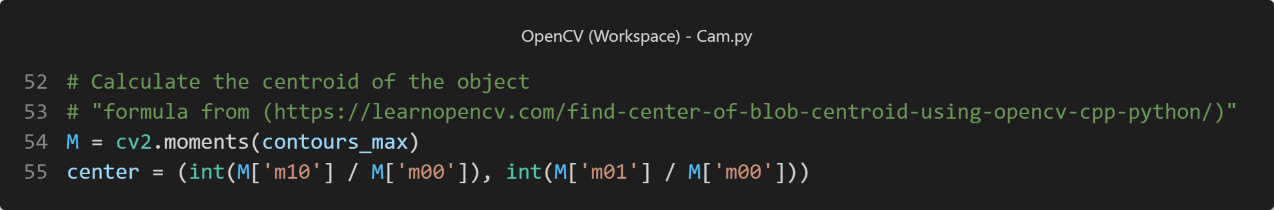


Figura 14 - Função cam() parte 9

Por fim, mostramos o frame na janela “Frame” e devolvemos o center com o valor da posição do centro do objeto.

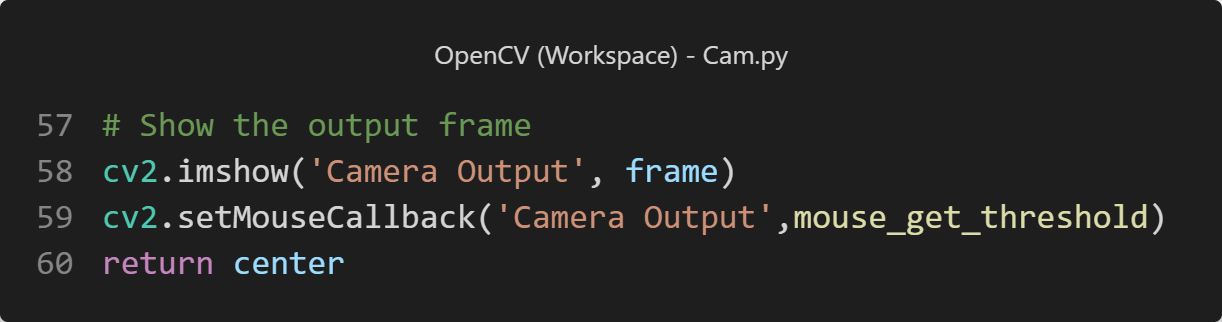


Figura 16 - Função cam() parte 11

Na função mouse\_get\_threshold verificamos se o evento registado foi um clique no botão esquerdo do rato, se sim utilizamos as coordenadas do rato na janela frame para obter os valores da cor nesse pixel e convertemos simultaneamente essa cor para o formato HSV com a função cvtColor da library cv2 e o parâmetro COLOR\_BGR2HSV.

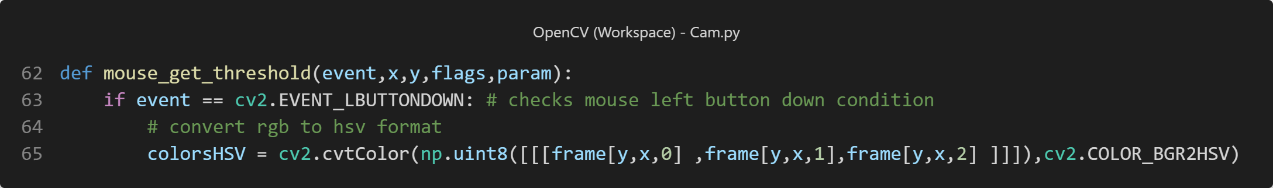


Figura 17 - Função mouse\_get\_threshold() parte 1

Com esta cor geramos novos intervalos de cor e alteramos os valores das variáveis threshold para os mesmos. Segmentando assim a partir da próxima iteração do game\_loop a nova cor selecionada.

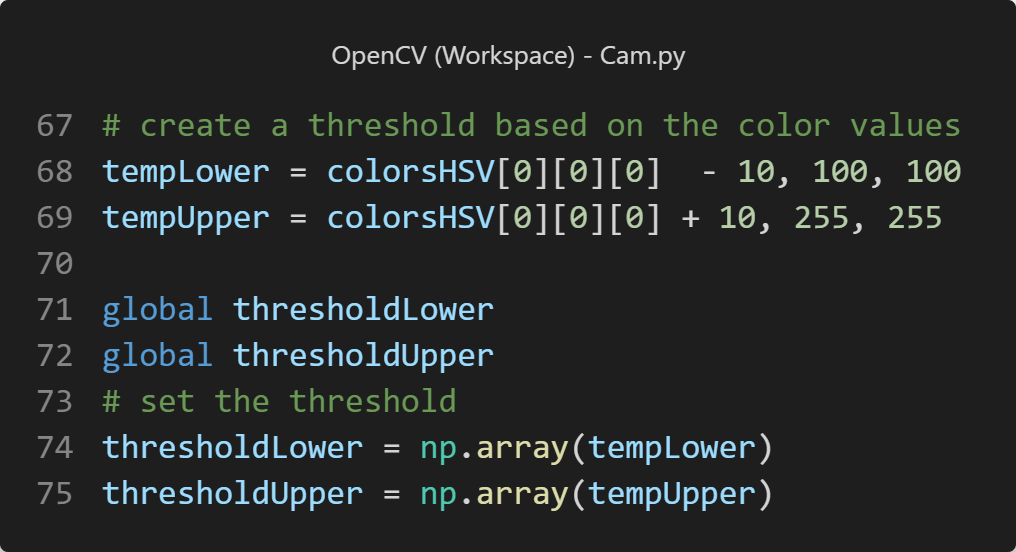


Figura 18 - Função mouse\_get\_threshold() parte 2

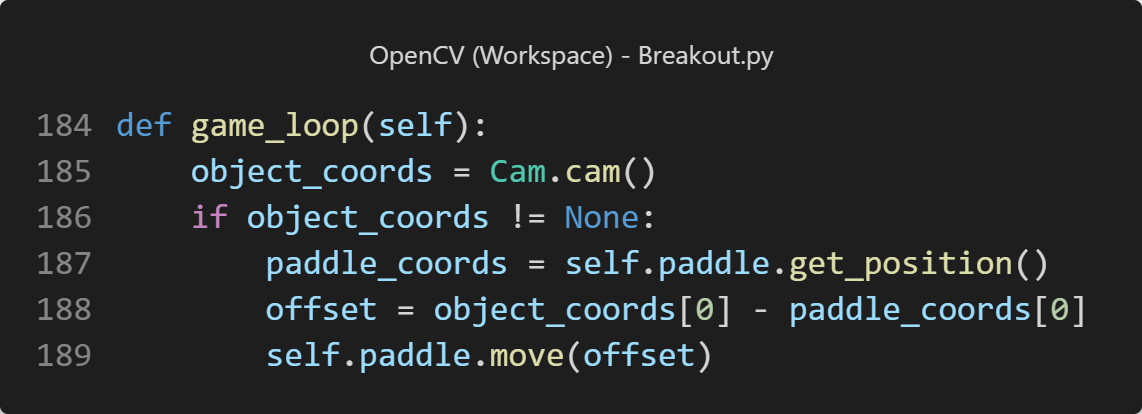
Já no código do breakout, mais especificamente na função game\_loop() da classe Game, utilizamos a função cam() do ficheiro Cam.py para obter as coordenadas do centro do objeto, após verificarmos que o centro não é nonetype usamos a função get\_position() da classe GameObject, para obter as coordenadas do objeto paddle. Tendo ambas as coordenadas declaramos que o offset é a diferença entre a coordenada X das mesmas, obtendo-se assim o valor necessário para através da função move() da classe Paddle alterar a posição do objeto paddle para coincidir com o objeto de cor igual à da segmentação.

Figura 19 - Função game\_loop()

## Conclusão

Com esta primeira fase foi possível aprofundar mais conhecimentos acerca da segmentação de uma imagem através de thresholds, e a utilização da mesma para certas aplicações. Aprendi também que fundamentalmente que ter um bom código e bem documentado é melhor que ter um código que funciona.