



INSTITUTO
POLITÉCNICO
DO CÁVADO
E DO AVE



ESCOLA
SUPERIOR
DE TECNOLOGIA
IPCA

Algoritmos de deteção de Movimento

Trabalho Prático – Fase 2

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	8
3 Conclusão	12
Avaliação de objetivos propostos alcançados	12
Análise critico-reflexiva	12
4 Bibliografia	14
5 Anexos	15
Anexo I – Código com implementação da solução - Fase 1	15

Índice de figuras

Figura 1 - Capa do jogo Super Breakout - Atari2600 (1976).	3
Figura 2 - Equação dos valores Kernell utilizados.	6
Figura 3 -Controlador do Paddle.	8
Figura 4 - Frame com filtro Gaussiano.	9
Figura 5 - Aplicação de cv2.normalize.	10
Figura 6 - Resultado.	10
Figura 7 - Print de movimentação (np.count_nonzero).	11
Figura 8 - Objetivos propostos no enunciado.	12

1 Introdução

Contextualização

O presente relatório procura novamente descrever de forma clara e estruturada o trabalho prático desenvolvido para responder ao segundo 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.

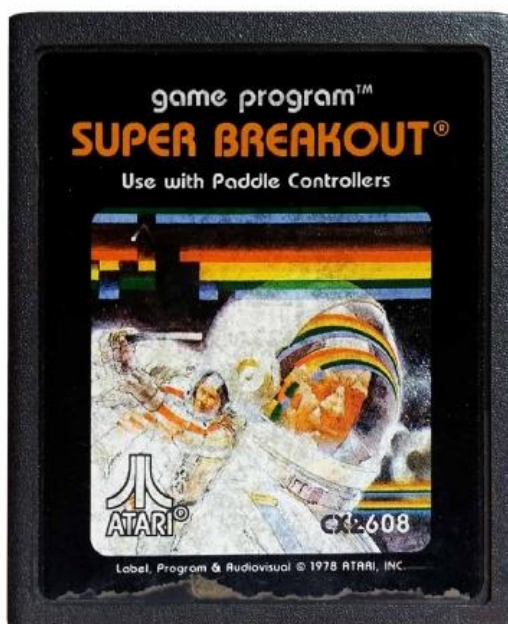


Figura 1 - Capa do jogo Super Breakout - Atari2600 (1976).

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, 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 da fase 1 do trabalho prático (algumas de forma igual, outras de forma diferente), e introduz novas, relacionadas com deteção de movimento. 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.

Captção 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 “Motion 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)

Mais uma vez usamos um filtro gaussiano para auxiliar o nosso método de controlo do *paddle*. Utilizando este filtro passa-baixo eliminamos as altas frequências, o que procura reduzir nos *frames* de deteção de movimento deteção de movimentos mais pequenos que possam interferir com o controlo do *paddle*.

Para esse efeito, o filtro vai proceder ao varrimento da imagem por um *kernell* bidimensional, no que resultará a diminuição/eliminação do ruído existente e não desejado.

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

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 σ 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* minimamente estabilizada.

Optical Flow: a técnica de Farneback (cv2.calcOpticalFlowFarneback)

Passamos então à secção que diferencia este projeto do anterior: o controlo do *paddle* do nosso jogo através dos movimentos captados pela câmara.

Voltando-nos para os conteúdos programáticos dados na disciplina verificamos que dentro dos algoritmos de deteção de movimento podemos adotar vários para realização deste projeto. Escolhemos a técnica desenvolvida por Gunnar Farneback em 2003, que no seu estudo tenta estabelecer uma equação matemática que de nos garanta com maior proximidade possível o trajeto que um movimento detetado vai ter baseando-se na comparação entre os pixéis de dois momentos de vídeo (frames). A teoria foi desenvolvida a partir de uma equação criada pelo mesmo chamada expansão polinomial, e procura estimar a movimentação de cada pixel na vizinhança pela leitura dos diferentes níveis de intensidade dos mesmos.

No sentido de otimizar este processo, fizemos uma conversão de imagem para uma escala de cinzentos e procedemos à aplicabilidade um limite (threshold) que procura reduzir movimentações de intensidade indesejada.

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

Dentro da classe MoveJoystick 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, instancia os frames de deteção de movimento, e cria a janela de controlo (“Motion Play!”);
- **def open_window**; que inicializa a utilização da câmara, lê o 1º frame, inverte a leitura no eixo XX, aplica um filtro gaussiano (implementa as funcionalidades de **Filtragem**), e converte para escala de cinzentos (cv2.cvtColor). Por seu lado, **def destroy_window** finaliza a variável “*cap*”;
- **def move_on_detection**; que inicializa a utilização da câmara, lê o 2º frame, inverte a leitura no eixo XX, converte para escala de cinzentos (cv2.cvtColor) e aplica um filtro gaussiano.

É também aqui que estão implementadas as funcionalidades de **Optical Flow** acima descritas.

Finalmente, está aqui também inserido o código responsável por associar a deteção do movimento no eixo XX, como podemos observar na imagem abaixo.

```

flow_xx = farneback[:, :, 0]

left_move = np.count_nonzero(flow_xx < -threshold)
right_move = np.count_nonzero(flow_xx > threshold)

cv2.imshow(self.window_name, frame2)

# print("Going left: " + str(left_move))
# print("Going right: " + str(right_move))
if left_move < 300:
    left_move = 0

if right_move < 300:
    right_move = 0

self.previous_gray = self.next_gray

if left_move > right_move:
    return Screen.LEFT
elif left_move < right_move:
    return Screen.RIGHT
else:
    return Screen.CENTER
  
```

Figura 3 -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 algoritmos lecionados, tendo optado pelo de Farneback.

Mais uma vez, utilizamos para a realização do trabalho um filtro que procura reduzir os ruídos indesejáveis para o controlo do nosso *paddle*. Neste caso a utilização passa-baixo procura reduzir a deteção de movimentos menores, secundários, e que não são controlados

voluntariamente por nós na leitura dos frames que vai servir para a leitura de movimento. Após alguns testes, utilizamos a mesma escala do trabalho anterior.

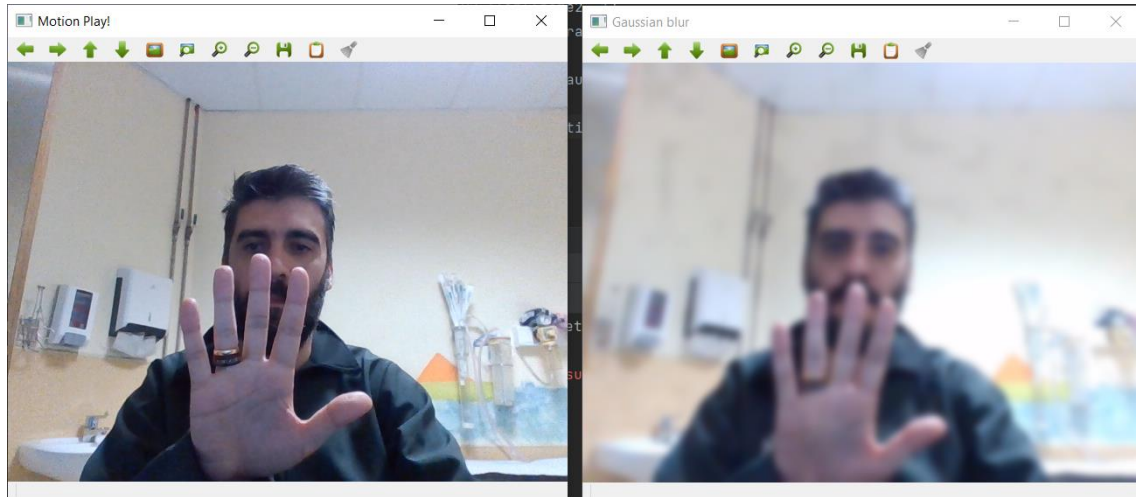


Figura 4 - Frame com filtro Gaussiano.

A seguir utilizamos o algoritmo de Farneback dado em aula para aplicar ao movimento do nosso *paddle*. Após estudo de todas as possibilidades, sentimo-nos mais confortáveis com este método e como tal, decidimo-nos pela sua implementação.

Utilizamos ainda a função `cv2.normalize` por forma a visualizar os pontos apelativos da imagem.

```
farneback = cv2.calcOpticalFlowFarneback(prev=self.previous_gray, next=self.next_gray, flow=None,
                                         pyr_scale=0.5, # 0.5,
                                         levels=1, # 3,
                                         winsize=10, # 15,
                                         iterations=1, # 3,
                                         poly_n=5,
                                         poly_sigma=1.1,
                                         flags=0)

threshold = 2.0

flow_norm = np.sqrt(farneback[:, :, 0] ** 2 + farneback[:, :, 1] ** 2)
flow_norm_norm = cv2.normalize(flow_norm, None, 0.0, 1.0, cv2.NORM_MINMAX)
cv2.imshow("Flow", flow_norm_norm)
```

Figura 5 - Aplicação de `cv2.normalize`.

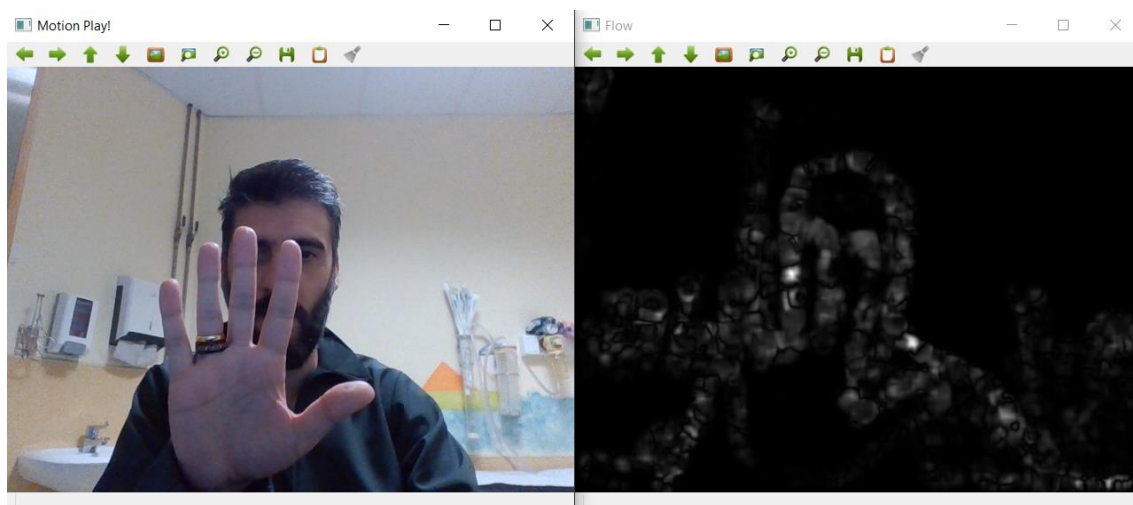


Figura 6 - Resultado.

Concluída esta fase, quisemos ainda verificar o movimento dos pixéis para esquerda e direita do ecrã, por forma a perceber a aplicabilidade da função de movimentação do paddle. A contagem dos pixéis por forma a perceber a movimentação do controlador foi importante para afinar a sensibilidade que os fatores que regulam a deteção de movimento têm (Farneback, threshold).

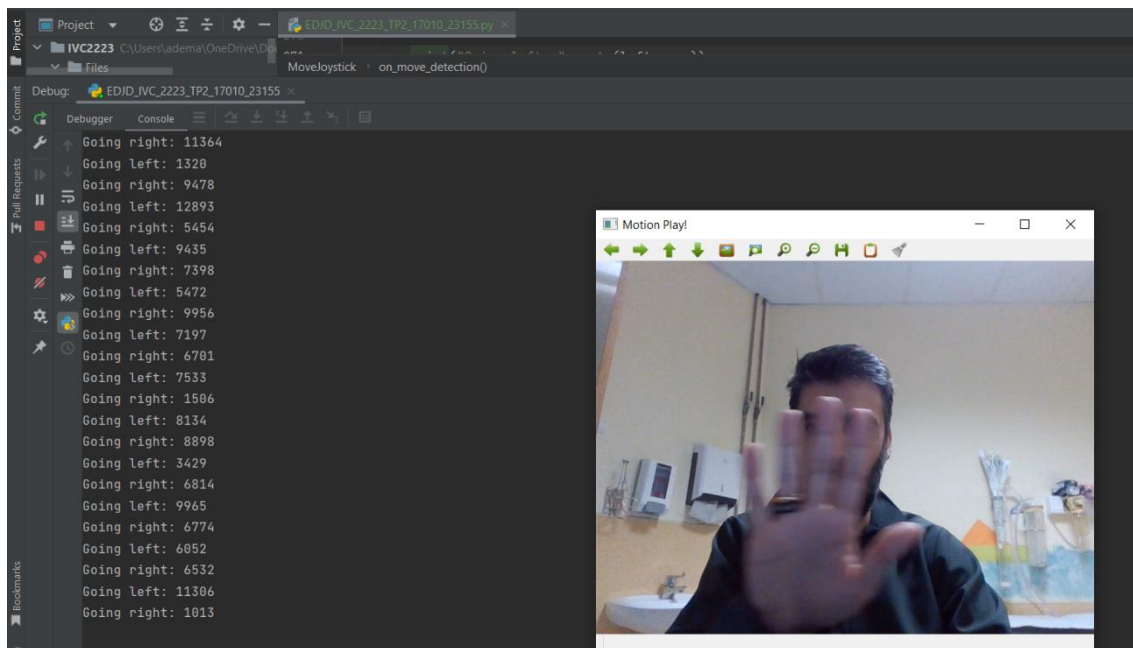


Figura 7 - Print de movimentação (np.count_nonzero).

Finalizando, tivemos algumas dificuldades no controlo do nosso *paddle*, e na construção da função que o determina. Inicialmente, não contemplamos a ausência de movimento como centro; o que fazia com que o *paddle* ficasse instável quando a câmara não detetava movimento. Esse problema foi debelado acrescentando “else: return CENTER”, dando a CENTER um valor na classe Enum.

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 2 – 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 movimento.	
Pode ser aplicado um dos algoritmos de deteção de movimento abordado em aula.	
O controlo do <i>paddle</i> deve basear-se no movimento dos pixels da imagem.	

Figura 8 - Objetivos propostos no enunciado.

Análise crítico-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 conjugação da implementação da Filtragem e do algoritmo de Optical Flow de Farneback que nos são dadas pelo OpenCV deu-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.

Sentimos, 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), regulação dos valores do Farneback e regulação do threshold de movimento, foram sendo afinados até se conseguir um resultado satisfatório, com fluidez suficiente na hora de controlar o *paddle* do nosso jogo. Verificamos que o `cv2.calcOpticalFlowFarneback` nos dá um controlo rigoroso e aceitável, desde que não existam movimentos abruptos. 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 comparativamente com o primeiro trabalho 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://www.geeksforgeeks.org/opencv-the-gunnar-farneback-optical-flow/>

<https://nanonets.com/blog/optical-flow/>

<http://www.diva-portal.org/smash/get/diva2:273847/FULLTEXT01.pdf>

<http://www.diva-portal.org/smash/get/diva2:302485/FULLTEXT01.pdf>

<https://www.pythonpool.com/cv2-normalize/>

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

        self.move_detection = MoveJoystick()

        # 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.move_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.move_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.move_detection.on_move_detection

    if screen_move == Screen.LEFT:
        self.paddle.move(-15)
    elif screen_move == Screen.RIGHT:
        self.paddle.move(15)

    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.move_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 MoveJoystick:

    def __init__(self):
        self.next_gray = None
        self.previous_gray = None
        self.cap = cv2.VideoCapture()
        self.window_name = "Motion Play!"

    def open_window(self):

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

        ret, frame1 = self.cap.read()
        frame1 = cv2.flip(frame1, 1)
        blur1 = cv2.GaussianBlur(frame1, (25, 25), 0)

        #cv2.imshow("Gaussian blur", blur1)

        self.previous_gray = cv2.cvtColor(blur1, cv.COLOR_BGR2GRAY)

    @property
    def on_move_detection(self):
        cap = self.cap

        ret, frame2 = cap.read()
        frame2 = cv2.flip(frame2, 1)
        blur2 = cv2.cvtColor(frame2, cv.COLOR_BGR2GRAY)

        self.next_gray = cv2.GaussianBlur(blur2, (25, 25), 0)

        farneback = cv2.calcOpticalFlowFarneback(prev=self.previous_gray,
next=self.next_gray, flow=None,
                                                pyr_scale=0.5, # 0.5,
                                                levels=1, # 3,
                                                winsize=10, # 15,
                                                iterations=1, # 3,
                                                poly_n=5,
                                                poly_sigma=1.1,
                                                flags=0)

        threshold = 2.0

        # flow_norm = np.sqrt(farneback[:, :, 0] ** 2 + farneback[:, :, 1] **
2)
        # flow_norm_norm = cv2.normalize(flow_norm, None, 0.0, 1.0,
cv2.NORM_MINMAX)
        # cv2.imshow("Flow", flow_norm_norm)

        flow_xx = farneback[:, :, 0]

        left_move = np.count_nonzero(flow_xx < -threshold)
        right_move = np.count_nonzero(flow_xx > threshold)

        cv2.imshow(self.window_name, frame2)

        # print("Going left: " + str(left_move))
        # print("Going right: " + str(right_move))
```

```
    if left_move < 300:
        left_move = 0

    if right_move < 300:
        right_move = 0

    self.previous_gray = self.next_gray

    if left_move > right_move:
        return Screen.LEFT
    elif left_move < right_move:
        return Screen.RIGHT
    else:
        return Screen.CENTER

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()
```