Relatório do trabalho da disciplina de Introdução à Visão por Computador

Fase 2 – Deteção de Movimentos

Henrique Azevedo - 23488

José Lourenço - 23496

Engenharia em Desenvolvimento de Jogos Digitais

Novembro de 2022

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Índice

Introdução

Contextualização

Este projeto diz respeito à proposta de trabalho da unidade curricular de Introdução à Visão por Computador (IVC), no âmbito do 2º ano da Licenciatura em Engenharia em Desenvolvimento de Jogos Digitais, no Instituto Politécnico do Cávado e Ave, realizado durante o período letivo de 2022/2023.

A partir da utilização do código jogo *Breakout*, previamente dado pelo docente, objetiva-se, com o uso de algoritmos de deteção de movimentos e com o uso da câmara seja possível detetar um objeto e com o mesmo, realizar as ações necessárias para o jogo.

Nesta segunda fase do projeto foi proposto a colocação do *paddle* do jogo *Breakout* a mover, utilizando assim, alguns métodos de algoritmia de deteção de movimento que foram lecionadas ao longo do decorrer das aulas.

Concluindo, este relatório vem a fazer explicação profunda de todos os métodos adotados que contribuíram para criação da proposta elaborada para a fase 2, com conceitos teóricos, problemas e resultados.

Desenvolvimento

Para a boa realização desta fase do projeto, foi necessária a revisão de vários conceitos teóricos sobre processamento de imagem que foram previamente lecionados e assim criar um melhor entendimento e encontrar a melhor forma de resolver o projeto proposto.

Conceitos Teóricos

Filtragem

Aparte da deteção do objeto que se pretende utilizar, através da câmara, é habitual que também se detete ruído na imagem, que pode fazer com que o resultado que queremos alcançar não seja o mais limpo.

A filtragem é utilizada para remover o ruído de uma imagem ou para salientar algum atributo em específico. Podendo se utilizar filtros passa-baixo e passa-alto, conforme o objetivo final.

Para a remoção de tal ruído, utilizou-se um filtro, passa-baixo, que procede á remoção de componentes de alta frequência da imagem, assim obtendo uma imagem mais suave e um pouco desfocada.

Quanto maior for a máscara aplicada melhor será o efeito de suavização.

No projeto foi utilizado um filtro gaussiano, aplicando assim a função *"cv.GaussianBlur"*, que multiplica cada pixel da imagem pelo valor da máscara

Para se obter a máscara do filtro gaussiano usa-se a seguinte formula:

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

Figura 1 - Fórmula *Kernel Gaussiano*

O *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.

Espaço de Cor HSV

O espaço de cor HSV proporciona um método intuitivo de especificar a cor. Neste espaço de cor, cada cor é representada por três componentes:

- Tonalidade ou Matiz (*Hue*);
- Saturação (*Saturation*);
- Valor (*Value*).

No espaço HSV é possível selecionar a tonalidade desejada, e posteriormente realizar ajustes na saturação e intensidade.

Esta separação entre a componente da luminância (luz) e da crominância (cor) traz vantagens relativamente ao espaço RGB quando se pretende realizar operações sobre cores.

Espaço de Cor RGB para Gray

O espaço de cor RGB faz uso de três cores primárias (vermelho, verde e azul) para produzir o conjunto de cores visíveis. É um sistema de cores aditivo, baseado na teoria tricromática.

O espaço de cor RGB é assim constituído por 3 componentes:

- Vermelho (RED);
- Verde (GREEN);
- Azul (BLUE);

Numa profundidade de cor de 24bpp (bits por pixel), cada um dos três componentes R, G e B é definida por uma variável de 8 bits, que pode tomar valores entre 0 e 255.

O valor 0 (zero) indica que a referida componente de cor não tem qualquer contribuição para a representação da cor final. O valor 255 indica que a contribuição dessa componente será máxima.

Para converter imagens RGB para Gray é primeiro necessário passar a imagem de RGB para BGR (azul, verde e vermelho) e de seguida converter para Gray.

Optical Flow

Optical Flow é o movimento de objetos entre *frames* consecutivos numa sequência, a partir de movimentos relativos entre a camara e o objeto. Existem 2 tipos de *Optical Flow*: *Sparse Optical Flow* e *Dense Optical Flow* (com o método *Farneback*).

Farneback Method:

Em *Dense Optical Flow*, olha-se para todos os pontos e deteta-se a mudança de intensidade dos pixéis entre *frames*, resultando numa imagem com pixéis destacados e em HSV para melhor visibilidade.

Assim, calcula a magnitude e direção o *Optical Flow* por um *array* de vetores. De seguida, visualiza o angulo (direção) do *flow* pela *hue* e distancia (magnitude) do *flow* por valores de HSV. Por fim, para uma melhor visibilidade, a saturação é colocada no valor máximo (255).

O *OpenCV* disponibiliza a função “cv2.calcOpticalFlowFarneback”, para o uso em *Dense Optical Flow*.

Descrição

Ao início foi criada uma classe, de nome *"camInput"*, contida no construtor do jogo, que permitirá captar vídeo, através da camara, assim captando o movimento de algo.

Na mesma, foram implementados métodos de segmentação de imagem para, assim, ao usar a variável *"cap"* que será assignada à classe *"cv.VideoCapture()"*, para obter a imagem da câmara, seguida de um método chamado *"showcam"* que tem como objetivo mostrar a imagem captada numa janela, com o uso da função *"cam.open(0)"*.

No mesmo método também se deteta a *frame* e inverte a imagem, para esta estar a par com a visão do jogador.

Após a obtenção do *frame*, o mesmo será transformado numa uma imagem *blur*, através da função de *Kernel Gaussiano* ("*cv.GaussianBlur*"), para tornar a imagem mais suave reduzindo o ruido, tendo sido aplicado uma convulsão de 25x25. Depois da transformação em imagem *blur*, segue se a transformação da imagem de RGB para *Gray Scale* que e depois é criada uma *mask* com o máximo valor de saturação.

Para seguintes processos de algoritmia foi criado um método de nome "*Farneback_Method*" onde será possível, como descrito na seguinte parte, a deteção de movimento necessária para a elaboração da fase.

Assim, é criada uma segunda *frame* da imagem detetada onde também se transforma em imagem *blur* para melhor suavidade.

Após a obtenção destas duas *frames*, é calculado o *Optical Flow* através do método de *Farneback* com o uso da função "*cv2.calcOpticalFlowFarneback*", resultando numa imagem com pixéis destacados, que mais tarde serão convertidos em HSV para uma melhor visibilidade.

Assim, com uso do método de *Farnback* calcula-se a magnitude e direção do *Optical Flow* através de *arrays* de vetores. Seguido da obtenção do angulo (direção) do *flow* pela alteração do *hue* e a distância (magnitude) pela variação do valor de HSV. Por fim, para uma melhor visibilidade, aplica-se a máscara criada no método anterior para alterar a saturação dos pixéis em destaque para o valor máximo (255).

De seguida é definida a direção do movimento detetado e sua implementação do método "*game_loop*" com a chamada do método previamente descrito.

No final, foi elaborado o método *"destroywindow"* que tem como propósito fechar a janela da imagem quando o jogo não está a decorrer.

Problemas de Codificação

O problema com mais destaque encontrado nesta fase foi a elaboração de um método que detetasse a quantidade de pixéis e assim para entender o movimento, caso ele fosse para a esquerda ou para a direita.

Resultados

No final da implementação de todo o algoritmo no código, conseguiu-se obter o resultado pedido no enunciado da fase 2.

Ao iniciar o programa, surge uma janela que apresenta o jogo *Breakout*, como ser pode ver imagem seguinte

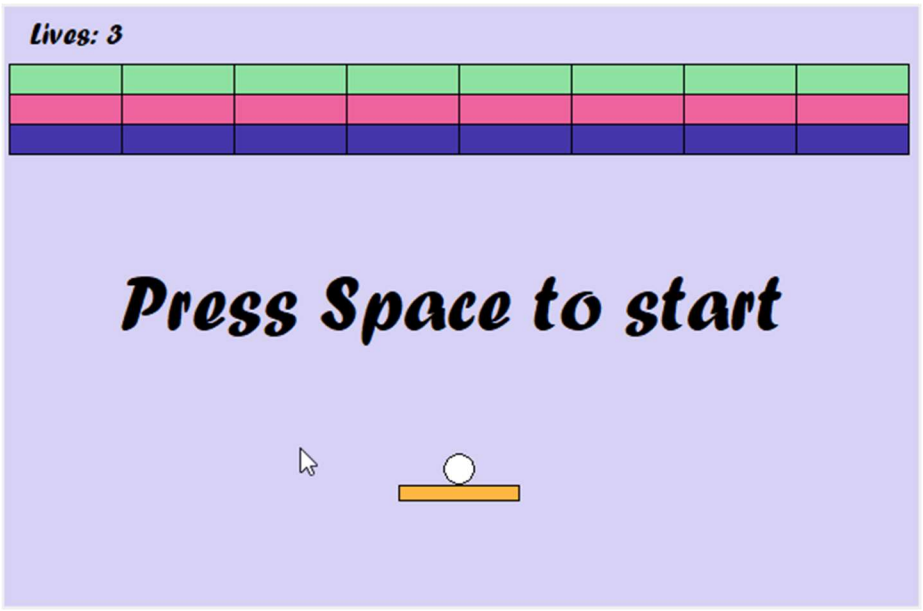

Figura 2 - Jogo *Breakout*

Assim para iniciar o jogo é necessário carregar a tecla espaço, o que fará com que abra uma janela com a camara se abra.

Nesta janela é possível fazer o reconhecer quaisquer movimentos detetados, que serão utilizados para mover o *paddle* no jogo. Assim, quando algum movimento é detetado a mover se para a direita da imagem, o *paddle* irá também para a direita e vice-versa, assim como quando não é detetado qualquer movimento, o *paddle* não se move.

Nas seguintes imagens é possível interpretar a deteção de movimento e deslocação do *paddle*:

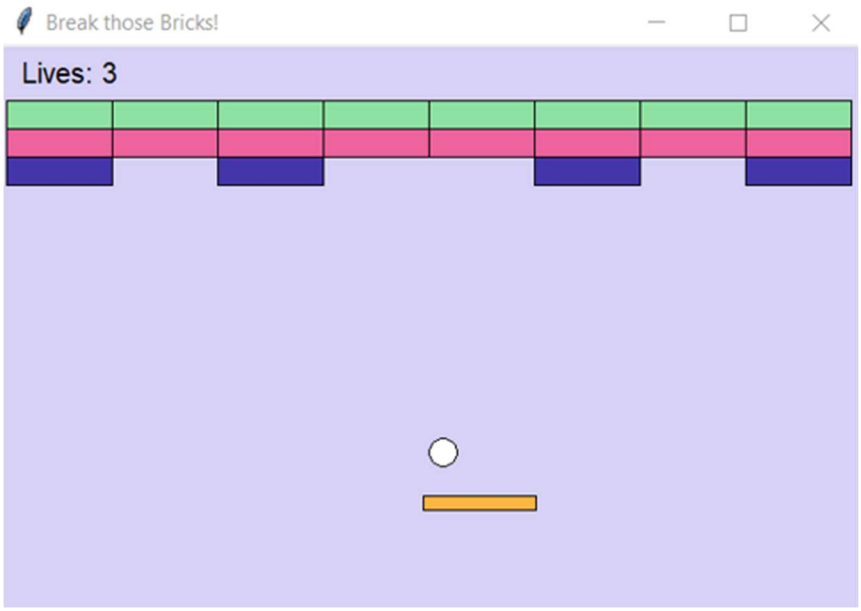

Figura 3 - Partida de Breakout

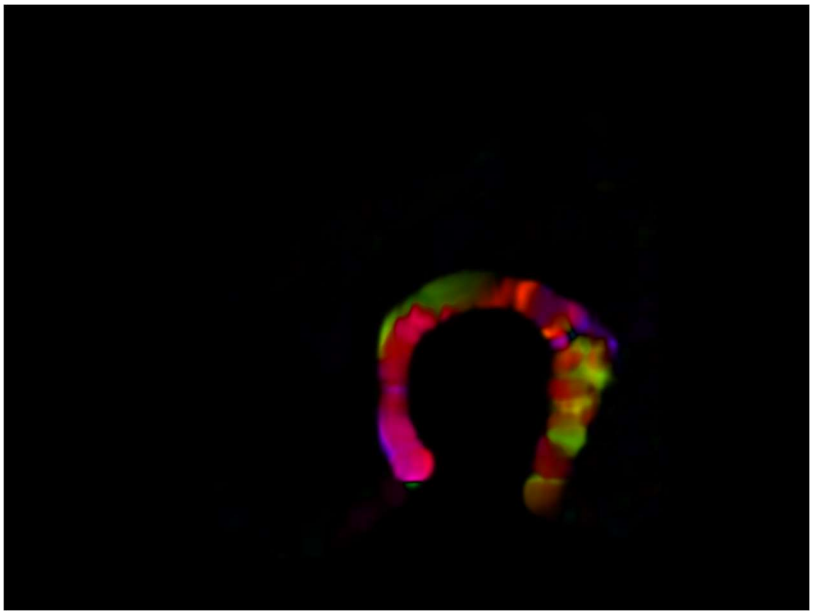

Figura 4 – Deteção e Destaque de Movimento

Conclusão

Com o final desta fase entende-se que foi possível a aprendizagem de vários conhecimentos sobre algoritmos de deteção de movimentos e sua respetiva implementação, tendo havido a grande contribuição dos exercícios realizados em aula pelo docente.

Sendo uma área nova foi necessário um maior empenho no estudo de informação para a melhor resolução do projeto, tal como discussões entre o grupo para decidir o melhor meio de trabalho, tendo escolhido o método que mais nos deu conforto na sua realização.

Ao longo desta fase surgiram diversas situações com dificuldades e até completos desentendimentos o que nos ajudou a ter uma evolução crítica e prática de como resolver futuros problemas.

A continuação do trabalho irá prosseguir para obtermos mais conhecimentos e prática nesta área, assim ajudando numa melhor realização da próxima fase.

Bibliografia

https://www.geeksforgeeks.org/opencv-the-gunnar-farneback-optical-flow/

https://stackoverflow.com/questions/61647936/cv2-farneback-optical-flow-values-are-too-low