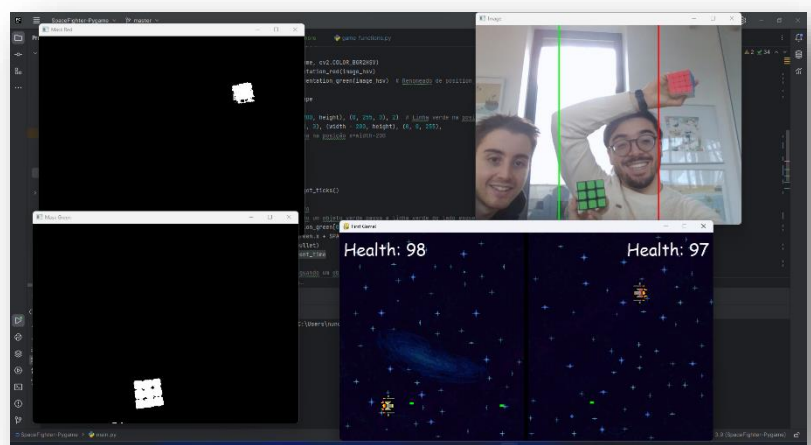


2023

# Space Fighters

Relatório para Introdução à Visão  
por Computador





## Trabalho Prático 2

### Relatório

#### Grupo :

20115 - André Cerqueira

20116 - Nuno Fernandes

**Orientador(a):** José Brito

**Engenharia em Desenvolvimento de Jogos Digitais**

Barcelos | novembro, 2023

## Índice

1.	Introdução ao trabalho 1 .....	4
1.1.	Contextualização .....	4
1.2.	Motivação e Objetivos .....	4
1.3.	Estrutura do Documento .....	4
2.	Metodologia e Desenvolvimento.....	5
2.1.	Escolha do Jogo: SpaceFighter .....	5
2.2.	Segmentação de Cores .....	5
2.2.1.	Definição de cores.....	5
2.2.2.	Segmentação de Objetos Vermelhos:.....	6
2.2.3.	Segmentação de Objetos Verdes.....	6
2.3.	Deteção e Rastreamento .....	6
2.4.	Disparo de Tiros.....	7
2.4.1.	Mecânica de Disparo:.....	7
2.4.2.	Deteção de Posição para Disparo: .....	7
2.4.3.	Condições para Disparo:.....	8
3.	Conclusões.....	8
3.1.	Avaliação do Sistema de Controlo .....	8
3.2.	Integração da Visão por Computador no Jogo .....	8
3.3.	Desafios e Limitações.....	8
3.4.	Impacto da Interface de Utilizador Inovadora .....	9
4.	Introdução ao trabalho 2.....	9
4.1.	Evolução do Projeto.....	9
4.2.	Novas Metodologias .....	9
4.3.	Objetivos Revistos .....	10
5.	Metodologia e Desenvolvimento.....	10
5.1.	Alterações no Jogo: SpaceFighter.....	10
5.2.	Nova Abordagem de Deteção e Rastreamento.....	11
5.3.	Nova Mecânica de Disparo .....	11
6.	Conclusões.....	12
6.1.	Comparação com Trabalho 1 .....	12
6.2.	Novos Desafios e Limitações.....	12
6.3.	Impacto das Melhorias na Interface de Utilizador .....	12
7.	Conclusões Finais e Perspetivas Futuras .....	12
8.	Anexos.....	13



## 1. Introdução ao trabalho 1

### 1.1. Contextualização

No âmbito da disciplina "Introdução à Visão por Computador" do curso de Licenciatura em Engenharia em Desenvolvimento de Jogos Digitais no Instituto Politécnico do Cávado e do Ave, foi proposto o desenvolvimento de um jogo que utiliza técnicas de Visão por Computador. Esta iniciativa visa proporcionar uma experiência prática e aprofundada na aplicação de algoritmos de visão computacional em contextos de jogos digitais.

### 1.2. Motivação e Objetivos

A motivação principal deste trabalho reside na exploração e aplicação de técnicas de segmentação em visão por computador para criar uma interface de utilizador inovadora e interativa. O objetivo da primeira fase é implementar um sistema de controlo para um jogo, onde o jogador pode interagir através da câmara, que utiliza algoritmos de segmentação para detetar e rastrear objetos de cor predefinida.

### 1.3. Estrutura do Documento

Este relatório está organizado em seções que detalham a abordagem adotada, a metodologia de implementação, os resultados obtidos e uma análise crítica dos mesmos. A seção seguinte aborda a metodologia, seguida pelos resultados e, finalmente, as conclusões.

## 2. Metodologia e Desenvolvimento

Para esta fase, optou-se por desenvolver um jogo inspirado no clássico "SpaceFighter". A principal interação dos jogadores é controlar as naves espaciais, movendo-as verticalmente. A segmentação baseia-se na deteção de objetos de cores específicas, neste caso, vermelho e verde.

### 2.1. Escolha do Jogo: SpaceFighter

Para a implementação deste projeto, optou-se pelo jogo "SpaceFighter". Este jogo foi escolhido devido à sua mecânica simples, mas envolvente, onde duas naves espaciais duelam e disparam balas uma contra a outra. A simplicidade do jogo permite que a ênfase seja colocada nas técnicas de Visão por Computador, conforme estipulado pelo enunciado.

### 2.2. Segmentação de Cores

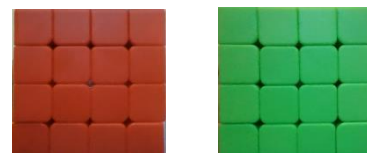
A segmentação de cores é uma técnica fundamental em visão computacional, especialmente quando se deseja isolar objetos específicos numa imagem. No projeto "SpaceFighter", essa técnica foi empregue para detetar e rastrear objetos coloridos que servem como controlos para as naves no jogo.

Que utiliza a biblioteca OpenCV, o espaço de cores HSV (Hue, Saturation, Value) foi empregue devido à sua capacidade de separar a componente de luminosidade (Value) das componentes de cor (Hue e Saturation), torna a segmentação de cores mais robusta sob diferentes condições de iluminação.

#### 2.2.1. Definição de cores

No script `color_segmentation.py`, são definidos intervalos específicos para as cores vermelha e verde no espaço de cores HSV (Hue, Saturation, Value). O espaço de cores HSV é frequentemente usado em visão computacional devido à sua capacidade de separar a informação de luminância da informação de cor, tornando-o menos sensível às variações de iluminação.

As cores foram retiradas de cubos mágicos que é o objeto utilizado para fazer a movimentação da nave.



### 2.2.2. Segmentação de Objetos Vermelhos:

A segmentação de objetos vermelhos é um pouco mais complexa devido à natureza do espaço de cores HSV, onde o vermelho pode aparecer em ambos os extremos do espectro e matiz. Assim, foram definidos dois intervalos para capturar objetos vermelhos:

- [Função update\\_segmentation\\_red](#)

Aqui, dois intervalos de matiz são definidos para capturar o vermelho. As duas máscaras resultantes são então combinadas ao usar uma operação OR bitwise para criar uma máscara final que segmenta objetos vermelhos.

### 2.2.3. Segmentação de Objetos Verdes

Para segmentar objetos verdes, foi definido um intervalo específico no espaço HSV:

- [Função update\\_segmentation\\_red;](#)

Neste trecho, a imagem capturada pela câmara é primeiro convertida para o espaço de cores HSV. Em seguida, é definido um intervalo de cores para o verde e uma máscara binária é criada pelo `cv2.inRange()`, onde os pixels dentro do intervalo são definidos como brancos (valor 255) e os pixels fora do intervalo são definidos como pretos (valor 0).

Estas segmentações permitem que o sistema detete e rastreie objetos coloridos em tempo real, proporciona uma interface de utilizador interativa e envolvente para o jogo.

## 2.3. Detecção e Rastreamento

Após a segmentação, uma etapa crucial é a deteção e o rastreamento dos objetos segmentados. Esta etapa permite que o sistema identifique a posição e o movimento dos objetos, traduzindo-os em comandos para o jogo.

### Identificação de Contornos:

Com a biblioteca OpenCV, os contornos dos objetos segmentados foram identificados. A função `findContours` foi empregue para este propósito:

- [findContours;](#)

Neste exemplo, mask é a máscara binária resultante da segmentação de objetos vermelhos. A função retorna uma lista de contornos detetados na máscara.

#### **Seleção do Contorno Principal:**

Dado que podem existir múltiplos contornos detetados, é essencial identificar qual deles representa o objeto de controlo principal. Para isso, o contorno de maior área foi selecionado:

- [largest\\_contour](#)

A função `contourArea` calcula a área de um contorno, e a função `max` é usada para selecionar o contorno com a maior área.

#### **Determinação da Posição:**

Com o contorno principal identificado, a posição central do objeto foi calculada para determinar a posição da nave no jogo:

- [Calcular a posição](#)

## 2.4. Disparo de Tiros

O disparo de tiros é uma mecânica central no jogo "SpaceFighter", que permite que os jogadores ataquem o adversário. Esta funcionalidade é implementada no script `main.py`, onde a lógica para disparar tiros é cuidadosamente integrada com a deteção de movimento através da visão computacional.

### 2.4.1. Mecânica de Disparo:

A mecânica de disparo é controlada por duas variáveis principais: `last_green_shot` e `last_red_shot`, que registam o tempo desde o último tiro disparado por cada nave. Isso é usado para implementar um atraso entre os tiros, evita que as naves disparem continuamente sem qualquer restrição.

### 2.4.2. Deteção de Posição para Disparo:

A posição dos objetos coloridos é continuamente atualizada através das funções `update_segmentation_red` e `update_segmentation_green`. Quando um objeto é detetado cruza



uma linha virtual definida na imagem da câmara, um tiro é disparado.

#### 2.4.3. Condições para Disparo:

O jogo verifica se a posição detetada do objeto vermelho está à esquerda de uma linha vermelha virtual (red\_line\_x) e se a posição detetada do objeto verde está à direita de uma linha verde virtual (green\_line\_x). Se essas condições forem atendidas e o tempo desde o último tiro for maior que o atraso definido (BULLET\_DELAY), um novo tiro é disparado.

- [Excerto de código com a condição de disparo.](#)

## 3. Conclusões

### 3.1. Avaliação do Sistema de Controlo

O sistema de controlo implementado demonstrou ser eficaz na deteção e rastreamento de objetos coloridos, que permite uma interação intuitiva e responsiva no contexto do jogo "SpaceFighter". A utilização do espaço de cores HSV provou ser uma escolha acertada, dada a sua robustez em diferentes condições de iluminação, o que é crucial para a consistência da jogabilidade em ambientes variados.

### 3.2. Integração da Visão por Computador no Jogo

A integração das técnicas de visão por computador no jogo foi bem-sucedida, proporciona uma nova forma de interação que vai além dos métodos tradicionais de input, como teclado e rato. Esta abordagem não só enriqueceu a experiência do utilizador, mas também demonstrou o potencial da visão por computador aplicada ao desenvolvimento de jogos digitais.

### 3.3. Desafios e Limitações

Durante o desenvolvimento, enfrentaram-se desafios relacionados com a precisão da segmentação de cores em diferentes ambientes de iluminação. Além disso, a deteção de múltiplos objetos e a seleção do objeto de controlo principal foram aspetos que exigiram atenção especial para garantir a precisão do sistema.

### 3.4. Impacto da Interface de Utilizador Inovadora

A interface de utilizador baseada em movimento proporcionou uma dimensão adicional de imersão e interatividade. Este aspeto do projeto não só cumpriu os objetivos pedagógicos da disciplina, mas também ofereceu insights sobre como interfaces de utilizador inovadoras podem ser desenvolvidas e integradas em jogos futuros.

## 4. Introdução ao trabalho 2

### 4.1. Evolução do Projeto

O projeto "SpaceFighter" evoluiu significativamente do Trabalho 1 para o Trabalho 2, marca numa transição importante na abordagem e na tecnologia utilizada. Inicialmente, o foco estava na segmentação de cores e deteção simples de objetos para controlar as naves no jogo. No entanto, com o Trabalho 2, avançamos para a integração de algoritmos mais sofisticados e robustos de Visão por Computador.

As principais mudanças incluem a substituição da segmentação de cores por um modelo de deteção de objetos baseado em YOLO (You Only Look Once), que oferece maior precisão e eficiência. Esta mudança permitiu uma deteção de objetos mais complexa e uma interação mais rica e dinâmica com o jogo. Além disso, a precisão aprimorada na deteção de movimentos e objetos resultou numa jogabilidade mais fluída e numa experiência de utilizador mais envolvente.

### 4.2. Novas Metodologias

No Trabalho 2, adotamos o modelo YOLO para deteção de objetos, uma mudança significativa em relação às técnicas de segmentação de cores usadas anteriormente. O YOLO é um dos algoritmos de deteção de objetos em tempo real mais avançados e eficientes disponíveis, conhecido pela sua rapidez e precisão. Ele permite identificar e classificar vários objetos em uma imagem simultaneamente, o que é ideal para o ambiente dinâmico de um jogo.

Implementamos o YOLO para identificar objetos específicos que os jogadores usam para controlar as naves no jogo, o que permitiu uma deteção mais precisa e uma melhor resposta aos movimentos dos jogadores. Esta abordagem também superou os desafios de

iluminação e variações de cor que enfrentávamos anteriormente, torna o jogo mais jogável em diferentes ambientes.

#### 4.3. Objetivos Revistos

Com as mudanças implementadas no Trabalho 2, os objetivos do projeto foram revisados e expandidos. O objetivo principal continua a ser a criação de uma interface de utilizador inovadora para o jogo "SpaceFighter", mas com uma ênfase renovada na precisão e na interatividade. Além disso, visamos demonstrar a aplicabilidade de algoritmos avançados de Visão por Computador no desenvolvimento de jogos digitais, particularmente em contextos onde a interação do utilizador com o jogo é essencial.

O uso do modelo YOLO para deteção de objetos elevou o padrão do nosso sistema de controlo, que permite uma jogabilidade mais intuitiva e responsiva. Esta abordagem reflete o nosso compromisso em explorar e integrar tecnologias emergentes no campo do desenvolvimento de jogos.

### 5. Metodologia e Desenvolvimento

#### 5.1. Alterações no Jogo: SpaceFighter

No trabalho 1 foi usado um algoritmo de segmentação de cores e a deteção de movimentos através de máscaras de cor HSV.

No Trabalho 2, a introdução do modelo YOLO (You Only Look Once) marca uma evolução significativa na forma como o jogo "SpaceFighter" processa e responde aos inputs do utilizador. O YOLO é um sistema de deteção de objetos em tempo real, conhecido pela sua eficiência e precisão.

- Utilização do modelo YOLO;
- Retorno da função `update_detection_yolo;`

Neste trecho de código, o modelo YOLO é inicializado com um modelo pré-treinado (yolov8n.pt). Quando uma imagem (frame) é capturada da câmara, ela é processada pelo modelo YOLO. O método `update_detection_yolo` transforma o frame para o espaço de cores RGB (necessário para o YOLO) e aplica o modelo para detetar objetos.

O resultado deste processo é uma lista de objetos detetados, cada um contém informações sobre sua posição, dimensão da caixa delimitadora e a classe a que pertence (por

exemplo, nave verde ou vermelha). Este método é uma mudança fundamental em relação ao Trabalho 1, onde a deteção era baseada unicamente em cores específicas. Com o YOLO, a deteção torna-se mais flexível e robusta, que permite o reconhecimento de uma variedade de objetos em diferentes condições de iluminação e fundo.

## 5.2. Nova Abordagem de Deteção e Rastreamento

No Trabalho 2 do projeto "SpaceFighter", a abordagem de segmentação de cores foi substituída pelo avançado modelo YOLO para deteção de objetos.

- Selecionar o “melhor” objeto;

Neste trecho de código, o processo de deteção com o YOLO é realizado. Para cada objeto detetado, o modelo fornece uma "caixa delimitadora" que indica a posição e o tamanho do objeto na imagem, além de um identificador de classe (`class_id`). Este identificador é crucial para determinar se o objeto detetado é relevante para o jogo (por exemplo, uma nave vermelha ou verde).

Diferente da segmentação de cores do Trabalho 1, que dependia de faixas de cores HSV específicas e era sensível a variações de iluminação e cor, o modelo YOLO oferece uma deteção baseada em características mais gerais dos objetos. Isso aumenta a robustez do sistema em diferentes condições ambientais e oferece uma experiência de jogo mais consistente e confiável, independentemente das cores ou formas dos objetos utilizados pelos jogadores para interagir com o jogo.

## 5.3. Nova Mecânica de Disparo

No Trabalho 2, a mecânica de disparo do jogo "SpaceFighter" foi aprimorada através da integração do modelo YOLO para a deteção de objetos. Diferentemente do Trabalho 1, onde o disparo dependia da deteção de posição através da segmentação de cores, o Trabalho 2 utiliza uma abordagem mais avançada. Por exemplo:

- Código relacionado com o disparo da nave;

Este trecho de código demonstra como a deteção de objetos com o YOLO influencia a mecânica de disparo. A identificação da classe do objeto (nave vermelha ou verde) e sua posição são usadas para determinar quando um tiro deve ser disparado, tornando o processo mais intuitivo e alinhado com os movimentos naturais dos jogadores.

## 6. Conclusões

### 6.1. Comparação com Trabalho 1

Compara com o Trabalho 1, o Trabalho 2 apresenta melhorias notáveis em termos de precisão, interatividade e robustez. A transição da segmentação de cores para a deteção de objetos com YOLO representou um salto qualitativo no controle e na resposta do jogo. Essas mudanças refletem uma evolução tecnológica importante, alinhada com as tendências atuais em visão por computador aplicada a jogos digitais.

### 6.2. Novos Desafios e Limitações

Embora o Trabalho 2 tenha trazido inovações significativas, também surgiram novos desafios e limitações. A complexidade aumentada do sistema com a integração do YOLO exigiu um processamento mais intenso, o que pode impactar o desempenho em máquinas com recursos limitados. Além disso, a precisão na deteção de objetos em ambientes com iluminação variável ainda representa um desafio a ser superado.

### 6.3. Impacto das Melhorias na Interface de Utilizador

As melhorias implementadas no Trabalho 2 tiveram um impacto positivo na interface de utilizador. A interação tornou-se mais fluida e natural, eleva a experiência do utilizador. O uso do YOLO para deteção de objetos ofereceu uma maneira mais flexível e precisa para os jogadores interagirem com o jogo, marca um avanço significativo em relação aos métodos tradicionais de controle.

## 7. Conclusões Finais e Perspetivas Futuras

Em suma, o projeto alcançou seus objetivos de implementar um sistema de controlo interativo que utiliza visão por computador e de explorar a aplicabilidade desta tecnologia no desenvolvimento de jogos. Para trabalhos futuros, considera-se a expansão do sistema para incluir mais cores e objetos, bem como a otimização da segmentação para ambientes com iluminação ainda mais variável. Adicionalmente, a implementação de algoritmos de *machine learning* para melhorar o rastreamento e a deteção de objetos poderia ser explorada, aumenta a precisão e a robustez do sistema de controlo.

Este projeto não só confirmou a viabilidade da visão por computador em jogos digitais, mas também abriu caminho para investigações futuras que podem levar a avanços significativos na forma como interagimos com os jogos.

## 8. Anexos

GitHub do projeto - <https://github.com/AndreCerqueira/SpaceFighter-Pygame.git>

```
def update_segmentation_red(image_hsv):
    mask_h1 = cv2.inRange(image_hsv[:, :, 0], hmin_red1, hmax_red1)
    mask_h2 = cv2.inRange(image_hsv[:, :, 0], hmin_red2, hmax_red2)
    mask_h = cv2.bitwise_or(mask_h1, mask_h2)
    mask_s = cv2.inRange(image_hsv[:, :, 1], smin_red, smax_red)
    mask_v = cv2.inRange(image_hsv[:, :, 2], vmin_red, vmax_red)

    mask = cv2.bitwise_and(mask_h, cv2.bitwise_and(mask_s, mask_v))
    cv2.imshow("Mask Red", mask)

    contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(largest_contour)
        return x + w // 2, y + h // 2, w, h
    else:
        return None
```

Figura 1 - Excerto de Código

```
def update_segmentation_green(image_hsv):
    mask_h = cv2.inRange(image_hsv[:, :, 0], hmin_green, hmax_green)
    mask_s = cv2.inRange(image_hsv[:, :, 1], smin_green, smax_green)
    mask_v = cv2.inRange(image_hsv[:, :, 2], vmin_green, vmax_green)

    mask = cv2.bitwise_and(mask_h, cv2.bitwise_and(mask_s, mask_v))
    cv2.imshow("Mask Green", mask)

    contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(largest_contour)
        return x + w // 2, y + h // 2, w, h
    else:
        return None
```

Figura 2 - Excerto de Código

```
contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
```

Figura 3 - Excerto de Código

```
largest_contour = max(contours, key=cv2.contourArea)
```

Figura 4 - Excerto de Código

```
x, y, w, h = cv2.boundingRect(largest_contour)
return x + w // 2, y + h // 2, w, h
```

Figura 5 - Excerto de Código

```
# Shooting logic for green and red spaceships
if position_green and position_green[0] > green_line_x and current_time - last_green_shot > BULLET_DELAY:
    bullet = pygame.Rect(green.x + SPACESHIP_WIDTH, green.y + SPACESHIP_HEIGHT // 2 - 2, 10, 5)
    green_bullets.append(bullet)
    last_green_shot = current_time

if position_red and position_red[0] < red_line_x and current_time - last_red_shot > BULLET_DELAY:
    bullet = pygame.Rect(red.x - 10, red.y + SPACESHIP_HEIGHT // 2 - 2, 10, 5)
    red_bullets.append(bullet)
    last_red_shot = current_time
```

Figura 6 - Excerto de Código

```
model = YOLO("yolov8n.pt")
```

Figura 7 - Excerto de Código

```
yolo_objects = update_detection_yolo(frame, model)
```

Figura 8 - Excerto de Código

```
# Dicionário para armazenar o objeto com maior confiança para cada classe de interesse
best_objects = {}
for object in results[0]:
    box = object.bboxes.data[0]
    confidence = box[4]
    class_id = int(box[5])

    if class_id in [67, 47]:
        if class_id not in best_objects or confidence > best_objects[class_id]['confidence']:
            best_objects[class_id] = {'object': object, 'confidence': confidence}

# Extrair apenas os objetos do dicionário
detected_objects = [info['object'] for info in best_objects.values()]
```

Figura 9 - Excerto de Código

```
if class_id == 67: # ID da classe para a nave vermelha
    cv2.rectangle(img=frame, pt1=pt1, pt2=pt2, color=(0, 0, 255), thickness=2)
    green_handle_movement(position, red)

# Lógica de disparo para a nave vermelha
if x_center < red_line_x and current_time - last_red_shot > BULLET_DELAY:
    bullet = pygame.Rect(red.x - 10, red.y + SPACESHIP_HEIGHT // 2 - 2, 10, 5)
    red_bullets.append(bullet)
    last_red_shot = current_time
```

Figura 10 - Excerto de Código