



**Politécnico  
Castelo Branco**

Escola Superior  
de Tecnologia

# **PowerSensation Games**

Experiências digitais para todos

Daniel Alexandre da Silva Santos

Sandro Filipe Almeida Cardoso

## **Orientadores**

Professora Doutora Ângela Cristina Marques de Oliveira

Professor Doutor Filipe Miguel Bispo Fidalgo

Trabalho de Projeto apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco, realizada sob a orientação da Professora Doutora Ângela Cristina Marques de Oliveira e coorientação do Professor Doutor Filipe Miguel Bispo Fidalgo do Instituto Politécnico de Castelo Branco

**Junho 2026**



## Composição do júri

Presidente do júri

Professora Doutora Ângela Cristina Marques de Oliveira

Professor Coordenador, Escola Superior de Tecnologia

Vogais

Professor Doutor Eurico Ribeiro Lopes

Professor Adjunto, Escola Superior de Tecnologia

Professor Doutor Pedro Nuno Moreira da Silva

Professor Adjunto, Escola Superior de Tecnologia



## **Agradecimentos**

Para o desenvolvimento deste projeto final de curso, feito com dedicação e esforço, que não teria sido possível sem apoio e ajuda crucial de determinadas pessoas. Por isso, gostaria de expressar o meu agradecimento a essas pessoas.

Aos nossos orientadores de projeto, a Professora Doutora Ângela Costa e Professor Doutor Filipe Fidalgo e, pelo apoio dado durante o desenvolvimento, opiniões e críticas construtivas que foram uma enorme ajuda.

Aos nossos familiares e amigos, pelo magnífico apoio e incentivo.



## **Resumo**

O presente relatório foi realizado no âmbito da Unidade Curricular de Projeto II da Licenciatura em Informática e Multimédia da Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco, dando continuidade ao trabalho iniciado em Projeto I.

O PowerSensation Games é um videojogo desportivo inclusivo, desenvolvido em Unity, que integra múltiplas mecânicas inspiradas em modalidades desportivas e assenta num princípio nuclear de redundância sensorial, a comunicação do estado de jogo em simultâneo por canais visuais, sonoros e táteis/hápticos, garantindo o acesso à mesma informação por jogadores com deficiência visual e/ou auditiva. O projeto é complementado por uma consola física dedicada, sucessora das anteriores +Power e +PowerSensations, que reformula o sistema baseado em Arduino, acrescenta novos botões de grande formato e integra uma estrutura concebida e produzida por impressão 3D.

## **Palavras-chaves**

Acessibilidade em videojogos; Redundância sensorial; Feedback háptico; Arduino; Impressão 3D; Design inclusivo; Unity.

## **Abstract**

This report was developed within the Project II course of the Computer Science and Multimedia degree at the School of Technology of the Polytechnic Institute of Castelo Branco, as a continuation of the work started in Project I.

PowerSensation Games is an inclusive sports video game, developed in Unity, that integrates multiple sport-inspired mechanics and is grounded on a core principle of sensory redundancy, communicating game state simultaneously through visual, auditory and haptic channels, ensuring that players with visual and/or hearing impairments access the same information. The project is complemented by a dedicated physical console, successor to the previous +Power and +PowerSensations, which reworks the Arduino-based system, adds new large-format buttons and integrates a 3D-printed enclosure.

## **Keywords**

Video game accessibility; Sensory redundancy; Haptic feedback; Arduino; 3D printing; Inclusive design; Unity.



# Índice

|                                                                   |    |
|-------------------------------------------------------------------|----|
| 1. Introdução .....                                               | 1  |
| 1.1. Enquadramento .....                                          | 1  |
| 1.2. Objetivos .....                                              | 1  |
| 1.3. Calendarização.....                                          | 1  |
| 1.4. Estrutura do relatório .....                                 | 2  |
| 2. Projeto Multimédia .....                                       | 3  |
| 2.1. Design da Consola e Modelação 3D.....                        | 3  |
| 2.1.1. Design da Consola.....                                     | 3  |
| 2.1.2. Modelação 3D .....                                         | 5  |
| 2.2. Produção de Assets .....                                     | 6  |
| 2.3. Desenvolvimento .....                                        | 8  |
| 2.3.1. Menu de Seleção de Controlos e Perfis de Input.....        | 8  |
| 2.3.2. Menu Principal e Navegação no Menu .....                   | 10 |
| 2.3.3. Menu de Opções.....                                        | 11 |
| 2.3.4. Sistema de Progressão por Modalidade.....                  | 15 |
| 2.3.5. Arquitetura Escalável de Modalidades .....                 | 16 |
| 2.3.6. Gestão de Áudio .....                                      | 18 |
| 2.3.7. Seleção de Personagem .....                                | 19 |
| 2.3.8. Tutorial .....                                             | 20 |
| 2.3.9. Modalidade de Ciclismo .....                               | 20 |
| 2.3.10. QuickToggle: Descrição Áudio .....                        | 29 |
| 2.3.11. QuickToggle: Legendas .....                               | 31 |
| 2.3.12. QuickToggle: Áudio Espacial.....                          | 32 |
| 2.3.13. QuickToggle: Feedback Háptico .....                       | 33 |
| 2.4. Distribuição e Manutenção.....                               | 34 |
| 3. Desenvolvimento da Consola.....                                | 35 |
| 3.1. Especificação e Fundamentação dos Componentes.....           | 35 |
| 3.2. Análise Detalhada das Ligações Elétricas e Eletrónicas ..... | 37 |
| 3.3 Implementação e Lógica do Código Arduino .....                | 39 |
| 3.4. Montagem .....                                               | 43 |

|                                                   |    |
|---------------------------------------------------|----|
| 4. Conclusão .....                                | 45 |
| 4.1. Reflexão sobre o Trabalho Desenvolvido ..... | 45 |
| 4.2. Trabalho Futuro .....                        | 45 |
| 5. Bibliografia .....                             | 47 |

## Índice de Figuras

|                                                                                     |    |
|-------------------------------------------------------------------------------------|----|
| Figura 1 - Consola +Power .....                                                     | 3  |
| Figura 2-Modelo 3D projetado para a consola.....                                    | 6  |
| Figura 3 - Nova Consola feita.....                                                  | 6  |
| Figura 4-Frames usados para animação do ciclista .....                              | 7  |
| Figura 5-Composição de uma melodia no Chrome Music Lab .....                        | 8  |
| Figura 6 - Menu de seleção do Controlo .....                                        | 8  |
| Figura 7 - Estrutura de um perfil de input (InputProfile).....                      | 9  |
| Figura 8 - Menu Inicial.....                                                        | 10 |
| Figura 9 - Leitura de input e navegação por zonas (GameSelectionManager) .....      | 11 |
| Figura 10 - Menu de Opções.....                                                     | 12 |
| Figura 11 - Navegação em níveis do menu de Opções (OpcoesNavigator) .....           | 13 |
| Figura 12 – Base abstrata das opções (OpcaoOpcoesBase).....                         | 14 |
| Figura 13 - Opção de volume (OpcaoVolume).....                                      | 14 |
| Figura 14 - Opção de módulo ativo (OpcaoModuloAtivo).....                           | 14 |
| Figura 15 - Opção de tipo de daltonismo (OpcaoDaltonismoTipo) .....                 | 15 |
| Figura 16 - Acumulação persistente de XP (ProgressaoModalidade).....                | 15 |
| Figura 17 -Cálculo do nível a partir do XP (ProgressaoModalidade) .....             | 15 |
| Figura 18 - Apresentação da progressão no cartão (CardView).....                    | 16 |
| Figura 19 - Estrutura de dados de uma modalidade (ModalidadeData).....              | 17 |
| Figura 20 - Asset de modalidade no inspetor do Unity .....                          | 17 |
| Figura 21 - Geração dinâmica dos cartões de modalidade (GameSelectionManager) ..... | 18 |
| Figura 22 - Persistência da escolha entre cenas (SelecaoJogo).....                  | 18 |
| Figura 23 - Efeitos de navegação e confirmação (AudioManager) .....                 | 19 |
| Figura 24 - Gestão de volumes e música de fundo (AudioManager).....                 | 19 |
| Figura 25 - Menu Seleção de Personagens .....                                       | 20 |
| Figura 26 - Modalidade do Ciclismo .....                                            | 21 |
| Figura 27 - Velocidade do scroll (GameManager).....                                 | 22 |
| Figura 28 - Colocar a "meta" no ecrã (RoadScroller).....                            | 22 |
| Figura 29 - Teste se clicou na tecla para uma ação (CyclistController).....         | 22 |
| Figura 30 - Validar ação e enviar para o EventManager (CyclistController) .....     | 23 |
| Figura 31 - Ciclo de geração de eventos sem sobreposição (EventManager).....        | 23 |
| Figura 32 - Janela de reação do evento (EventManager).....                          | 24 |

|                                                                                                                                              |    |
|----------------------------------------------------------------------------------------------------------------------------------------------|----|
| Figura 33 - Validação da resposta adaptada ao dispositivo (EventManager) .....                                                               | 25 |
| Figura 34 - Indicação redundante do evento (IndicacaoEvento).....                                                                            | 25 |
| Figura 35 - Momento de ação no ecrã (Ciclismo).....                                                                                          | 26 |
| Figura 36 - Geração e deslocamento de obstáculos (ObstacleEvent) .....                                                                       | 26 |
| Figura 37 – Aumento de Velocidade com o Tempo (GameManager).....                                                                             | 27 |
| Figura 38 – Decisão probabilística da linha de meta (EventManager).....                                                                      | 28 |
| Figura 39 - Inclinação da câmara nas curvas (CameraEffects) .....                                                                            | 28 |
| Figura 40 - Ecrã de fim de jogo (ciclismo) .....                                                                                             | 29 |
| Figura 41 – Lógica do fim de jogo (UIManager) .....                                                                                          | 29 |
| Figura 42 - Código Integral do TTSManger .....                                                                                               | 30 |
| Figura 43 - Verificação do texto para a Descrição Audio (AccessibleButton) .....                                                             | 31 |
| Figura 44 - Ciclismo com as Legendas Ativas.....                                                                                             | 32 |
| Figura 45 - Cálculo da proximidade do Beep (ObstacleEvent) .....                                                                             | 33 |
| Figura 46 - Padrões semânticos (HapticoManager) .....                                                                                        | 34 |
| Figura 47 - Arduino Uno R4 Wifi .....                                                                                                        | 35 |
| Figura 48 - Botão Arcade de 60mm .....                                                                                                       | 36 |
| Figura 49 - Botão de Pressão de 30 mm.....                                                                                                   | 36 |
| Figura 50 - Modulo de Vibração para arduino .....                                                                                            | 37 |
| Figura 52 - Cantoneiras em L.....                                                                                                            | 37 |
| Figura 53 - Conectores Machos e Femeas .....                                                                                                 | 37 |
| Figura 54 - Esquema de Ligação dos botões ao Arduino .....                                                                                   | 38 |
| Figura 55 - Esquema de Ligação dos modulos de vibração .....                                                                                 | 39 |
| Figura 56 - Inicialização dos arrays sincronizados e do marcador de ausência de LED .....                                                    | 40 |
| Figura 57 - Implementação do algoritmo de debounce não-bloqueante, recorrendo à função millis() para filtragem de ruído eletromecânico. .... | 41 |
| Figura 58 - Funções de abstração responsáveis pela emulação USB HID (Keyboard.h) e pela atuação simultânea do feedback visual nos LEDs. .... | 42 |
| Figura 59 - Rotinas de telemetria: receção assíncrona orientada a buffers .....                                                              | 43 |
| Figura 60 - Função de decodificação com modulação segura de sinal PWM .....                                                                  | 43 |



## Índice de Tabelas

|                                                                      |   |
|----------------------------------------------------------------------|---|
| Tabela 1- Calendarização .....                                       | 2 |
| Tabela 2 - Comparação entre a +PowerSensation e a nova consola ..... | 4 |



## **Lista de abreviaturas, siglas e acrónimos**

DHH - Deaf and Hard of Hearing

HMI - Human-Machine Interface

IPCB - Instituto Politécnico de Castelo Branco

IoT - Internet of Things

NV-SDH - Non-Verbal Subtitles for the Deaf and Hard of Hearing

PRISMA - Preferred Reporting Items for Systematic Reviews and Meta-Analyses

PWM - Pulse Width Modulation

SUS - System Usability Scale

TTS - Text-to-Speech

UI - User Interface

XP - Experience Points

# 1. Introdução

Neste capítulo apresenta-se o enquadramento do projeto e a sua ligação a Projeto I, os objetivos gerais e específicos, a calendarização do segundo semestre e a estrutura do presente relatório.

## 1.1. Enquadramento

O presente relatório insere-se no âmbito da Unidade Curricular de Projeto II da Licenciatura de Informática e Multimédia da Escola Superior de Tecnologia de Castelo Branco, dando continuidade ao trabalho iniciado em Projeto I.

Em Projeto I foi realizada uma revisão sistemática da literatura segundo o protocolo PRISMA, com o objetivo de fundamentar o desenvolvimento de um videojogo desportivo inclusivo, assente no princípio da redundância sensorial a comunicação do estado de jogo em simultâneo por canais visuais, sonoros e táteis/hápticos. A partir dessa fundamentação, foram desenvolvidas as primeiras fases do modelo de projeto multimédia, nomeadamente a análise e planeamento, o design e o início da produção [1].

O presente documento retoma esse percurso e foca-se na fase de produção, descrevendo o desenvolvimento do videojogo em Unity e da consola física que o acompanha. A consola constitui uma evolução das anteriores +Power e +PowerSensations, reformulando o sistema baseado em Arduino, acrescentando botões.

## 1.2. Objetivos

O objetivo geral deste projeto é desenvolver um videojogo desportivo inclusivo e a respetiva consola adaptada, aplicando o princípio da redundância sensorial validado na revisão sistemática de Projeto I.

Os objetivos específicos passam por:

- Desenvolver, em Unity, as modalidades de ciclismo e de voleibol, assentes numa arquitetura escalável a novas modalidades;
- Implementar mecanismos de redundância sensorial (feedback visual, áudio direcional e háptico) em todas as modalidades;
- Reformular a consola física, sucessora da +Power e +PowerSensations, com novo sistema Arduino, botões e estrutura em impressão 3D;
- Documentar o desenvolvimento do projeto.

## 1.3. Calendarização

Tabela 1- Calendarização

| Atividade                        | Fev | Mar | Abr | Mai | Jun | Jul/Set |
|----------------------------------|-----|-----|-----|-----|-----|---------|
| Arquitetura de modalidades       | X   | X   |     |     |     |         |
| Produção: Ciclismo               |     | X   | X   | X   |     |         |
| Modulos de Acessibilidade        |     |     | X   | X   | X   |         |
| Nova Consola                     |     |     |     | X   | X   |         |
| Integração hardware-<br>software |     |     |     |     | X   | X       |
| Relatório                        |     |     |     | X   | X   | X       |

## 1.4. Estrutura do relatório

O presente relatório está organizado em três capítulos. No primeiro capítulo, Introdução, apresenta-se o enquadramento do projeto e a sua ligação a Projeto I, os objetivos gerais e específicos, a calendarização e a estrutura do relatório.

No segundo capítulo, Projeto Multimédia, descreve-se o desenvolvimento do projeto segundo o modelo de cinco fases adotado e fundamentado em Projeto I. Concluem-se neste documento as fases que ficaram em aberto: completa-se a produção do jogo e da consola, e desenvolvem-se as fases de testes e validação e de distribuição e manutenção.

No terceiro capítulo, Conclusão, apresenta-se uma reflexão sobre o trabalho desenvolvido e uma perspetiva do trabalho futuro.

## 2. Projeto Multimédia

A metodologia adotada corresponde ao desenvolvimento de um projeto multimédia, dividido em cinco fases: análise e planeamento, design, produção, testes e validação, e distribuição e manutenção. Este modelo foi adotado e fundamentado em Projeto I [1], neste capítulo desenvolvem-se as fases em falta para concluir o projeto.

### 2.1. Design da Consola e Modelação 3D

Nesta secção recuperam-se, de forma sintética, o conceito do jogo, as suas funcionalidades e os requisitos definidos em Projeto I, que enquadram as decisões de implementação descritas nas secções seguintes.

#### 2.1.1. Design da Consola

A consola dedicada do PowerSensation Games é uma evolução direta das versões anteriores +Power e +PowerSensations, desenvolvidas em projetos precedentes. A conceção desta nova versão partiu da análise das limitações das anteriores e da necessidade de as alinhar com os princípios de acessibilidade identificados na revisão de literatura

A consola +Power, representada na Figura 1, consiste num conjunto de três botões de formato grande que se conectam a um computador através de motherboards reaproveitadas de teclados antigos. Esta não possui quaisquer canais de retorno para o jogador[2].



*Figura 1 - Consola +Power*

A +PowerSensations é bastante parecida a +Power, porem, foi acrescentado dois motores de vibração, introduzindo um primeiro canal háptico, mas mantendo as duas limitações estruturais: apenas três botões e uma vibração dupla que devido a estrutura pequena não é possível distinguir direções, de acordo com os nossos próprios testes.

Estas duas limitações são relevantes do ponto de vista da acessibilidade. Por um lado, três botões restringem fortemente o vocabulário de interação disponível, obrigando a mapear ações distintas ao mesmo botão e dificultando jogos que exijam mais do que duas ou três

escolhas direcionais. Por outro lado, a falta de distinção na vibração não permite distinguir tipos de evento pelo tato, o que anula grande parte do potencial do canal háptico enquanto via de informação para jogadores com deficiência visual. A análise de funcionalidades de acessibilidade em jogos populares de Trick et al. evidencia precisamente que a ausência de canais de retorno ricos e de controlos flexíveis é uma das lacunas mais comuns e mais penalizadoras para estes jogadores [3].

*Tabela 2 - Comparação entre a +PowerSensation e a nova consola*

| <b>Característica</b>    | <b>+PowerSensations</b>                    | <b>Nova consola</b>                           |
|--------------------------|--------------------------------------------|-----------------------------------------------|
| Quantidade de Botões     | 2 Botões Grandes + 1 Pequeno               | 4 Botões Grandes + 1 Pequeno                  |
| Vocabulário de interação | Limitado; ações sobrepostas                | Alargado; suporta evasão e decisão direcional |
| Canal háptico            | 2 motores;                                 | 2 motores independentes;                      |
| Intensidade da vibração  | Fixa (ligado/desligado)                    | Modulável por PWM                             |
| Estrutura física         | Adaptada de um teclado e estrutura em MDF. | Estrutura dedicada, modelada e impressa em 3D |

A Tabela 2 sistematiza a comparação entre a +PowerSensations e a nova consola, evidenciando a evolução em cada uma das dimensões relevantes para a acessibilidade.

A diferença mais imediata reside no número de botões. A +PowerSensations dispunha de dois botões grandes e um pequeno, ao passo que a nova consola passa a quatro botões grandes e um pequeno. Este aumento não é meramente quantitativo: como discutido anteriormente, um número reduzido de botões obriga a sobrepor ações distintas ao mesmo controlo, limitando o vocabulário de interação disponível. Ao alargar o conjunto de botões, a nova consola amplia esse vocabulário de interação, passando a suportar tanto a evasão como a decisão direcional como ações distintas, sem as obrigar a partilhar o mesmo botão.

No canal háptico, ambas as versões integram dois motores de vibração, mas a forma como os utilizam difere substancialmente. Na +PowerSensations, a vibração era fixa, limitada a um estado de ligado ou desligado, o que impedia transmitir qualquer gradação de informação pelo tato. A nova consola controla os dois motores de forma independente e com intensidade modulável por PWM, o que permite não só distinguir o lado da vibração, atribuindo um motor a cada lado para indicação direcional, como variar a sua força consoante o evento, transformando o canal háptico de um simples alerta binário num verdadeiro veículo de informação gradual.

### 2.1.2. Modelação 3D

A modelação 3D da estrutura final da consola foi desenvolvida com recurso ao software *Tinkercad*. A escolha desta ferramenta de modelação assistida por computador (CAD) justifica-se pela sua eficiência na prototipagem rápida e facilidade na execução de operações booleanas (adição e subtração de volumes), que são ideais para criar os recortes exatos exigidos pelos componentes de hardware.

#### Ergonomia e Design Estrutural

O modelo digital foi desenhado com uma geometria angular (em formato de "V" aberto) e arestas arredondadas (fillets). Esta disposição espacial não é meramente estética; foi pensada para maximizar a ergonomia. O ângulo permite que o utilizador repouse as mãos de forma natural e confortável, garantindo um acesso equidistante e facilitado aos quatro botões Arcade principais e ao botão de sistema central.

#### Composição do Modelo

Como ilustrado na Figura 2, o projeto 3D é composto por duas peças complementares:

- **Corpo Principal (Chassis superior):** Atua simultaneamente como painel de controlo e caixa de proteção. Possui os orifícios perfeitamente dimensionados para o encaixe por pressão/rosca dos botões de 60 mm e 30 mm. Na face frontal, integra a gravação em relevo do nome "Power Sensation", reforçando a identidade visual do projeto.
- **Placa Base (Tampa inferior):** Uma peça plana concebida para fechar o chassis. O design em duas partes separadas é crucial do ponto de vista da engenharia, pois garante que o interior da consola (onde ficarão alojados o Arduino Uno R4, os motores de vibração e as baterias) permanece facilmente acessível para futuras manutenções ou substituições de cablagem.

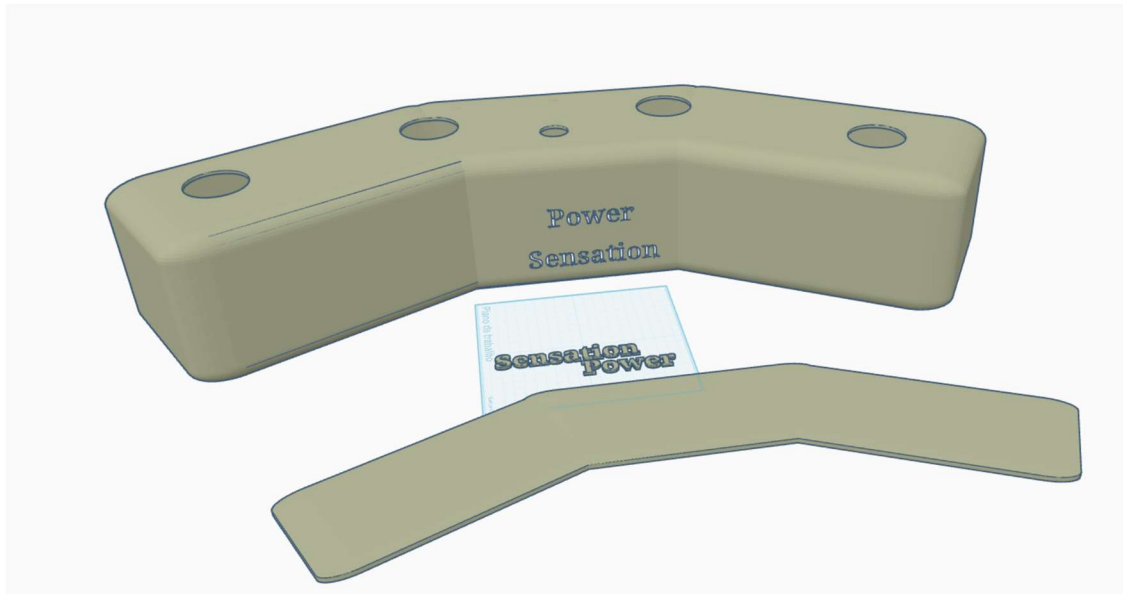


*Figura 2-Modelo 3D projetado para a consola*



*Figura 3 - Nova Consola feita*

## 2.2. Produção de Assets

A produção dos recursos visuais e sonoros do jogo apoiou-se em ferramentas de geração assistida, posteriormente integradas no projeto após tratamento. Esta abordagem permitiu obter um conjunto coerente de assets, personagens, cenários e elementos de interface, bem como a música e os efeitos sonoros sem recurso a bibliotecas externas de terceiros, garantindo controlo total sobre o estilo visual e sonoro e a sua conformidade com os princípios de acessibilidade do projeto.

### Recursos visuais

Os recursos visuais Figura 4, os sprites dos ciclistas, os obstáculos, as peças de cenário e os fundos, foram gerados através do modelo de geração de imagem da Google, Gemini (cuja funcionalidade de imagem é também conhecida por Nano Banana). Para cada

recurso foi redigido um prompt descritivo que especificava o conteúdo, o estilo e a perspectiva pretendidos, de modo a manter a coerência visual entre todos os elementos do jogo. A geração produz imagens com fundo, pelo que cada recurso foi submetido a uma etapa de pós-produção em Adobe Photoshop, na qual se removeu o fundo, isolando o elemento com transparência. Os recursos tratados foram em seguida importados para o Unity e configurados como sprites, prontos a ser utilizados pelas modalidades. Este processo aplicou-se de forma transversal a todos os elementos visuais, desde os personagens jogáveis até aos fundos de cenário.

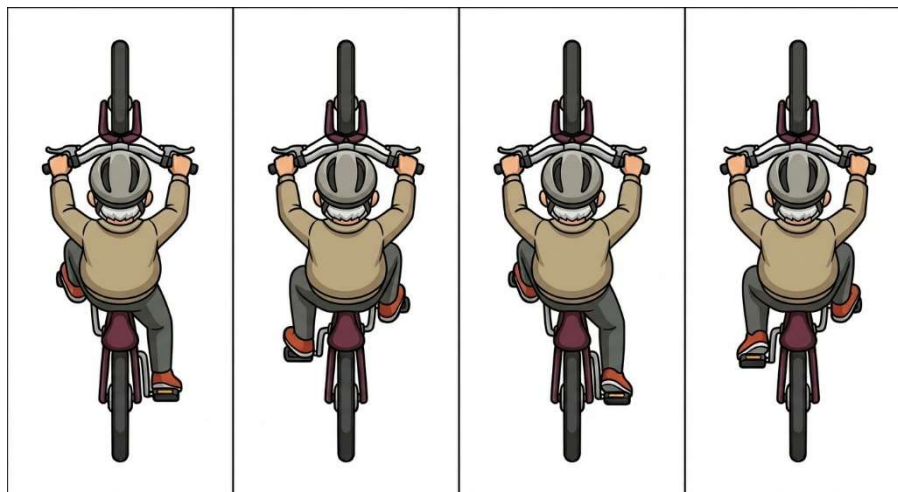


*Figura 4-Frames usados para animação do ciclista*

## **Recursos sonoros**

A música e os efeitos sonoros foram produzidos com o Chrome Music Lab Song Maker, na Figura 5, uma ferramenta web da Google para composição musical. A escolha desta ferramenta permitiu compor melodias simples e curtas, adequadas ao contexto do jogo e ao seu carácter inclusivo, e exportá-las para integração direta no projeto. Os sons resultantes foram importados para o Unity e atribuídos aos eventos correspondentes, a música de fundo, os earcons de navegação e confirmação dos menus, e os sons de acerto, vitória e derrota, geridos centralmente pelo AudioManager.

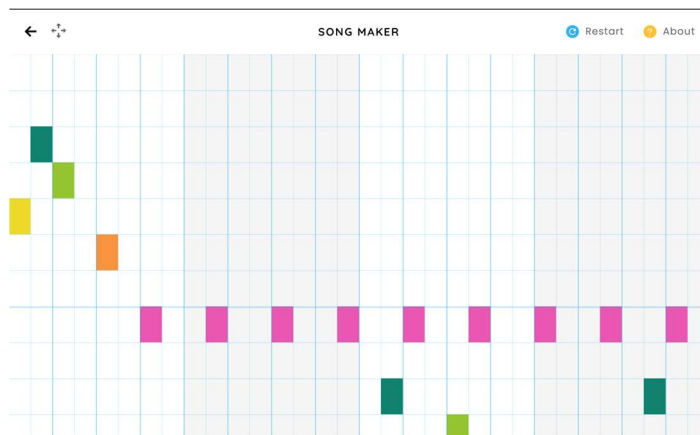


Figura 5-Composição de uma melodia no Chrome Music Lab

## 2.3. Desenvolvimento

Nesta secção apresenta-se o design do sistema na sua globalidade, articulando a componente de software (o jogo em Unity) com a componente de hardware (a consola dedicada), bem como os ecrãs, o fluxo de navegação e as ferramentas utilizadas.

### 2.3.1. Menu de Seleção de Controlos e Perfis de Input

Ao entrar no jogo, o primeiro ecrã é o de seleção de controlos, Figura 6, onde se escolhe o dispositivo de entrada que se vai utilizar: o teclado, a +Power/+PowerSensations ou a nova consola de cinco botões. Esta escolha é determinante, porque é a partir dela que todo o jogo se configura: o esquema de comandos, as indicações apresentadas ao jogador e o comportamento do feedback háptico passam a corresponder ao dispositivo seleccionado.



Figura 6 - Menu de seleção do Controlo

Cada dispositivo é descrito por um perfil de input, implementado como um recurso de dados (*ScriptableObject*) designado *InputProfile*. O código representado na Figura 7 mostra que o perfil define o mapeamento entre cada ação lógica, as de navegação de menu (confirmar, avançar, recuar) e as de jogo (virar e desviar, para cada lado), e a entrada física

concreta do dispositivo, bem como o conjunto de imagens ou ícones que representam visualmente cada botão desse dispositivo. Ao selecionar um controlo, o jogo carrega o perfil correspondente e mantém-no ativo, através de um gestor persistente (*InputManager*) que sobrevive às transições de cena, garantindo que a configuração escolhida se aplica de forma consistente a todos os ecrãs e modalidades seguintes.

```
[CreateAssetMenu(
    fileName = "InputProfile",
    menuName = "Sensation5ports/Input Profile",
    order = 0)]
public class InputProfile : ScriptableObject
{
    [Header("Identificação")]
    public string nomeAparelho = "Teclado";
    public TipoAparelho tipo = TipoAparelho.Teclado;

    [Header("Botões físicos de jogo")]
    [Tooltip("Que slots existem neste aparelho, a tecla e a imagem de cada um.")]
    public List<MapeamentoBotao> botoesJogo = new List<MapeamentoBotao>();

    [Header("Ciclismo — mapeamento das 4 ações a slots")]
    [Tooltip("Teclado/Consola3: virar e desviar do mesmo lado partilham slot. " +
        "Consola5: virar=exteriores, desviar=interiores.")]
    public List<MapeamentoAcaoCiclismo> acoesCiclismo = new List<MapeamentoAcaoCiclismo>();

    [Header("Botão de sistema")]
    [Tooltip("Dentro dos jogos: toque = ação normal; segurar = menu de toggles. " +
        "No menu principal não é usado.")]
    public Key teclaSistema = Key.Tab;
    [Tooltip("Segundos a segurar o botão de sistema para abrir o menu de toggles (dentro dos jogos).")]
    public float tempoSegurarSistema = 0.7f;

    [Header("Navegação de menus (varrimento)")]
    [Tooltip("Avança para o item seguinte (cicla). Teclado: seta direita. " +
        "+Power: Tab (Esq). Consola5: Esq.")]
    public Key teclaMenuSeguinte = Key.RightArrow;

    [Tooltip("Confirma/entra no item. Teclado: Espaço. " +
        "+Power: Enter (Dir). Consola5: Dir.")]
    public Key teclaMenuConfirmar = Key.Space;

    [Tooltip("Alterna entre a zona de cima (barra) e a de baixo (cartões). " +
        "Teclado: seta para baixo. +Power: Backspace (Centro). Consola5: Central.")]
    public Key teclaTrocarZona = Key.DownArrow;

    [Tooltip("Voltar/cancelar. Usado para sair de submenus.")]
    public Key teclaMenuVoltar = Key.Escape;

    [Header("Detecção no arranque")]
    [Tooltip("Tecla distintiva que identifica este aparelho ao ser premeida pela " +
        "primeira vez. Deve ser única entre todos os perfis.")]
    public Key teclaIdentificacao = Key.Space;

    [Tooltip("Símbolo / imagem do aparelho mostrado no ecrã de escolha de controlo.")]
    public Sprite simboloAparelho;

    [Tooltip("Texto de indicação no ecrã de escolha de controlo.")]
    [TextArea(2, 3)]
    public string textoDetecao = "Carrega em Espaço para usar o teclado";
}
```

Figura 7 - Estrutura de um perfil de input (*InputProfile*)

Esta arquitetura traz três vantagens diretas. Primeiro, desacopla a lógica do jogo do hardware: as modalidades referem-se sempre a ações lógicas e nunca a teclas concretas, pelo que acrescentar um novo dispositivo se resume a criar um perfil, sem alterar o código das modalidades. Segundo, permite que as indicações no ecrã sejam coerentes com o dispositivo real em uso, por exemplo, quando o jogo precisa de indicar uma ação, mostra a imagem do botão correto daquele controlo. Terceiro, sustenta a acessibilidade: a separação

entre ação lógica e botão físico é o que permite que o mesmo jogo seja operável tanto por teclado como pela consola adaptada, sem versões distintas.

A seleção e a persistência do perfil concretizam-se, respetivamente, no ecrã de seleção de controlos e no gestor de input.

### 2.3.2. Menu Principal e Navegação no Menu

O menu inicial, na Figura 8, foi concebido para ser inteiramente operável por um ciclo sequencial de navegação dos botões (lógica representada na Figura 9), isto permite fluidez de navegação mesmo com poucos botões e facilita para jogadores que não dependem do ecrã. A navegação é gerida pelo *GameSelectionManager* e organiza-se em duas zonas: a zona inferior, com os cartões das modalidades, e a zona superior, uma barra com os seis *Quick Toggles* de acessibilidade juntamente com acesso a botões para os menus de Controlos e o de Opções.



Figura 8 - Menu Inicial

```

private void Update()
{
    // Congela enquanto a camada de escolha de controlo está visível.
    if (camadaControlo != null && camadaControlo.Ativa) return;

    // Congela enquanto o painel de Opções está aberto.
    if (painelOpcoes != null && painelOpcoes.activeSelf) return;

    // Congela enquanto o painel de personagem está aberto.
    if (painelPersonagem != null && painelPersonagem.PainelAtivo) return;

    var teclado = Keyboard.current;
    if (teclado == null) return;

    var perfil = InputManager.Instance != null ? InputManager.Instance.PerfilAtivo : null;
    if (perfil == null) return; // sem aparelho escolhido, não navega

    // --- Trocar de zona ---
    bool trocou = teclado[perfil.teclaTrocarZona].wasPressedThisFrame;
    // Extra só no teclado: a seta de cima também troca de zona.
    if (perfil.tipo == TipoAparelho.Teclado && teclado.upArrowKey.wasPressedThisFrame)
    {
        trocou = true;
    }
    if (trocou) { TrocarZona(); return; }

    // --- Avançar item (cicla) ---
    bool avançar = teclado[perfil.teclaMenuSeguinte].wasPressedThisFrame;
    bool recuar = false;
    // Extra só no teclado: setas esquerda/direita avançam/recuam.
    if (perfil.tipo == TipoAparelho.Teclado)
    {
        if (teclado.rightArrowKey.wasPressedThisFrame) avançar = true;
        if (teclado.leftArrowKey.wasPressedThisFrame) recuar = true;
    }
    if (avançar) Navegar(+1);
    if (recuar) Navegar(-1);

    // --- Confirmar ---
    if (teclado[perfil.teclaMenuConfirmar].wasPressedThisFrame)
    {
        Confirmar();
    }
}

```

Figura 9 - Leitura de input e navegação por zonas (GameSelectionManager)

A troca entre zonas e o avanço dentro de cada zona não estão presos a teclas fixas: o gestor lê as teclas a partir do perfil de input ativo (*InputManager.PerfilAtivo*), pelo que o mesmo código funciona em teclado e em ambas as consolas. No teclado, acrescentam-se atalhos diretos (setas) por conveniência; nas consolas, usa-se o botão central para trocar de zona e os botões laterais para percorrer e confirmar. A navegação congela automaticamente enquanto estiver aberta a camada de seleção de controlo, o painel de Opções ou o painel de personagem, evitando ações acidentais.

Cada elemento de navegação é um *AccessibleButton*, que concretiza diretamente o princípio de redundância sensorial ao nível da interface: quando recebe foco, comunica o estado em vários canais em simultâneo, aumento de escala e um *earcon* de navegação emitido pelo *AudioManager*.

### 2.3.3. Menu de Opções

Na Figura 10 está o menu de Opções permite configurar em detalhe os cada um dos **QuickToggles** presentes e opções gerais do jogo.



Figura 10 - Menu de Opções

A sua navegação é gerida pelo *OpcoesNavigator* e está organizada em níveis: categorias na coluna esquerda, as opções de cada categoria na coluna direita e foco numa opção para poder percorrer cada opção com recurso a poucos botões. O gestor suporta dois modelos de entrada a partir do perfil ativo: no teclado representado pela Figura 11 onde mostra que a navegação é direta com setas, Espaço e Esc; nas consolas, uma lógica de níveis em que o botão esquerdo passa por todas as opções a cada nível, o direito desce ou opera e o botão central recua.

```

private void InputTeclado(Keyboard t)
{
    bool cima    = t.upArrowKey.wasPressedThisFrame;
    bool baixo   = t.downArrowKey.wasPressedThisFrame;
    bool esq     = t.leftArrowKey.wasPressedThisFrame;
    bool dir     = t.rightArrowKey.wasPressedThisFrame;
    bool espaco  = t.spaceKey.wasPressedThisFrame;
    bool esc     = t.escapeKey.wasPressedThisFrame;

    if (esc) { Recuar(); return; }

    switch (nivel)
    {
        case Nivel.Categorias:
            if (cima)  NavegarCategoria(-1);
            if (baixo) NavegarCategoria(+1);
            if (espaco) AbrirCategoria();
            break;

        case Nivel.Opcoes:
            if (cima)  NavegarOpcao(-1);
            if (baixo) NavegarOpcao(+1);
            if (espaco) FocarOpcao();
            break;

        case Nivel.Foco:
            var op = OpcaoAtual();
            if (op == null) break;
            // toggle: Espaço alterna. slider/ciclo: setas operam.
            if (espaco) op.Direita();           // alternar toggle / confirmar
            if (esq)   op.Esquerda();          // slider -, ciclo recua
            if (dir)   op.Direita();           // slider +, ciclo avança
            break;
    }
}

```

Figura 11 - Navegação em níveis do menu de Opções (OpcoesNavigator)

Todas as opções partilham uma base comum, a classe abstrata *OpcaoOpcoesBase* (Figura 12), que define a interface usada pela navegação sem conhecer o que cada opção faz por baixo. Sobre essa base existem três tipos opção, e cada um com o seu subtipo para o tipo de dados que devem receber:

**Sliders:** Controlam valores numéricos entre um valor mínimo e máximo, estes tipos são utilizados por: Velocidade da descrição audio, duração de legendas no ecrã e Volumes de sons representado pela Figura 13.

**Botões Booleanos:** Controlam principalmente os módulos de acessibilidades, ativando e desativando diretamente cada *Quick Toggle* correspondente como no exemplo da Figura 14.

**Opção Multipla:** Alterna entre uma lista de opções, este tipo é utilizado: nas cores de fundo e texto das legendas e no tipo de daltonismo como presente na Figura 15.

Os valores configurados persistem em estruturas próprias e compatíveis com *WebGL*. Isto permite que a navegação só conhece a base abstrata e cada opção se liga ao seu próprio alvo, torna trivial acrescentar novas opções sem alterar o navegador.

```
public abstract class OpcaoOpcoesBase : MonoBehaviour
{
    [Header("Comum")]
    [SerializeField] protected TMP_Text rotulo; // nome da opção
    [SerializeField] protected GameObject destaqueFoco; // moldura/realce quando focada (opcional)

    /// <summary>Texto do rótulo (ex.: "Volume Geral", "Ativo").</summary>
    [SerializeField] protected string nomeOpcao = "Opção";

    protected virtual void Start()
    {
        if (rotulo != null) rotulo.text = nomeOpcao;
        AtualizarVisual();
        MostrarFoco(false);
    }

    /// <summary>Chamado quando a navegação foca/desfoca esta opção (Nível 3).</summary>
    public virtual void MostrarFoco(bool focado)
    {
        if (destaqueFoco != null) destaqueFoco.SetActive(focado);
    }

    /// <summary>Esquerda dentro do foco (ex.: slider -, ciclo recua). Override conforme o tipo.</summary>
    public virtual void Esquerda() { }

    /// <summary>Direita dentro do foco (ex.: slider +, ciclo avança, toggle alterna).</summary>
    public virtual void Direita() { }

    /// <summary>Atualiza a parte visual ao valor atual (estado on/off, barra, texto do ciclo).</summary>
    public abstract void AtualizarVisual();
}
```

Figura 12 – Base abstrata das opções (OpcaoOpcoesBase)

```
private void Ajustar(float delta)
{
    float novo = Mathf.Clamp01(Ler() + delta);
    Escrever(novo);
    AtualizarVisual();
    AudioManager.Instance?.AplicarVolumes(); // reaplica ao som já a tocar
    AudioManager.Instance?.TocarNavegacao();
}
```

Figura 13 - Opção de volume (OpcaoVolume)

```
private void Alternar()
{
    if (AccessibilityManager.Instance == null) return;
    AccessibilityManager.Instance.Alternar(modulo);
    AtualizarVisual();
    AudioManager.Instance?.TocarConfirmacao();
}
```

Figura 14 - Opção de módulo ativo (OpcaoModuloAtivo)

```
private void Avançar()
{
    if (config == null) return;
    config.Avançar();
    AtualizarVisual();
    AudioManager.Instance?.TocarConfirmacao();
}
```

Figura 15 - Opção de tipo de daltonismo (OpcaoDaltonismoTipo)

#### 2.3.4. Sistema de Progressão por Modalidade

Cada modalidade tem uma progressão própria, expressa em pontos de experiência (XP) acumulados e num nível derivado desses pontos. A progressão é gerida pela classe estática *ProgressaoModalidade*, que guarda o XP total de cada modalidade diretamente em *PlayerPrefs*, com chamada explícita a *Save()* para garantir a persistência em *WebGL*, como representado na Figura 16. O nível é sempre calculado a partir do XP total (cem pontos por nível, até um máximo de dez níveis), pelo que o XP nunca é repostado ao subir de nível, guarda-se apenas o total acumulado, o que torna este sistema simples e robusto como representado na Figura 17.

```
public static int AdicionarXP(string modalidade, int quantidade)
{
    if (quantidade <= 0) return 0;
    int atual = XP(modalidade);
    if (atual >= XP_MAXIMO) return 0; // já no máximo, não acumula

    int novo = Mathf.Min(atual + quantidade, XP_MAXIMO);
    PlayerPrefs.SetInt(Chave(modalidade), novo);
    PlayerPrefs.Save(); // explícito — WebGL
    return novo - atual;
}
```

Figura 16 - Acumulação persistente de XP (*ProgressaoModalidade*)

```
/// <summary>Nível atual derivado do XP (0..10).</summary>
public static int Nivel(string modalidade) =>
    Mathf.Min(XP(modalidade) / XP_FOR_NIVEL, NIVEL_MAXIMO);

/// <summary>XP dentro do nível atual (0..99). No máximo, fica em 0.</summary>
public static int XPNoNivel(string modalidade)
{
    int xp = XP(modalidade);
    if (xp >= XP_MAXIMO) return 0;
    return xp % XP_FOR_NIVEL;
}
```

Figura 17 - Cálculo do nível a partir do XP (*ProgressaoModalidade*)

Esta progressão é apresentada ao jogador nos próprios cartões do menu. O CardView lê o XP e o nível atuais da modalidade e atualiza o texto de nível, a barra de progresso e o estado do cartão. Este objeto é gerado ao carregar o Menu Inicial, demonstrado na Figura 18. Cada cartão é configurado ao gerar de acordo com cada modalidade registada no jogo.

```
public void Configurar(ModalidadeData dados)
{
    Dados = dados;

    if (titulo != null) titulo.text = dados.nomeModalidade;
    if (descricao != null) descricao.text = dados.descricao;

    // progresso REAL, do PlayerPrefs
    string chave = dados.nomeModalidade;
    int nivel = ProgressaoModalidade.Nivel(chave);
    int xpTotal = ProgressaoModalidade.XP(chave);

    if (nivelXP != null)
    {
        if (ProgressaoModalidade.NoMaximo(chave))
            nivelXP.text = $"Nível {nivel} · {xpTotal}/{ProgressaoModalidade.XP_MAXIMO}";
        else
        {
            int limiarSeguinte = (nivel + 1) * ProgressaoModalidade.XP_POR_NIVEL;
            nivelXP.text = $"Nível {nivel} · {xpTotal}/{limiarSeguinte}";
        }
    }

    if (barraXP != null)
    {
        barraXP.minValue = 0f;
        barraXP.maxValue = 1f;
        // fração dentro do nível atual (0..1)
        barraXP.value = ProgressaoModalidade.NoMaximo(chave)
            ? 1f
            : ProgressaoModalidade.XPNoNivel(chave) / (float)ProgressaoModalidade.XP_POR_NIVEL;
    }

    if (preview != null)
    {
        if (dados.preview != null) { preview.sprite = dados.preview; preview.enabled = true; }
        else preview.enabled = false;
    }

    if (textoBtnEntrar != null)
        textoBtnEntrar.text = dados.TemCena ? textoDisponivel : textoBrevemente;

    Selecionar(false);
}
```

Figura 18 - Apresentação da progressão no cartão (CardView)

### 2.3.5. Arquitetura Escalável de Modalidades

O jogo foi estruturado de modo a permitir acrescentar novas modalidades sem reescrever o seu núcleo. Cada modalidade é descrita por um recurso de dados do tipo *ModalidadeData* (Figura 19), que define os seus parâmetros como: nome, descrição, imagem de pré-visualização e a cena de jogo associada; evitando valores fixos no código e permitindo que uma nova modalidade seja adicionada apenas criando um *asset* (Figura 20).

```

[CreateAssetMenu(
    fileName = "ModalidadeData",
    menuName = "SensationSports/Modalidade",
    order = 1)]
public class ModalidadeData : ScriptableObject
{
    public const int NIVEL_MAXIMO = 10;

    [Header("Identificação")]
    public string nomeModalidade = "Ciclismo";
    [TextArea(2, 4)]
    public string descricao = "Pedala em percursos progressivos. Controla a direção do ciclista para não sair da pista!";

    [Tooltip("Imagem de pré-visualização mostrada no topo do cartão.")]
    public Sprite preview;

    [Header("Progressão (por modalidade, máx. 10 níveis)")]
    [Range(1, NIVEL_MAXIMO)]
    public int nivel = 1;
    [Tooltip("XP acumulado no nível atual.")]
    public int xpAtual = 0;
    [Tooltip("XP necessário para passar de nível.")]
    public int xpNecessario = 100;

    [Header("Desbloqueio de personagens (níveis fixos)")]
    [Tooltip("Nível a que desbloqueia o 2.º personagem desta modalidade.")]
    public int nivelPersonagem2 = 3;
    [Tooltip("Nível a que desbloqueia o 3.º personagem desta modalidade.")]
    public int nivelPersonagem3 = 7;
    // NOTA: a lista dos 3 personagens (com bónus próprios) entra aqui
    // quando construirmos o ecrã de seleção de personagem.

    [Header("Cena")]
    [Tooltip("Nome exato da cena a carregar ao entrar (tem de estar nos Build Settings). " +
        "Deixar vazio enquanto a cena não existir — o menu trata isso sem rebaratar.")]
    public string nomeCena = "Ciclismo";

    [Header("Personagens desta modalidade")]
    public System.Collections.Generic.List<PersonagemData> personagens =
        new System.Collections.Generic.List<PersonagemData>();
}

```

Figura 19 - Estrutura de dados de uma modalidade (ModalidadeData)

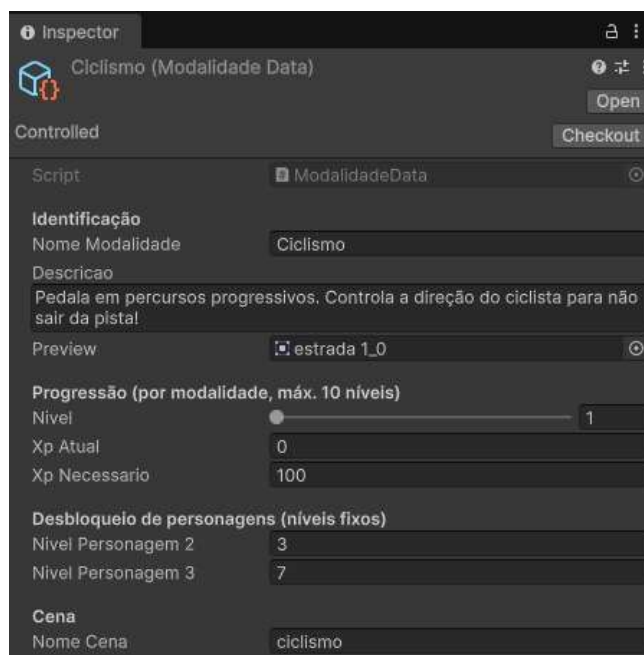


Figura 20 - Asset de modalidade no inspetor do Unity

Na Figura 21 mostra como o menu de seleção é orquestrado pelo *GameSelectionManager*, este percorre a lista de modalidades e instancia dinamicamente um cartão (*CardView*) para cada uma.

```

private void GerarCartoes()
{
    if (prefabCartao == null || painelCartoes == null)
    {
        Debug.LogError("[GameManager] Falta o prefab do cartão ou o painel.");
        return;
    }

    foreach (var dados in modalidades)
    {
        if (dados == null) continue;

        CardView cartao = Instantiate(prefabCartao, painelCartoes);
        cartao.name = $"Card_{dados.nomeModalidade}";
        cartao.Configurar(dados);

        int indiceDeste = cartoes.Count;

        var botaoCartao = cartao.GetComponent<Button>();
        if (botaoCartao != null)
        {
            botaoCartao.onClick.AddListener(() =>
            {
                zona = Zona.Baixo;
                SelecionarIndice(indiceDeste);
            });
        }

        if (cartao.BotaoEntrar != null)
        {
            cartao.BotaoEntrar.onClick.AddListener(() => EntrarNaModalidade(dados));
        }

        cartoes.Add(cartao);
    }
}

```

Figura 21 - Geração dinâmica dos cartões de modalidade (GameSelectionManager)

Quando o jogador confirma uma modalidade, o gestor não carrega de imediato a cena: encaminha primeiro para a seleção de personagem. A escolha da modalidade e do personagem é guardada no *SelecaoJogo*, um *singleton* que sobrevive à transição de cena através de *DontDestroyOnLoad*, representado na Figura 22, de forma que a cena de jogo correspondente possa depois ler essa escolha e aplicá-la. Cada cena de jogo tem o seu próprio gestor, que lê o *SelecaoJogo* e gere o estado dessa partida XP da ronda, progressão e condição de vitória/derrota de forma independente.

```

private void Awake()
{
    if (Instance != null && Instance != this)
    {
        Destroy(gameObject);
        return;
    }
    Instance = this;
    DontDestroyOnLoad(gameObject);
}

/// <summary>Define a modalidade (ao clicar "Entrar" num cartão).</summary>
public void DefinirModalidade(ModalidadeData modalidade)
{
    ModalidadeSelecionada = modalidade;
}

/// <summary>Define o personagem (ao escolher no painel) — não carrega a cena.</summary>
public void DefinirPersonagem(PersonagemData personagem)
{
    PersonagemSelecionado = personagem;
}

```

Figura 22 - Persistência da escolha entre cenas (SelecaoJogo)

### 2.3.6. Gestão de Áudio

Todos os sons do jogo são geridos pelo *AudioManager*, este objeto é uma *singleton* que persiste em todos os ecrãs e é responsável por tocar os sons de efeitos e a música do

jogo. Todos os efeitos são tocados através da função *PlayOneShot*, como representado na Figura 23. Sempre que o volume da música é ajustado, ele chama a função *AplicarVolumes*, como na Figura 24, para alterar o volume da música assim que o jogador começa a mexer no *slider*.

```
public void TocarNavegacao()
{
    if (somNavegacao != null)
        fonteEfeitos.PlayOneShot(somNavegacao, ConfigVolumes.Efeitos);
}

public void TocarConfirmacao()
{
    if (somConfirmacao != null)
        fonteEfeitos.PlayOneShot(somConfirmacao, ConfigVolumes.Efeitos);
}
```

Figura 23 - Efeitos de navegação e confirmação (AudioManager)

```
private void IniciarMusica()
{
    if (musicaFundo == null || fonteMusica == null) return;
    fonteMusica.clip = musicaFundo;
    fonteMusica.volume = ConfigVolumes.Musica;
    fonteMusica.Play();
}

/// <summary>Reaplica os volumes atuais (So para a musica). Chamar quando um slider muda.</summary>
public void AplicarVolumes()
{
    if (fonteMusica != null)
        fonteMusica.volume = ConfigVolumes.Musica;
}
```

Figura 24 - Gestão de volumes e música de fundo (AudioManager)

### 2.3.7. Seleção de Personagem

Na seleção de personagem, o jogador escolhe entre três personagens, cada uma com uma habilidade passiva distinta que ajusta a exigência do jogo: a personagem padrão, sem modificador; a segunda, com tempo de reação aumentado; e a terceira, que reduz a exigência dos desafios. Como mostrado na Figura 25 as personagens são desbloqueadas conforme a progressão.

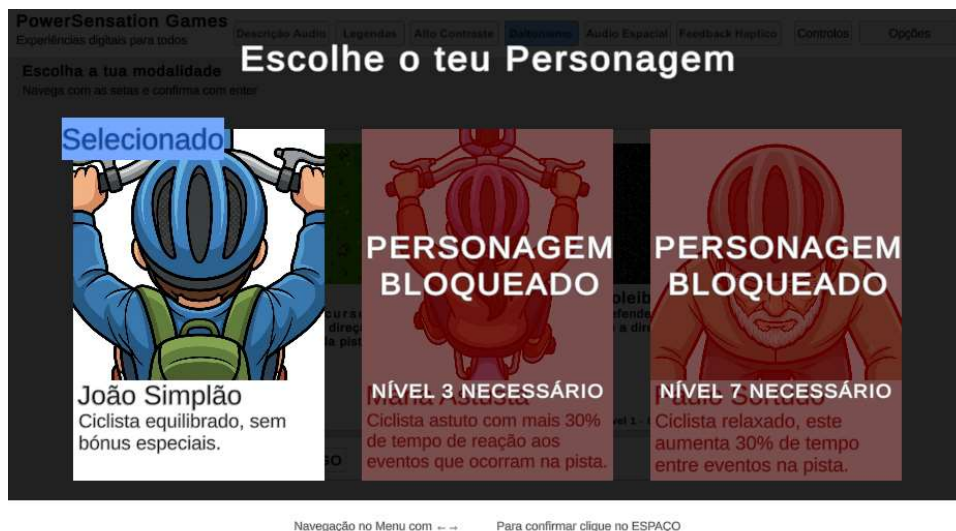


Figura 25 - Menu Seleção de Personagens

### 2.3.8. Tutorial

Ao entrar numa modalidade o jogador é deparado com um ecrã de tutorial que tem uma imagem central que explica as mecânicas, similar ao botão do menu ao clicar no botão de navegar ele troca de imagem para o utilizador observar e aprender como funciona o jogo e só depois clica no botão de confirmar para realmente iniciar o jogo.

### 2.3.9. Modalidade de Ciclismo

A primeira modalidade implementada é o Ciclismo, na Figura 26 podemos perceber que este foi concebido em perspetiva *top-down*, no qual o jogador controla um ciclista ao longo de uma ciclovia que se desloca automaticamente de cima para baixo no ecrã. Foi escolhida como ponto de partida pela simplicidade do seu esquema de controlo, que permite validar a totalidade da arquitetura de acessibilidade antes de avançar para interações mais complexas.



Figura 26 - Modalidade do Ciclismo

### Arquitetura de scripts

A modalidade está repartida por componentes com responsabilidades únicas, o que mantém o código legível e facilita a manutenção e os testes. Cada script tem um papel bem delimitado:

**GameManager:** gestor central da cena: mantém o estado da corrida, que é composto por tempo de sobrevivência, XP da ronda, vitória/derrota; calcula a dificuldade e aplica os bónus do personagem;

**RoadScroller:** desloca a ciclovia e gere o aparecimento da meta;

**CyclistController:** lê o input do perfil ativo e move o ciclista;

**EventManager:** gere a ordem e faz os eventos aparecerem no ecrã juntamente com o controlo de ação do jogador perante o tempo de reação;

**Bussola:** mantém e apresenta a orientação cardeal;

**IndicacaoEvento:** apresenta a indicação redundante (texto + botão + som) do evento atual;

**ObstacleEvent:** responsável por instanciar e deslocar o obstáculo no ecrã;

**CameraEffects:** faz o efeito de inclinação através da câmara nas curvas;

**UIManager:** apresenta no ecrã o XP acumulado durante a ronda e mostra o painel de fim de jogo;

**GameOverNavigator:** este é responsável para permitir a navegação entre botões no ecrã de gameover.

### Deslocamento contínuo da ciclovia (RoadScroller)

A sensação de avanço é criada pelo *RoadScroller*, que desloca para baixo um conjunto de peças de estrada à velocidade definida pelo *GameManager* (Figura 27). Quando uma peça sai do ecrã por baixo, é reposicionada no topo, formando um ciclo infinito com poucos *sprites*, uma técnica eficiente que evita instanciar estrada continuamente. A altura de cada peça é medida automaticamente a partir dos seus limites, tornando o sistema independente da resolução do *sprite*. Quando o *EventManager* decide que a próxima é a meta o *Roadscroller* tem um dos seus valores atualizado e como mostrado na Figura 28 ele irá colocar o próximo *sprite* como a meta e irá parar de dar o *scroll*.

```
[Header("Dificuldade — Velocidade")]
public float baseScrollSpeed = 3f;
public float currentScrollSpeed;
public float speedIncreasePerSecond = 0.05f;
public float maxScrollSpeed = 12f;
```

Figura 27 - Velocidade do scroll (*GameManager*)

```
// se foi pedida a meta, a PRÓXIMA peça a reciclar torna-se a meta
if (metaPedida && peçaMeta == null && metaSprite != null)
{
    peçaMeta = piece;
    piece.sprite = metaSprite;
    metaPedida = false;
}
```

Figura 28 - Colocar a "meta" no ecrã (*RoadScroller*)

### Controlo do ciclista e leitura de input (*CyclistController*)

O ciclista é comandado por quatro ações lógicas, virar à esquerda, virar à direita, desviar à esquerda e desviar à direita, estas são representadas pela enumeração *AcaoCiclismo*. O *CyclistController* não lê teclas fixas: a cada *frame* consulta o perfil de input ativo, indicado pelo *InputManager.PerfilAtivo* e, para cada ação, verifica se a tecla mapeada nesse perfil foi premida, como demonstrado na Figura 29, e posteriormente comunicando-a ao *EventManager* (Figura 30). Isto é o que permite que o mesmo código funcione em teclado e nas duas consolas, concretizando a separação entre ação lógica e botão físico que sustenta a acessibilidade do jogo [1].

```
// Testa cada ação: se a tecla mapeada foi premida este frame, comunica.
TestarAcao(teclado, perfil, AcaoCiclismo.VirarEsq);
TestarAcao(teclado, perfil, AcaoCiclismo.VirarDir);
TestarAcao(teclado, perfil, AcaoCiclismo.DesviarEsq);
TestarAcao(teclado, perfil, AcaoCiclismo.DesviarDir);
```

Figura 29 - Teste se clicou na tecla para uma ação (*CyclistController*)

```
private void TestarAcao(Keyboard teclado, InputProfile perfil, AcaoCiclismo acao)
{
    if (!perfil.TryTeclarAcao(acao, out Key tecla)) return;
    if (tecla == Key.None) return;
    if (teclado[tecla].wasPressedThisFrame)
    {
        EventManager.Instance.OnPlayerAction(acao);
    }
}
```

Figura 30 - Validar ação e enviar para o EventManager (CyclistController)

### Ciclo de geração de eventos (EventManager)

O ritmo do jogo é dado pelo *EventManager* através de um *EventLoop* que corre durante toda a corrida (Figura 31). Entre eventos, espera um intervalo definido pelo *GameManager* com uma pequena variação aleatória e só lança o evento seguinte quando não há nenhum evento, obstáculo ou ação do ciclista ainda em curso. Esta guarda garante que nunca há dois eventos sobrepostos, mantendo a leitura do jogo inequívoca em todos os canais sensoriais, um requisito essencial para jogadores que dependem do áudio ou do tato [4].

```
IEnumerator EventLoop()
{
    while (!GameManager.Instance.isGameOver)
    {
        float wait = GameManager.Instance.IntervaloEventos + Random.Range(-0.5f, 0.5f);
        yield return new WaitForSeconds(Mathf.Max(0.5f, wait));

        while (eventActive
            || obstacleEvent.IsActive()
            || cyclist.currentAction != CyclistController.ActionType.None)
        {
            yield return null;
        }

        if (GameManager.Instance.isGameOver) break;

        // --- Teste da meta ---
        if (!metaLancada && eventosFeitos >= metaMinEventos)
        {
            float chance = metaChanceBase + GameManager.Instance.survivalTime * metaChancePorSegundo;
            if (Random.value < chance)
            {
                LancarMeta();
                break; // a meta é a última coisa: para o ciclo de eventos
            }
        }

        TriggerRandomEvent();
        eventosFeitos++;
    }
}
```

Figura 31 - Ciclo de geração de eventos sem sobreposição (EventManager)

Cada evento é de um de dois tipos, sorteado aleatoriamente: virar para fazer uma curva ou desviar para evitar um obstáculo. Quando um evento é iniciado, o jogador dispõe de uma janela de tempo, mais ampla para as curvas, por exigirem leitura da bússola, e mais curta para os desvios, de natureza reflexa, medida através no *Update* (Figura 32) do gestor,

durante a qual a barra de reação se esvazia. Se a janela terminar sem resposta correta, o jogo termina.

```
void Update()
{
    if (GameManager.Instance.isGameOver)
    {
        indicacao?.Esconder();
        return;
    }

    if (!eventActive) return;

    eventTimer += Time.deltaTime;

    if (eventTimer >= eventDuration)
        GameManager.Instance.GameOver();

    UIManager.Instance.UpdateReactionBar(1f - (eventTimer / eventDuration));
}
```

Figura 32 - Janela de reação do evento (EventManager)

### Validação da resposta e tolerância por dispositivo (EventManager)

Quando o jogador atua, o *EventManager* compara a ação recebida com a ação correta (*OnPlayerAction*). Há aqui uma decisão de acessibilidade importante: na consola de cinco botões, em que cada ação tem um botão único, a verificação é rigorosa, a ação tem de ser exatamente a esperada; no teclado e na consola de três botões, em que virar e desviar de um lado partilham tecla, basta acertar no lado correto Figura 33. Esta tolerância adapta à exigência ao número de botões do dispositivo, evitando penalizar quem joga com menos botões, uma resposta direta à lacuna de controlos flexíveis identificada na análise de acessibilidade Trick et al. [3].

```

public void OnPlayerAction(AcaoCiclismo acao)
{
    if (!eventoAtivo) return;

    var perfil = InputManager.Instance != null ? InputManager.Instance.PerfilAtivo : null;
    bool rigoroso = perfil != null && perfil.tipo == TipoAparelho.Consola5;

    bool acertou = rigoroso
        ? (acao == acaoCorreta)
        : (LadoDe(acao) == LadoDe(acaoCorreta));

    if (acertou)
    {
        if (currentEvent == EventType.Turn) ResolverVirar();
        else if (currentEvent == EventType.Obstacle) ResolverDesviar();
    }
    else
    {
        GameManager.Instance.GameOver();
    }
}

```

Figura 33 - Validação da resposta adaptada ao dispositivo (EventManager)

### Indicação redundante do evento (IndicacaoEvento)

Cada evento é comunicado em vários canais em simultâneo, concretizando o princípio de redundância sensorial [1]. Visualmente, o *IndicacaoEvento* apresenta uma frase, “Virar para Este” ou “Desviar para a direita”, acompanhada da imagem do botão a premir, que pisca para atrair o olhar (Figura 34). A imagem do botão é obtida do perfil ativo, pelo que corresponde sempre ao dispositivo em uso. A própria janela de tempo é redundante: além do texto e da imagem, é representada pela barra de reação que se esvazia que se observa na Figura 35. Note-se que a indicação de virar usa referência absoluta do ponto cardeal e a de desviar usa referência relativa de esquerda ou direita, uma distinção coerente com a natureza de cada ação.

```

public void MostrarVirar(Bussola.Cardeal destino, BotaoSlot slot,
    CyclistController.Direction lado)
{
    if (texto != null)
        texto.text = $"Virar para {Bussola.Nome(destino)}";

    MostrarImagemBotao(slot);
    Ativar();

    TocarVoz(ClipVirar(destino), lado);
}

```

Figura 34 - Indicação redundante do evento (IndicacaoEvento)



Figura 35 - Momento de ação no ecrã (Ciclismo)

### Obstáculos (ObstacleEvent)

Nos eventos de desvio, o *ObstacleEvent* instância um obstáculo num dos lados da ciclovia e desloca-o para baixo à mesma velocidade da estrada (Figura 36). O obstáculo aparece do lado oposto àquele para onde o jogador se deve desviar; se o desvio é bem-sucedido, o obstáculo é marcado como ultrapassado e, ao sair do ecrã, liberta o ciclista da posição de desvio.

```
public void Show(CyclistController.Direction side)
{
    if (currentObstacle != null)
    {
        Destroy(currentObstacle);
        currentObstacle = null;
    }

    float x = (side == CyclistController.Direction.Left) ? leftX : rightX;
    currentObstacle = Instantiate(obstaclePrefab,
        new Vector3(x, spawnY, 0), Quaternion.identity);
    dodgeSuccessful = false;
    ladoObstaculoAtual = side;

    IniciarSom(side);
}
```

Figura 36 - Geração e deslocamento de obstáculos (ObstacleEvent)

## Dificuldade progressiva e XP (GameManager)

A dificuldade aumenta de forma contínua com o tempo de sobrevivência (Figura 37): a velocidade de *scroll* cresce até um limite máximo e as janelas de reação para virar e desviar encurtam até um mínimo, tal como o intervalo entre eventos. Os valores efetivos incorporam ainda os multiplicadores do personagem, o que permite que as personagens desbloqueáveis tornem o jogo mais acessível, por exemplo, concedendo mais tempo de reação. Este ajuste dinâmico de dificuldade sustenta o equilíbrio entre desafio e competência ao longo da partida [5].

Durante a corrida, o jogador acumula XP da ronda: por cada desvio, por cada curva, por tempo de sobrevivência e por chegar à meta. No fim, esse XP é somado à progressão persistente da modalidade, gravada em *PlayerPrefs*. A recompensa explícita e a progressão visível entre rondas reforçam a motivação, em linha com as implicações de design dos jogos sérios estudados [6].

```
void Update()
{
    if (isGameOver) return;

    survivalTime += Time.deltaTime;

    // XP por tempo
    timerXP += Time.deltaTime;
    if (timerXP >= intervaloXP)
    {
        timerXP -= intervaloXP;
        xpRonda += xpPorIntervalo;
    }

    currentScrollSpeed = Mathf.Min(
        baseScrollSpeed + survivalTime * speedIncreasePerSecond, maxScrollSpeed);

    currentReactionVirar = Mathf.Max(
        baseReactionVirar - survivalTime * reactionVirarDecrease, minReactionVirar);

    currentReactionDesviar = Mathf.Max(
        baseReactionDesviar - survivalTime * reactionDesviarDecrease, minReactionDesviar);

    currentEventInterval = Mathf.Max(
        baseEventInterval - survivalTime * intervalDecreasePerSecond, minEventInterval);
}
```

Figura 37 – Aumento de Velocidade com o Tempo (GameManager)

## Duração variável e linha de meta (EventManager)

O jogo não é infinito: tem duração variável e uma condição de vitória. Depois de um número mínimo de eventos, passa a existir uma probabilidade, crescente com o tempo, de a rotina de eventos lançar a meta em vez de mais um evento (Figura 38). Quando tal acontece, o *EventManager* pede ao *RoadScroller* para transformar a próxima peça reciclada na peça de meta; quando essa peça cruza a posição do ciclista, o *GameManager* declara vitória. Esta mecânica dá um objetivo claro e um encerramento positivo à corrida, em vez de uma

repetição indefinida, alinhando-se com a importância de experiências de conclusão e de recompensa [6].

```
// --- Teste da meta ---  
if (!metaLancada && eventosFeitos >= metaMinEventos)  
{  
    float chance = metaChanceBase + GameManager.Instance.survivalTime * metaChancePorSegundo;  
    if (Random.value < chance)  
    {  
        LancarMeta();  
        break; // a meta é a última coisa: para o ciclo de eventos  
    }  
}
```

Figura 38 – Decisão probabilística da linha de meta (EventManager)

### Câmara e feedback de curva (CameraEffects)

Para acentuar a sensação de curva, o *CameraEffects* inclina ligeiramente a câmara para o lado da viragem e repõe-na suavemente (Figura 39), um reforço visual do evento que complementa, sem substituir, os restantes canais sensoriais.

```
public void StartCurve(CyclistController.Direction dir, float duration)  
{  
    inCurve = true;  
    curveTimer = duration;  
    targetRotation = (dir == CyclistController.Direction.Left)  
        ? -curveRotationAmount  
        : curveRotationAmount;  
}
```

Figura 39 - Inclinação da câmara nas curvas (CameraEffects)

### Fim de jogo (UIManager)

No fim da corrida, o *UIManager* apresenta o painel de fim de jogo (Figura 40) com o resultado, “Concluiu a corrida” ou “Não encontrou a meta desta vez”, o XP ganho e o nível atualizado da modalidade Figura 41. A navegação deste painel, com as opções Reiniciar e Voltar ao menu, segue o mesmo modelo acessível dos restantes menus.



Navegação no Menu com ← → ↑ ↓ Para confirmar clique no ESPAÇO

Figura 40 - Ecrã de fim de jogo (ciclismo)

```
public void ShowGameOver(int xpGanho, bool venceu)
{
    if (gameOverPanel != null) gameOverPanel.SetActive(true);

    if (resultadoText != null)
        resultadoText.text = venceu
            ? "Concluiu a corrida"
            : "Não encontrou a meta desta vez";

    if (xpGanhoText != null)
        xpGanhoText.text = "+" + xpGanho + " EXP";

    var mod = SeleccionJogo.Instance != null
        ? SeleccionJogo.Instance.ModalidadeSeleccionada
        : null;

    if (nivelText != null && mod != null)
    {
        string chave = mod.nomeModalidade;
        int nivel = ProgressaoModalidade.Nivel(chave);
        int xpTotal = ProgressaoModalidade.XP(chave);

        if (ProgressaoModalidade.NoMaximo(chave))
            nivelText.text = $"Nível {nivel} · {xpTotal}/{ProgressaoModalidade.XP_MAXIMO} (Máximo)";
        else
        {
            int limiarSeguinte = (nivel + 1) * ProgressaoModalidade.XP_POR_NIVEL;
            nivelText.text = $"Nível {nivel} · {xpTotal}/{limiarSeguinte}";
        }
    }

    if (gameOverNav != null) gameOverNav.Abrir();
}
```

Figura 41 – Lógica do fim de jogo (UIManager)

### 2.3.10. QuickToggle: Descrição Áudio

A descrição áudio constitui o canal de saída sonora para informação textual, dirigido sobretudo a jogadores com deficiência visual. Quando ativo, este módulo lê em voz alta os elementos da interface à medida que recebem foco, bem como as indicações de jogo, recorrendo à síntese de voz do navegador através do *Web Speech API* que faz uma ponte para JavaScript. A opção pela síntese do navegador, em detrimento de ficheiros pré-gravados, justifica-se pela natureza dinâmica do conteúdo a ler, nomes de modalidades, valores de opções, estados de toggles; e pela facilidade de implementação e flexibilidade.

A leitura é gerida pelo *TTSTManager* (Figura 42), um gestor persistente que expõe um método de leitura único e que verifica, antes de qualquer locução, se o módulo está ativo; cada nova leitura interrompe a anterior, replicando o comportamento esperado de um leitor de ecrã, em que o foco mais recente cancela o anterior. A velocidade da locução é configurável pelo jogador, sendo lida de uma estrutura persistente própria e aplicada a cada locução, o que permite adaptar o ritmo da fala às preferências individuais.

```
public class TTSTManager : MonoBehaviour
{
    public static TTSTManager Instance { get; private set; }

    #if UNITY_WEBGL && !UNITY_EDITOR
    [DllImport("__Internal")] private static extern void TT5_Falar(string texto, float rate);
    [DllImport("__Internal")] private static extern void TT5_Parar();
    #endif

    private void Awake()
    {
        if (Instance != null && Instance != this)
        {
            Destroy(gameObject);
            return;
        }
        Instance = this;
        DontDestroyOnLoad(gameObject);
    }

    /// <summary>Lê um texto, se a Descrição Áudio estiver ligada. Interrompe o anterior.</summary>
    public void Falar(string texto)
    {
        if (string.IsNullOrEmpty(texto)) return;

        var acc = AccessibilityManager.Instance;
        if (acc == null || !acc.Esta(ModuloAcessibilidade.DescricaoAudio)) return;

    #if UNITY_WEBGL && !UNITY_EDITOR
        TT5_Falar(texto, ConfigTT5.Rate);
    #else
        Debug.Log($"[TT5 rate={ConfigTT5.Rate}] {texto}");
    #endif
    }

    public void Parar()
    {
    #if UNITY_WEBGL && !UNITY_EDITOR
        TT5_Parar();
    #else
        Debug.Log("[TT5] (parar)");
    #endif
    }
}
```

Figura 42 - Código Integral do TTSTManager

Como demonstrado na Figura 43 a integração com a interface concretiza-se em dois planos. No primeiro, cada elemento navegável transporta o texto que deve ser dito: ao receber foco, o próprio elemento solicita a sua leitura, o que mantém a descrição áudio coerente com a navegação sem código duplicado. No segundo, o menu de seleção de controlo e o menu de Opções emitem locuções contextuais, por exemplo, ao escolher um controlo é lida uma frase que explica como navegar com esse dispositivo, e no ajuste de uma opção é anunciado o valor corrente. Este módulo encontra-se ativo por predefinição, uma decisão de acessibilidade que garante que o canal está disponível desde o primeiro arranque, sem exigir configuração prévia por parte de quem dele depende.

```
private void Focar(bool valor)
{
    if (focado == valor) return;
    focado = valor;
    AplicarEstado(valor, instantaneo: false);

    if (valor)
    {
        if (emitirSomNavegacao)
            AudioManager.Instance?.TocarNavegacao();

        // descrição auditiva do item focado
        if (!string.IsNullOrEmpty(textoTT5))
            TTSManager.Instance?.Falar(textoTT5);

        HapticoManager.Instance?.Navegar();
    }
}
```

Figura 43 - Verificação do texto para a Descrição Audio (AccessibleButton)

### 2.3.11. QuickToggle: Legendas

O módulo de legendas materializa o canal visual textual para informação sonora, dirigido a jogadores surdos ou com dificuldades auditivas (DHH). Distingue-se de uma legendagem convencional por aplicar a tipologia de descrição não-verbal para surdos (NV-SDH) fundamentada em Projeto I [1], [7]: além de transcrever as instruções verbais, descreve também os sons não-verbais do jogo, comunicando o ambiente sonoro a quem não o ouve.

O sistema, gerido pelo *LegendasManager*, distingue duas naturezas de legenda, como demonstrado na Figura 44. As de fala apresentam o texto da instrução tal como é enunciado ("Desviar para a direita"). As de som descrevem um evento sonoro não-verbal e são apresentadas entre parênteses retos e em itálico, segundo a convenção NV-SDH, por exemplo, a aproximação de um obstáculo, o som de acerto, ou a música de vitória e o som de derrota. As legendas surgem na zona inferior central do ecrã, por cima de toda a restante interface, e empilham-se quando vários eventos coincidem, mantendo-se a mais recente na posição inferior; cada legenda tem duração própria e desaparece de forma independente.



Figura 44 - Ciclismo com as Legendas Ativas

A duração no ecrã e o esquema cromático são configuráveis. O jogador pode ajustar quanto tempo cada legenda permanece e trocar a cor de fundo (entre preto e branco) e as cores do texto de fala e de som (entre branco, preto, amarelo e azul), o que permite adaptar o contraste às suas necessidades, uma flexibilidade alinhada com a recomendação de personalização de legendas identificada na literatura [3], [7]. As legendas de som de duração indeterminada, como o aviso de obstáculo, persistem enquanto o evento decorre e são removidas quando este termina, ao passo que as de evento pontual usam a duração configurada.

### 2.3.12. QuickToggle: Áudio Espacial

O áudio espacial acrescenta informação direcional ao canal sonoro, comunicando pela posição do som de onde provém um evento. Destina-se sobretudo a jogadores com deficiência visual, para quem a lateralidade da fonte sonora substitui a leitura visual da posição. Quando ativo, este módulo posiciona o som no campo estéreo (deslocação para a esquerda ou direita) consoante o lado do evento.

O módulo manifesta-se em dois sons distintos do ciclismo. O primeiro é o alerta de obstáculo: um sinal sonoro curto, repetido, cuja taxa de repetição aumenta à medida que o obstáculo se aproxima do jogador, acelerando o ritmo dos sinais (Figura 45), uma sonificação por frequência de repetição, em que a cadência, mais do que o volume, comunica a proximidade [8]. O sinal é emitido do lado em que o obstáculo se encontra e cessa quando este ultrapassa o jogador. O segundo é a locução direcional: quando a descrição áudio

anuncia a ação a executar, essa locução é posicionada do lado correspondente à ação, reforçando pela posição o que é dito pelo conteúdo.

```
private void AtualizarBeep()
{
    if (!somAtivo || fonteAudio == null || beep == null) return;

    float y = currentObstacle.transform.position.y;

    // já passou o jogador → para de apitar
    if (y <= yParaBeep)
    {
        somAtivo = false;
        LegendasManager.Instance?.Remove<LegendaObstaculo>(legendaObstaculo);
        legendaObstaculo = null;
        return;
    }

    float t = Mathf.Clamp01(Mathf.InverseLerp(spawnY, 0f, y));
    HapticoManager.Instance?.Direcao(ladoObstaculoAtual, t);

    float intervalo = Mathf.Lerp(intervaloLonge, intervaloFerto, t);
    float volume = Mathf.Lerp(volumeMin, volumeMax, t);

    if (dodgeSuccessful)
        volume *= (1f - reducaoPosAcerto);

    volume *= ConfigVolumes.Efeitos;

    timerBeep -= Time.deltaTime;
    if (timerBeep <= 0f)
    {
        fonteAudio.panStereo = panAtual;
        fonteAudio.PlayOneShot(beep, volume);
        timerBeep = intervalo;
    }
}
```

Figura 45 - Cálculo da proximidade do Beep (ObstacleEvent)

### 2.3.13. QuickToggle: Feedback Háptico

O feedback háptico comunica o estado de jogo pelo tato, através dos motores de vibração da consola, constituindo o terceiro canal do princípio de redundância sensorial. É o canal que permite a um jogador com deficiência visual perceber, sem recurso à visão nem necessariamente ao som, qual a ação a executar e qual o desfecho da partida, concretizando o princípio de "substituição da visão pelo tato" [5].

A comunicação entre o jogo e a consola faz-se por porta série. Como a aplicação corre em WebGL e o navegador não permite, por omissão, o acesso direto a portas série, recorre-

se à *Web Serial API*: ao selecionar o controlo, e caso a autorização ainda não exista, o jogo solicita ao jogador o acesso à porta, que este concede manualmente, um requisito de segurança do navegador que obriga a que o pedido parta de uma ação explícita do utilizador [9]. Estabelecida a ligação, o jogo envia à consola pares de valores de intensidade (um por motor, de 0 a 255), que o *firmware* aplica por modulação de largura de pulso (PWM).

A Figura 46 mostra que os padrões de vibração foram desenhados a partir do mapeamento estruturado de eventos a vibrações proposto por Huang et al. no quadro do modelo DPE [5], em que a duração e a cadência transportam significado semântico. Adotaram-se assim: uma vibração breve e ligeira para a navegação de menu (cerca de 150 ms), uma vibração breve, mas mais intensa para a confirmação, e uma vibração contínua mais longa (cerca de 250 ms) para os resultados da partida, dois impulsos para a vitória e um impulso prolongado para a derrota, comunicando o desenlace de forma emocionalmente distinta. Como o *firmware* mantém a intensidade até nova ordem, as durações são geridas do lado do jogo, que envia o valor, aguarda o intervalo e envia a paragem.

```
/// <summary>Navegação de menu: 150 ms ligeiro, ambos os motores.</summary>
public void Navegar() => Pulso(intensidadeNavegar, intensidadeNavegar, msNavegar);

/// <summary>Confirmação: 150 ms mais forte, ambos os motores.</summary>
public void Confirmar() => Pulso(intensidadeConfirmar, intensidadeConfirmar, msConfirmar);

/// <summary>Vitória: 2 pulsos contínuos de 250 ms (resultado positivo).</summary>
public void Vitoria() => PulsoRepetido(intensidadeResultado, intensidadeResultado, msResultado, 2, 120);

/// <summary>Derrota: 1 pulso longo de 250 ms (resultado, emocionalmente ressonante).</summary>
public void Derrota() => Pulso(intensidadeResultado, intensidadeResultado, msResultado);
```

Figura 46 - Padrões semânticos (*HapticoManager*)

A estes padrões acrescenta-se a indicação direcional: dispondo de um motor por lado, o jogo aciona o motor correspondente ao lado da ação a executar, indicando pelo tato se o jogador deve atuar à esquerda ou à direita. No evento de desvio, a intensidade desse motor cresce com a aproximação do obstáculo, à semelhança do alerta sonoro; no evento de curva, um impulso direcional assinala o lado. Esta espacialização tátil torna o canal háptico autossuficiente para a tomada de decisão, e não apenas um reforço dos restantes. Todo o módulo só atua quando ativo e a sua intensidade e durações são parametrizáveis.

## 2.4. Distribuição e Manutenção

O jogo foi compilado no formato WebGL e será distribuído oficialmente através do portal IPCB GAMES. A consola dedicada constitui um periférico opcional que enriquece a experiência através do feedback háptico e dos botões de grande formato.

### 3. Desenvolvimento da Consola

A distribuição do *PowerSensation Games* tira partido das capacidades de exportação multiplataforma do motor Unity, tendo a *build* final sido compilada no formato WebGL. Esta decisão tecnológica é fundamental para a acessibilidade do projeto, pois permite que o jogo seja executado de forma nativa em qualquer navegador *web* moderno, eliminando a necessidade de transferências ou instalações locais por parte dos utilizadores finais. O alojamento e a distribuição oficial do jogo serão realizados através do portal IPCB GAMES.

#### 3.1. Especificação e Fundamentação dos Componentes

##### Arduino Uno R4 WiFi

O Arduino Uno R4 WiFi (Figura 47) representa uma evolução significativa em relação ao modelo anterior R3. Equipado com um microcontrolador Renesas RA4M1 (32-bit ARM Cortex-M4) que opera a 48 MHz, oferece uma capacidade de processamento substancialmente superior, maior memória SRAM (32 kB) e Flash (256 kB). A escolha deste modelo específico justifica-se por dois fatores primordiais.

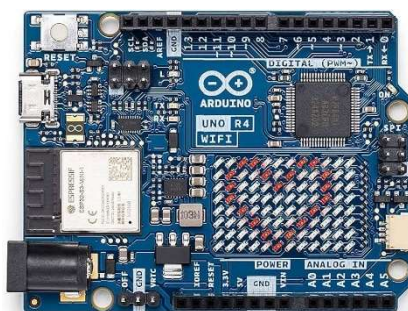


Figura 47 - Arduino Uno R4 Wifi

Conectividade Nativa, permite conectividade *Wi-Fi* e *Bluetooth* integrada. Isto viabiliza a transformação do projeto num dispositivo IoT, permitindo a monitorização de dados em tempo real, armazenamento de pontuações na nuvem ou controlo remoto através de uma interface *web*.

Matriz de LEDs Integrada, adiciona uma camada de feedback visual diretamente na placa de controlo, útil para depuração (*debugging*) ou sinalização de estados do sistema sem necessidade de hardware adicional.

##### Botões Arcade de 60 mm

Estes componentes, representados na Figura 48, constituem a interface principal de entrada do utilizador (*Human-Machine Interface* - HMI). A especificação de botões industriais tipo "Arcade" de grande formato (60 mm) fundamenta-se nos seguintes critérios:

**Ergonomia e Acessibilidade:** O tamanho ampliado facilita o acionamento rápido, minimizando erros e tornando o dispositivo utilizável por crianças, idosos ou pessoas com dificuldades de coordenação motora fina.

**Robustez Mecânica:** Projetados para suportar impactos frequentes e uso intensivo, estes botões possuem *microswitches* internos de alta vida útil mecânica.

**Diferenciação Cromática:** A utilização de quatro cores distintas, permite a programação de dinâmicas visuais complexas (ex: sequências de cores estilo *Simon Says* ou mapeamento de categorias de respostas).



*Figura 48 - Botão Arcade de 60mm*

### **Botão de Pressão de 30 mm**

Este botão, representado na Figura 49, de dimensões reduzidas é estrategicamente incluído para desempenhar funções de controlo de sistema. Ao ser diferenciado fisicamente, evita-se o acionamento accidental por parte do utilizador final. Destina-se a funções de administração, tais como, *Reset* do jogo, início (*Start*), ou calibração de parâmetros.



*Figura 49 - Botão de Pressão de 30 mm*

### **Módulos de Vibração para Arduino**

O feedback háptico é essencial para garantir uma experiência imersiva e acessível. A inclusão de dois atuadores de vibração (Figura 50) permite segmentar os alertas táteis (por exemplo, associando frequências ou padrões de vibração diferentes a botões ou estados distintos). A escolha de Módulos de Vibração em detrimento de motores DC comuns justifica-se porque estes módulos já integram na sua placa (PCB) os componentes de proteção e amplificação necessários (transístor de acionamento, díodo de roda livre e resistências

limitadoras). Isto simplifica a cablagem e elimina o risco de danificar os pinos do microcontrolador por picos de corrente indutiva.



*Figura 50 - Modulo de Vibração para arduino*

### **Cantoneiras em "L" e Conectores Macho/Fêmea**

As cantoneiras em "L" (Figura 51) atuam como elementos de fixação mecânica vitais para a integridade estrutural da caixa/chassis do projeto, garantindo que a estrutura resista às forças mecânicas exercidas pelos utilizadores ao baterem nos botões. Os conectores macho e fêmea (Figura 52) permitem a criação de uma cablagem modular, facilitando a substituição rápida de sensores e a manutenção do hardware sem necessidade de soldaduras permanentes na placa de controlo.



*Figura 51 - Cantoneiras em L*



*Figura 52 - Conectores Machos e Femeas*

## **3.2. Análise Detalhada das Ligações Elétricas e Eletrónicas**

### Configuração dos Circuitos de Entrada (Pinos D2, D3, D4 e D8)

A arquitetura elétrica para a leitura dos quatro botões principais foi desenhada com foco na máxima simplicidade de hardware e imunidade ao ruído elétrico. Cada botão possui dois pinos de ligação:

- Linha de Sinal: Ligada individualmente aos pinos digitais configurados no microcontrolador (D2, D3, D4 e D8).
- Linha de Referência: Todos os segundos terminais dos botões estão interligados em paralelo e conectados ao barramento de GND (Terra) do Arduino.

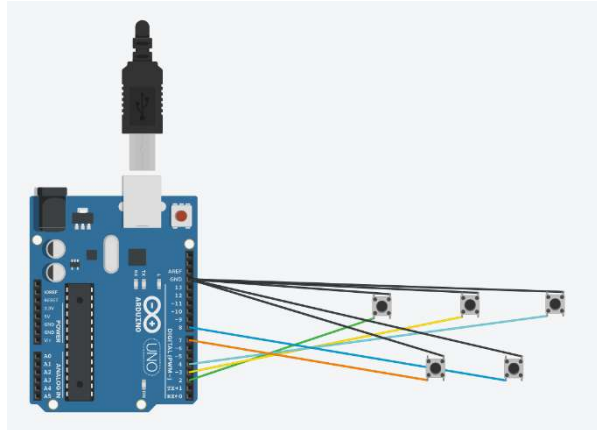


Figura 53 - Esquema de Ligação dos botões ao Arduino

### Justificação do modo INPUT\_PULLUP:

A ausência de resistências físicas externas de *pull-up* ou *pull-down* no esquema elétrico implica a ativação das resistências de pull-up internas do microcontrolador *Renesas* através do *firmware* (`pinMode(pino, INPUT_PULLUP);`). Esta abordagem eletrônica ópera da seguinte forma.

Botão em repouso (Aberto). A resistência interna de *pull-up* liga o pino digital à linha de 5V interna. O microcontrolador lê um estado lógico ALTO (*HIGH* / 1) estável, prevenindo flutuações eletrostáticas (pino flutuante).

Botão pressionado (Fechado), o circuito fecha-se diretamente para o GND. O pino digital é forçado a 0V, e o microcontrolador regista um estado lógico BAIXO (*LOW* / 0)

### Configuração dos Circuitos de Saída Háptica (Pinos D5, D6 e D7)

Os dois módulos de vibração são controlados de forma independente através dos pinos digitais D5, D6 e D7. Cada módulo possui uma interface de três pinos como vemos na imagem Figura 54:

- VCC: Ligado ao barramento de alimentação de 5V regulado do Arduino.
- GND: Ligado ao barramento de GND comum do sistema.
- IN (Sinal/Controlo): Ligado ao respetivo pino digital do Arduino.

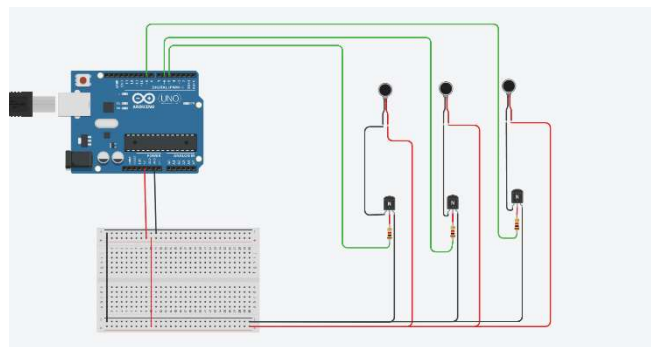


Figura 54 - Esquema de Ligação dos módulos de vibração

Como os pinos escolhidos no Arduino Uno suportam modulação por largura de pulso (PWM - *Pulse Width Modulation*), o sistema ganha a capacidade técnica de controlar não apenas o estado ligado/desligado do motor, mas também a intensidade da vibração. Variando o *duty cycle* do sinal PWM, varia-se a tensão média aplicada ao pequeno motor excêntrico do módulo, modulando a força do *feedback* háptico consoante a gravidade ou tipo do evento mecânico programado.

### Integração Hardware–Software

A integração entre a consola e o jogo assenta na tradução das ações físicas em comandos de jogo e dos eventos de jogo em *feedback* háptico. Os botões da consola apresentam-se ao Unity como entradas de teclado, sendo mapeados às ações do jogo através do novo *Input System*; o canal háptico é acionado pelo envio, do jogo para a consola, do padrão de vibração correspondente a cada evento.

### 3.3 Implementação e Lógica do Código Arduino

O *software* desenvolvido para o microcontrolador foi estruturado para garantir escalabilidade, baixa latência e a execução simultânea de múltiplas tarefas (leitura de botões e receção de comandos de vibração).

A lógica do código divide-se em quatro componentes fundamentais:

#### Estrutura de Dados e Abstração de Hardware

Em vez de declarar variáveis individuais para cada pino físico, o sistema utiliza *arrays* (vetores) sincronizados por índices. Um índice (de 0 a 4) associa automaticamente o pino físico do botão, o pino do respetivo LED e o *scan code* da tecla *Numpad* correspondente. Para gerir componentes assimétricos, como o botão de sistema (*microswitch*) que não possui

iluminação integrada, foi implementado um marcador lógico (`const uint8_t SEM_LUZ = 255;`). O código verifica este marcador antes de enviar sinais de saída, evitando a ativação indevida de pinos não atribuídos (Figura 55).

```
// =====  
// 1. CONFIGURAÇÃO DOS BOTÕES E LUZES  
// =====  
const uint8_t numBotoes = 5;  
const uint8_t SEM_LUZ = 255; // Marcador para botões sem luz (como o microswitch verde)  
  
// A ordem dita os pares!  
// Índice:           0 (Azul) | 1 (Verde) | 2 (Azul) | 3 (Verde) | 4 (Verde simples)  
const uint8_t buttonPins[] = {4,      2,      6,      8,      10};  
const uint8_t ledPins[]    = {5,      3,      7,      9,      SEM_LUZ};  
  
// Códigos do Teclado Numérico (Numpad)  
// Numpad 2 (226) | Numpad 4 (228) | Numpad 8 (232) | Numpad 6 (230) | Numpad 5 (229)  
const uint8_t numpadKeys[] = {226,    228,    232,    230,    229};
```

Figura 55 - Inicialização dos arrays sincronizados e do marcador de ausência de LED

### Leitura de Entradas e Filtragem (Debounce Não-Bloqueante)

A fiabilidade da leitura dos botões é assegurada por um algoritmo de *debounce* por *software*. Devido à natureza eletromecânica dos botões, o encosto das placas metálicas gera "ruído" elétrico (múltiplas flutuações rápidas entre 0V e 5V) que o microcontrolador leria como dezenas de cliques acidentais num único segundo. Para filtrar este ruído, o código regista o tempo exato (*millis()*) em que o estado elétrico do pino sofre alteração. O sinal só é validado como um clique real se permanecer estável durante uma janela de 50 milissegundos. Esta abordagem utiliza os temporizadores internos do microcontrolador em vez de funções de pausa (*delay()*), garantindo que o ciclo principal (*loop*) nunca é bloqueado (Figura 56).

```

void loop() {
    unsigned long now = millis();

    // =====
    // BLOCO A: LER E ENVIAR CLIQUES
    // =====
    for (uint8_t i = 0; i < numBotoes; i++) {
        bool reading = digitalRead(buttonPins[i]);

        if (reading != lastReading[i]) {
            lastDebounceTime[i] = now;
        }

        if ((now - lastDebounceTime[i]) > debounceDelay) {
            if (reading != buttonState[i]) {
                bool prev = buttonState[i];
                buttonState[i] = reading;

                if (buttonState[i] == LOW && prev == HIGH) {
                    handlePress(i);
                }
                else if (buttonState[i] == HIGH && prev == LOW) {
                    handleRelease(i);
                }
            }
        }
        lastReading[i] = reading;
    }
}

```

Figura 56 - Implementação do algoritmo de debounce não-bloqueante, recorrendo à função `millis()` para filtragem de ruído eletromecânico.

## Emulação de Teclado (USB HID)

Uma vez validada a pressão ou libertação de um botão, o sistema invoca as funções de abstração `handlePress()` e `handleRelease()`. Estas funções executam duas ações simultâneas:

Ativam ou desativam o sinal de tensão (*HIGH/LOW*) para o pino do LED associado, fornecendo *feedback* visual imediato na consola.

Enviam comandos através da biblioteca `Keyboard.h`, instruindo o controlador USB nativo do Arduino a enviar o scancode exato de uma tecla (ex: *Numpad 2*, *Numpad 4*). Para o motor de jogo Unity, o sinal é indistinguível do de um teclado informático convencional.

```

// =====
// FUNÇÕES AUXILIARES
// =====

void handlePress(uint8_t index) {
    // Acende a luz (se existir) e carrega na tecla
    if (ledPins[index] != SEM_LUZ) {
        digitalWrite(ledPins[index], HIGH);
    }
    Keyboard.press(numpadKeys[index]);
}

void handleRelease(uint8_t index) {
    // Apaga a luz e larga a tecla
    if (ledPins[index] != SEM_LUZ) {
        digitalWrite(ledPins[index], LOW);
    }
    Keyboard.release(numpadKeys[index]);
}

```

Figura 57 - Funções de abstração responsáveis pela emulação USB HID (Keyboard.h) e pela atuação simultânea do feedback visual nos LEDs.

## Receção de Telemetria e Modulação Háptica

Para receber as ordens de vibração geradas pelo jogo, o código implementa uma leitura Serial assíncrona orientada a *buffers*. A cada ciclo do programa, o microcontrolador verifica se existem novos caracteres na porta USB e guarda-os sequencialmente num *array* de memória de 32 bytes. Quando o caractere de quebra de linha (`\n`) é detetado, a mensagem (ex: "255, 128") é dada como completa e processada pela função *parseAndApplyPWM()*. Esta rotina utiliza as funções da linguagem C (*strtok* e *atoi*) para separar a mensagem nas vírgulas e converter o texto em valores numéricos inteiros. Por fim, a função de segurança *constrain()* garante que os valores não excedem os limites do *hardware* (0 a 255) antes de os aplicar através de *analogWrite()* aos pinos dos motores. Esta função gera um sinal PWM (*Pulse Width Modulation*) que regula com precisão a força física da vibração sentida pelo jogador.

```

// =====
// BLOCO B: OUVIR O UNITY (Vibrações)
// =====
while (Serial.available() > 0) {
    char c = Serial.read();
    if (c == '\n') {
        bufReady = true;
        break;
    }
    if (bufIdx < bufSize - 1) {
        inputBuffer[bufIdx++] = c;
    }
}

if (bufReady) {
    inputBuffer[bufIdx] = '\0';
    parseAndApplyPWM(inputBuffer);
    bufIdx = 0;
    bufReady = false;
}
}

```

Figura 58 - Rotinas de telemetria: recepção assíncrona orientada a buffers

```

void parseAndApplyPWM(char* buf) {
    // Separa o texto (Ex: "255, 128") em dois valores inteiros
    char* p = strtok(buf, ", ");
    if (!p) return;
    int v1 = atoi(p);

    p = strtok(NULL, ", ");
    if (!p) return;
    int v2 = atoi(p);

    // Aplica a força (0 a 255) aos pinos dos motores
    analogWrite(motorPins[0], constrain(v1, 0, 255));
    analogWrite(motorPins[1], constrain(v2, 0, 255));
}

```

Figura 59 - Função de decodificação com modulação segura de sinal PWM

### 3.4. Montagem

A fase de montagem física da consola exigiu uma adaptação estratégica face ao planeamento inicial. Embora o design original prevísse o fabrico de um chassis totalmente personalizado através de impressão 3D, esta abordagem revelou-se inviável para a construção deste primeiro protótipo funcional devido a três fatores constrangedores:

- **Limitações de Hardware de Impressão:** O volume de impressão das máquinas disponíveis era inferior às dimensões totais projetadas para a nova consola, o que obrigaria a imprimir o chassis em várias peças pequenas, comprometendo a sua integridade estrutural.
- **Fator Tempo:** A impressão 3D de peças de grande volume exige dezenas de horas de operação, o que iria atrasar a fase de testes e validação do software.
- **Custos Associados:** O elevado consumo de filamento para uma estrutura desta escala tornava a produção do protótipo desnecessariamente dispendiosa nesta fase de desenvolvimento.

## A Solução de Prototipagem

Como alternativa sustentável e eficiente, optou-se por uma abordagem de reciclagem de hardware (*upcycling*). Foram reaproveitadas e unidas fisicamente as estruturas em MDF de duas consolas antigas (modelo +Power).

Esta junção resultou num chassis suficientemente amplo, robusto e perfeitamente capaz de acomodar os cinco novos botões *Arcade*, o Arduino Uno R4 e as baterias. Esta decisão revelou-se um sucesso, pois permitiu validar em ambiente real toda a eletrónica, o código e a ergonomia do sistema sem atrasar o calendário do projeto, reservando a produção da estrutura em impressão 3D (detalhada no Capítulo 2) para a versão final de produção.

## 4. Conclusão

### 4.1. Reflexão sobre o Trabalho Desenvolvido

Este relatório descreveu a fase de produção do PowerSensation Games, um videojogo desportivo inclusivo que concretiza, em software, o princípio da redundância sensorial fundamentado na revisão sistemática do Projeto I. O trabalho desenvolvido permitiu transformar esse princípio, até então sustentado apenas em literatura, num sistema funcional e jogável.

Foi implementada a modalidade de Ciclismo sobre uma arquitetura escalável, concebida desde o início para acomodar novas modalidades sem reescrita do núcleo do jogo. Sobre essa base, desenvolveram-se os mecanismos de acessibilidade nos três canais: visual, sonoro e háptico; ativáveis em tempo real através dos Quick Toggles e configuráveis em detalhe no menu de Opções. Concretizaram-se, em particular, a descrição áudio por síntese de voz, as legendas com descrição não-verbal de sons segundo a tipologia NV-SDH, o áudio espacial direcional e o feedback háptico segmentado por lado, cada um fundamentado nas evidências reunidas em Projeto I [1].

A separação entre ação lógica e dispositivo físico, materializada nos perfis de input, revelou-se uma decisão estruturante: é ela que permite que o mesmo jogo seja operável por teclado e pelas consolas adaptadas sem versões distintas, e que as indicações ao jogador correspondam sempre ao dispositivo em uso. Em paralelo, reformulou-se a consola física, sucessora das +Power e +PowerSensations, com botões de grande formato e feedback háptico, comunicando com o jogo através da porta série.

O maior desafio desta fase residiu, precisamente, em garantir que cada informação de jogo permanecesse acessível em simultâneo por todos os canais, sem que a ativação ou desativação de um módulo compromettesse a legibilidade dos restantes. O resultado é um sistema em que o estado de jogo é comunicado de forma redundante e modular, aproximando-se do objetivo de tornar a mesma experiência acessível a jogadores com diferentes perfis sensoriais.

### 4.2. Trabalho Futuro

Como trabalho futuro, perspetiva-se, em primeiro lugar, o alargamento do jogo a novas modalidades, tirando partido da arquitetura escalável já validada. A modalidade de Voleibol, prevista desde o início do projeto, encontra-se desenhada, mas ainda por implementar, e constitui o passo natural seguinte: assenta numa mecânica de decisão direcional distinta da do Ciclismo, permitindo exercitar a arquitetura de acessibilidade num segundo contexto de jogo e confirmar a sua reutilização.

No plano da acessibilidade, identificam-se dois módulos visuais ainda por concretizar. O alto contraste, cujo estado já é guardado pelo sistema, requer a sua aplicação efetiva à interface e aos elementos de jogo, previsivelmente através da substituição de paletas de cor

que reforcem a distinção entre elementos. O filtro de daltonismo, para o qual o jogo já guarda o tipo selecionado (protanopia, deuteranopia ou tritanopia), carece da aplicação visual correspondente, que ajuste as cores apresentadas ao tipo de daltonismo do jogador.

## 5. Bibliografia

- [1] D. Santos e S. Cardoso, «PowerSensation Games: Experiências digitais para todos», Trabalho de Projeto, Escola Superior de Tecnologia, Instituto Politécnico de Castelo Branco, 2026.
- [2] maispower, «+Power: Recursos Educativos para TODOS». Acedido: 25 de Junho de 2026. [Em linha]. Disponível em: <https://portal.maispower.pt/>
- [3] N. Trick, C. Phillips, C. Gutwin, e L. Bradford, «Content Analysis of Basic Accessibility Features in 30 Top Played Video Games», *Int. J. Hum. Comput. Interact.*, vol. 42, n. 1, pp. 425–440, 2026, doi: 10.1080/10447318.2025.2508307.
- [4] K. K. G, S. D. V. Chundru, e M. R. Chintapalli, «BlindBattle: Tactile Game For Blind», em *Proc. 2024 IEEE Int. Conf. on Communication, Computing and Signal Processing (IICCCS)*, 2024. doi: 10.1109/IICCCS61609.2024.10763650.
- [5] X. Huang, N. M. Ali, S. Sahrani, e Y. Zhang, «A Haptic-Driven Serious Game for Cognitive Stimulation and Visual Impairment Mitigation in Older Adults Based on the Design-Play-Experience Framework: Cross-Sectional Mixed Methods Pilot Study», *JMIR Serious Games*, vol. 14, p. e86290, 2026, doi: 10.2196/86290.
- [6] N. Ben Itzhak, I. Franki, B. Jansen, K. Kostkova, J. Wagemans, e E. Ortibus, «Usability and user experience of an individualized and adaptive game-based therapy for children with cerebral visual impairment», *Int. J. Child. Comput. Interact.*, vol. 35, p. 100551, 2023, doi: 10.1016/j.ijcci.2022.100551.
- [7] M. Al-Batineh, «Audio accessibility in Arabic game localization: A multimodal study of nonverbal subtitles for players who are d/Deaf and hard of hearing», *Translation Spaces*, vol. 14, n. 2, pp. 277–302, 2025, doi: 10.1075/ts.24042.alb.
- [8] R. Savaglio, J. M. Brown, B. Kapralos, B. Franco-Arellano, A. LeSage, e J. Arcand, «An Accessible Version of the Foodbot Factory Serious Game for Nutrition Education», *IEEE Trans. Games*, vol. 17, n. 1, pp. 181–194, 2025, doi: 10.1109/TG.2024.3414657.
- [9] mozilla.org, «Web Serial API». Acedido: 25 de Junho de 2026. [Em linha]. Disponível em: [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Serial\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Serial_API)