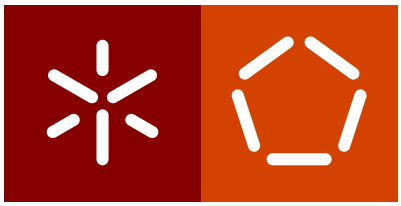
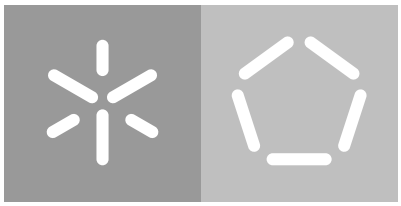


Universidade do Minho
Escola de Engenharia
Departamento de Informática

José Carlos Silva Brandão Gonçalves

Design e Desenvolvimento de um Simulador de Ambientes para PVSio-web

Dezembro 2018



Universidade do Minho
Escola de Engenharia
Departamento de Informática

José Carlos Silva Brandão Gonçalves

Design e Desenvolvimento de um Simulador de Ambientes para PVSio-web

Dissertação de Mestrado
Mestrado em Engenharia Informática

Dissertação supervisionada por
Professor Doutor José Francisco Creissac Freitas de Campos
Doutor Paolo Masci

Dezembro 2018

AGRADECIMENTOS

Em primeiro lugar, quero agradecer intimamente à minha família, em particular aos meus pais, Abel e Alice, e aos meus irmãos, João e Jorge, por toda a ajuda, dedicação e amor que sempre me ajudou a ultrapassar todos os obstáculos que surgissem ao longo de toda a vida. Gostaria também de agradecer todo o apoio que os meus amigos me deram ao longo deste último ano.

Ao meu orientador, Professor Doutor José Francisco Creissac Freitas de Campos, e ao meu co-orientador, Doutor Paolo Masci, investigador Pós-Doutoramento, um muito obrigado pela paciência e orientação nos trabalhos que realizamos nestes últimos dois anos. Obrigado pelas correções, pelas critícas construtivas, e pela disponibilidade em encontrarmos-nos sempre que necessitei de auxílio. E, para terminar, obrigado pelas valiosas contribuições para o trabalho desenvolvido nesta dissertação e, principalmente, obrigado por estimularem o meu conhecimento académico.

Gostaria também de deixar o meu agradecimento ao Departamento de Informática da Universidade do Minho e ao seu corpo docente por todo o conhecimento de excelência que me transmitiram ao longo da minha Licenciatura e Mestrado em Engenharia Informática.

Agradeço também a todos os meus colegas do Mestrado em Engenharia Informática, especialmente a João Bernardo, que disponibilizou um controlador externo mais sofisticado, que permitiu uma melhor realização de testes ao trabalho desenvolvido nesta dissertação.

ABSTRACT

A library for the construction of a race simulation environment, combining a track generator and a rendering engine, was developed and is described in this dissertation. The library was built on simple, but realistic rendering techniques used in old arcade games. This library also allows to visualize more realistic renderings, than the graphics aforementioned if the user wishes. The focus will be on the automotive domain. Virtual prototypes were developed that can be used to analyze semi-autonomous functions of a car that involve *HMI (Human-Machine Interfaces)* and depend on aspects of the wider system, such as road topography. A case study based on a dangerous design flaw of the 2015 Lincoln MKC automotive dashboard will be used throughout the dissertation to test the flexibility and capabilities of the library.

Human error is defined as the main cause of vehicle accidents, however it is important to highlight that incident investigations sometimes also reveal that some human errors are due to latent design flaws, rather than careless behavior. The design of interfaces in the automotive domain is of utmost importance, as any flaw could lead to use errors that have safety consequences. Interactive simulations of dashboard designs in conjunction with driving environments allow rapid simulation of different situations and facilitate the process of analysis of these designs.

This dissertation describes how to use a prototype and analysis toolkit to support the development of virtual prototypes that compose an interactive simulation with an automotive dashboard and its contexts of use, called driving environments. It also explored the integration of external controllers, such as gamepads, steering wheels, pedals (used in game consoles), virtual gamepads, virtual arrow keyboards and mobile devices with gyroscope sensor to facilitate simulation.

Keywords: Driving Environments, Rapid Prototyping, Virtual Prototyping, Interactive Simulation, Arcade and Realistic Driving Simulation, External Controllers, Design Flaws

RESUMO

Uma biblioteca para a construção de um ambiente de simulação de corrida, combinando um gerador de pistas e um motor de renderização, foi desenvolvida e é descrita nesta dissertação. A biblioteca foi construída sobre técnicas de renderização simples, mas realistas, usadas em antigos jogos de *arcade*. Esta biblioteca também permite visualizar renderizações mais realistas, para além dos gráficos acima supracitados, caso o utilizador desejar. O foco será no domínio automóvel.

Foram desenvolvidos protótipos virtuais que podem ser usados para analisar funções semi-autónomas de um veículo, que envolvem *HMI (Human-Machine Interface)* e que dependem de aspectos mais amplos do sistema, como a topografia da estrada. Um caso de estudo, baseado numa falha de *design* perigosa do painel de instrumentos do veículo Lincoln MKC de 2015, será usado em toda a dissertação para testar a flexibilidade e as capacidades da biblioteca.

O erro humano é definido como a principal causa de acidentes que envolvem veículos, no entanto, é importante ressaltar que existem investigações após-incidente, que às vezes também revelam que alguns erros humanos são causados por falhas latentes de *design*, e não devido a comportamentos descuidados dos condutores. O *design* de interfaces no domínio automóvel é de extrema importância, pois qualquer falha pode levar a erros de utilização, com consequências negativas, relacionadas com a segurança dos seus ocupantes.

Simulações interactivas de *designs* de painéis de instrumentos em conjunto com os ambientes de condução permitem a simulação rápida de diferentes situações e facilitar o processo de análise desses *designs*.

Esta dissertação descreve como utilizar um *toolkit* de prototipagem e de análise para apoiar o desenvolvimento de protótipos virtuais que compõem uma simulação interativa com um painel de instrumentos e com os seus contextos de utilização, designados por ambientes de condução. Foi também explorada a integração de controladores externos, como *gamepads*, volantes, pedais (utilizados em consolas de jogos), *gamepads* virtuais, teclados de setas virtuais e controladores com sensor giroscópio, para facilitar a simulação.

Keywords: Ambientes de Condução, Prototipagem Rápida, Prototipagem Virtual, Simulações Interactivas, Simulador de Condução *Arcade* e Realista, Controladores Externos, Falhas de *Design*

ÍNDICE

LISTA DE FIGURAS

EXCERTOS DE CÓDIGO

LISTA DE ACRÓNIMOS E SIGLAS

A

APEX Agile Prototyping for user EXperience.

API Application Programming Interface.

APIS Application Programming Interfaces.

C

CEO Chief Executive Officer.

D

DI Departamento de Informática.

DOM Document Object Model.

G

GPS Global Positioning System.

H

HCI Human-Computer Interactions.

HMIS Human-Machine Interfaces.

HMI Human-Machine Interface.

M

MIEI Mestrado Integrado em Engenharia Informática.

MIT Massachusetts Institute of Technology.

S

SVG Scalable Vector Graphics.

U

UI User Interface.

UM Universidade do Minho.

W

WRC World Rally Championship.

1

INTRODUÇÃO

Esta dissertação descreve a implementação do projeto de Mestrado desenvolvido no contexto de *Mestrado Integrado em Engenharia Informática (MIEI)*, realizada no *Departamento de Informática (DI)*, *Universidade do Minho (UM)*. Motivação, contextualização, objetivos, contribuições e estrutura da dissertação serão discutidos nas seções vindouras.

Esta dissertação diz respeito à prototipagem de Interfaces Pessoa Máquina (do inglês *Human-Machine Interfaces (HMIs)*) no contexto automóvel e ao impacto que determinados contextos de utilização têm nesse processo. Uma biblioteca para a construção de simulações de pistas e dos seus ambientes de condução foi desenvolvida.

1.1 MOTIVAÇÃO

Nesta secção são apresentadas algumas classes de problemas para ajudar a demonstrar a necessidade de se considerarem os contextos de utilização no desenvolvimento de interfaces[1]. Para as classes de problemas identificadas indicam-se exemplos de veículos, onde é possível observar algumas falhas de *design* nas respectivas HMIs.

Um caso de estudo, com base na ativação do modo desportivo, no painel de instrumentos do veículo *Lincoln MKC* de 2015, será também apresentado. Este caso de estudo permitirá demonstrar como a solução proposta permite prototipar a sua interface com os ambientes de condução criados conforme a parametrização fornecida e que se focam no acto de conduzir.

As quatro classes de problemas identificadas apresentam-se de seguida.

1.1.1 *Problemas de Feedback*

Esta classe de problemas diz respeito aos problemas relacionados com a falta de *feedback* ou com *feedback* fornecido que não é imediatamente perceptível. Estes problemas verificam-se com a substituição de botões físicos por botões tácteis, que não fornecem um bom *feedback*

1 `https://jalopnik.com/the-ten-worst-automotive-design-flaws-1743090273` (*The Ten Worst Automotive Design Flaws, A. Brown, November 2015*)

Figura 1.: **Interface de Controlo Áudio no Painel de Instrumentos do Veículo *Cadillac XT5* de 2017** (As setas verde e vermelho representam, respectivamente, os botões físicos e tácteis que controlam o volume.)

ao condutor, uma vez que ao tocar neste tipo de interfaces, sem olhar diretamente, o condutor não percebe, imediatamente, se tocou no botão táctil certo. Os veículos *Cadillac XT5* de 2017 e *Tesla Model 3* de 2017 apresentam problemas nas interfaces dos seus painéis de instrumentos. O problema na interface do veículo *Cadillac XT5* de 2017 é a ausência de botão físico de volume, na consola central do painel de instrumentos, que pode ser observada na Figura 1, página 2. O condutor tem de tocar repetidamente num botão táctil, após a inicialização do sistema áudio, para controlar o respectivo volume. Neste caso, só quando o volume de áudio aumentar ou diminuir a quantidade necessária para que o condutor verifique a alteração do volume é que este perceberá se tocou efetivamente no botão certo. Contudo, esta falha de *design* é menos séria visto que o condutor possui no volante dois botões físicos que permitem também controlar o volume do áudio.

Os problemas na interface do veículo *Tesla Model 3* de 2017 são a ativação ou desativação das escovas, bem como a definição da velocidade com que as escovas funcionam[2], Figura 2, página 3, e a ativação e configuração do ar condicionado, Figura 3, página 4. Estas duas tarefas apenas podem ser concretizadas através da única interface disponibilizada, que é o ecrã táctil localizado na parte central do seu painel de instrumentos. Em relação ao Tesla, a interface que permite regularizar a temperatura interior possui outro problema

2 `https://www.carscoops.com/2017/09/youll-have-to-use-touchscreen-to-adjust/` (*You'll Have To Use The Touchscreen To Adjust Tesla Model 3's Wipers, C. Gnaticov, September 26, 2017*)

Figura 2.: **Interface de Ativação das Escovas do Pára-Brisas no Painel de Instrumentos do Veículo *Tesla Model 3* de 2017** (A seta vermelha representa o botão táctil que liga as escovas do pára-brisas.)

que advém do tamanho, pequeno, definido para as setas que permitem aumentar e/ou diminuir a temperatura. Devido ao pequeno tamanho atribuído a essas setas, o mínimo deslize poderá diminuir a temperatura em vez de aumentar, o que dificulta ainda mais a realização da tarefa.

As interfaces tácteis não são tão intuitivas para os condutores determinarem se ativaram ou se clicaram no botão e, como tal, muitas vezes, torna-se necessário desviar o olhar da estrada para ter a certeza que se ativa ou clica no botão correcto, podendo resultar em graves acidentes. Por exemplo, ligar as escovas do pára-brisas numa curva perigosa e estreita terá de ser realizada sem desviar o olhar da estrada, caso contrário o condutor pode também correr o risco de se despistar devido à falta de visibilidade promovida pela acumulação de gotas de água no pára-brisas.

1.1.2 *Problemas de Incoerência*

Esta classe de problemas diz respeito aos problemas relacionados com a existência de diferentes mecanismos de controlo inconsistentes para a realização de uma mesma tarefa. O veículo *Nissan Leaf* de 2017 possui um problema na interface do seu painel de instrumentos relacionado com o controlo do volume do sistema de áudio do veículo. A interface que permite controlar o volume de áudio do sistema de som desse veículo pode ser observada na Figura 4, página 5. A incoerência na interface do painel de instrumentos verifica-se nas duas interfaces para controlar o volume do áudio disponíveis. Uma interface permite ajustar o volume utilizando os botões com as setas para cima e/ou para baixo, na consola central, para aumentar e/ou diminuir o volume. A outra interface permite ajustar o volume

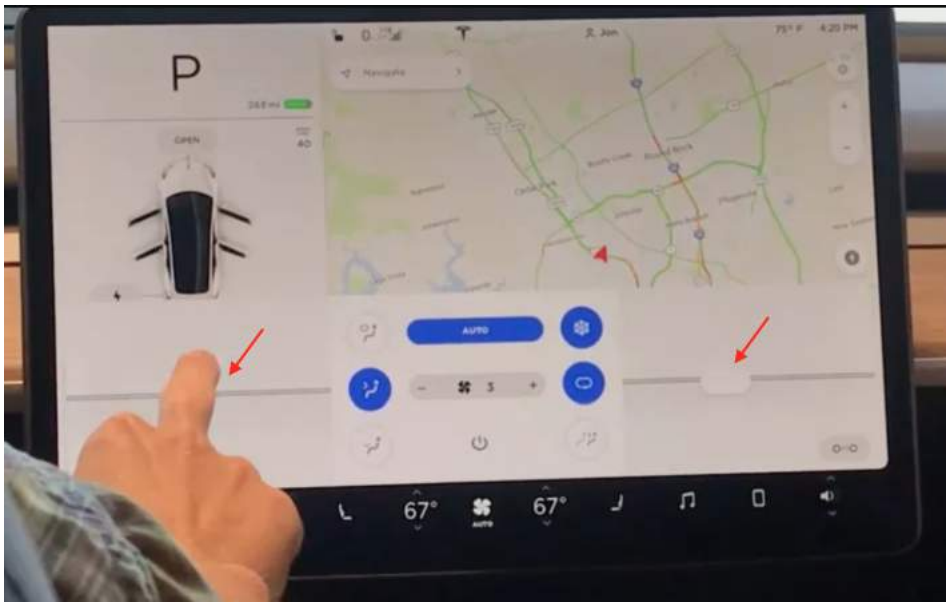


Figura 3.: **Interface de Ativação do Ar Condicionado no Painel de Instrumentos do Veículo *Tesla Model 3* de 2017** (As setas vermelhas representam os botões tácteis que regulam a temperatura no interior do veículo.)

do áudio utilizando os botões com as setas a apontar para o lado direito e/ou para o lado esquerdo no volante. Inevitavelmente, ocorrem situações em que o condutor vai clicar nos botões com as setas para cima e/ou para baixo no volante, e essa ação não vai resultar nem no aumento nem na diminuição do volume do áudio. E poderá ser, potencialmente, perigoso, dependendo da funcionalidade que esses botões do volante têm.

1.1.3 *Problemas de Exagerada Complexidade*

Esta classe de problemas diz respeito aos problemas relacionados com a exagerada complexidade envolvida na realização de tarefas nas interfaces de painéis de instrumentos. O veículo *Subaru Outback* de 2018 possui um problema na interface do seu painel de instrumentos, Figura 5, página 7, relacionado com o ajuste manual do relógio. O ajuste manual do relógio implica a realização de um conjunto de 19 passos sequenciais no volante, no ecrã táctil e nos botões ao lado esquerdo do ecrã táctil, ambos na consola central do painel de instrumentos. De seguida enumeram-se as 19 etapas necessárias para alterar as horas apresentadas no relógio deste veículo,

1. Clicar no botão físico *Home* (seta vermelha na Figura 5).
2. Clicar no botão táctil *Settings*, que é o botão mais à direita do conjunto de botões apresentado, na parte inferior do ecrã táctil (seta azul na Figura 5).

Figura 4.: **Interface de Controlo Áudio no Painel de Instrumentos do Veículo *Nissan Leaf* de 2017** (As setas vermelhas representam, respectivamente, os botões físicos e tácteis que controlam o volume.)

3. Clicar na opção *Vehicle*, que aparecerá no ecrã táctil.
4. Clicar na opção *Clock Adjustment*, que aparecerá no ecrã táctil.
5. Clicar na opção *Manual*, que aparecerá no ecrã táctil. A opção *Auto* é a outra possibilidade, contudo a vasta maioria dos condutores queixam-se que as horas definidas por esta opção, que recorre a um satélite GPS, nunca são as horas corretas[3].
6. Clicar no botão físico *Home*.
7. Rodar o volante até os botões com as setas para cima e/ou para baixo e o botão *i/set*(setas amarelas e verde, respectivamente, na Figura 5) estarem na parte superior (ao nível dos medidores de velocidade e de conta-rotações e do pequeno ecrã LED (seta azul claro na Figura 5) que se encontra entre esses medidores).
8. Clicar num dos botões com as setas para cima e/ou para baixo até aparecer a opção *Pull and hold i/set switch for menu* no ecrã LED que se encontra entre os medidores de velocidade e conta-rotações.
9. Pressionar o botão *i/set* durante cerca de 4 segundos, que deverá apresentar no ecrã LED a opção *Clock*.
10. Clicar no botão *i/set* para selecionar a opção *Clock* que aparece no ecrã LED após a execução do passo anterior.

3 `https://www.youtube.com/watch?v=aFJfSe4SLaQ/` (*How to: Change the Clock in a 2018 Subaru Outback, Groove Subaru, December 22, 2017*)

11. Clicar nos botões com as setas para cima e/ou para baixo para escolher a nova hora do relógio.
12. Clicar no botão *i/set* para definir nova hora escolhida no passo anterior e permitir a mudança dos minutos a realizar no próximo passo.
13. Clicar nos botões com as setas para cima e/ou para baixo para escolher o novo valor para os minutos do relógio.
14. Clicar no botão *i/set* para definir o novo valor para os minutos escolhido no passo anterior.
15. Clicar nos botões com as setas para cima e/ou para baixo até aparecer no ecrã LED a opção *Go back*.
16. Clicar no botão *i/set* para selecionar a opção *Go back*, que permitirá sair do menu de definição dos novos valores para a hora e para os minutos.
17. Clicar nos botões com as setas para cima e/ou para baixo até aparecer no ecrã LED a opção *Go back*.
18. Clicar no botão *i/set* para selecionar a opção *Go back*, que permitirá sair do menu *Clock*.
19. Clicar nos botões com as setas para cima e/ou para baixo até aparecer no ecrã LED a opção referente ao menu que esse ecrã apresentava antes da modificação da hora, que normalmente é a opção que apresenta os consumos atuais do veículo e a previsão da quilometragem restante com o volume de combustível do depósito atual.

Após a enumeração das 19 etapas supracitadas é possível concluir que esta interface implica a realização de um conjunto de etapas confusas e morosas para uma tarefa simples.

1.1.4 *Problemas de Localização Inadequada*

Esta classe de problemas diz respeito aos problemas relacionados com a localização inadequada de botões que possuem funcionalidades relevantes para o processo de condução. O caso estudo utilizado nesta dissertação diz respeito à localização do botão responsável pela ativação do modo desportivo (*S*) e do botão responsável por ligar e desligar (*Start/Stop*) o veículo *Lincoln MKC* de 2015.

O fabricante deste veículo decidiu colocar o botão *Start/Stop* imediatamente a baixo do botão *S*. O modo desportivo é uma funcionalidade que permite ter um desempenho melhor na estrada no que diz respeito à rapidez de aceleração. A qualidade do *design* foi colocada em causa por vários proprietários que alertaram para o perigo de o condutor a desligar acidentalmente o veículo em movimento, devido à proximidade desses dois botões. O

Figura 5.: **Painel de Instrumentos Completo do Veículo *Subaru Outback* de 2018** (A seta vermelha indica o botão físico *Home*. A seta azul indica o ecrã táctil na consola central. As setas amarelas indicam os botões físicos com as setas para cima e/ou para baixo. A seta verde indica o botão *i/set*. A seta azul claro indica o pequeno ecrã LED entre o velocímetro e o tacómetro.)

clique no botão *Start/Stop* em vez do clique no botão *S* implica que o veículo é desligado a uma velocidade considerável que dificulta o seu controlo e que, com elevada probabilidade, termina com a perda de controlo do mesmo, que poderá resultar em graves acidentes.

Para além do problema da localização dos botões *Start/Stop* e *S*, existe outro problema reportado por um condutor aos órgãos reguladores de segurança federais, dos Estados Unidos da América, acerca da proximidade da localização do botão *Start/Stop* em relação ao ecrã táctil, que se encontra também na consola central do painel de instrumentos deste veículo. O ecrã táctil possui várias funcionalidades como o sistema de navegação e o sistema de som (áudio via *bluetooth* ou rádio). Esse condutor reportou o caso em que o passageiro estava a procurar uma estação de rádio, quando equivocadamente clicou o botão *Start/Stop*, o que provocou a paragem imediata e subita do veículo, como se o condutor tivesse pressionado o pedal de travagem até ao fundo.

O número de queixas foi elevado de tal forma que obrigou o fabricante a reconhecer[4] a falha de *design*, ao comprovar a existência de acidentes em que vários condutores tocaram equivocadamente no botão *Start/Stop*, enquanto o carro estava em movimento, e a dar uma ordem de recolha [15] de todos os veículos afetados para proceder à correção imediata da interface. A Figura 6, página 8, mostra o painel de instrumentos completo deste veículo antes da recolha dos veículos para correção da respectiva interface. A Figura 7, página 8, mostra o painel de instrumentos completo deste veículo depois da recolha mencionada

4 `http://money.cnn.com/2015/01/06/autos/ford-push-button-ignition-recall/index.html` (*Ford recalls SUVs because drivers are accidentally turning them off, C. Isidore, January, 2015*)

Figura 6.: **Painel de Instrumentos Completo do Veículo *Lincoln MKC* de 2015 - Antes da Recolha do Fabricante** (A seta verde indica o botão *S*. A seta vermelha indica o botão *Start/Stop*.)

Figura 7.: **Painel de Instrumentos Completo do Veículo *Lincoln MKC* de 2015 - Depois da Recolha do Fabricante** (A seta verde indica o botão *S*. A seta vermelha indica o botão *Start/Stop*.)

acima. A correção passou pela alteração da localização do botão *Start/Stop*, que foi movido para o topo do *cluster* de botões na consola central (seta vermelha na Figura 7). Na interface corrigida o botão *S*, ficou na última posição dentro do *cluster* de botões na consola central (seta verde na Figura 7).

1.2 CONTEXTUALIZAÇÃO

Nesta secção é apresentado o problema que a solução proposta nesta dissertação pretende resolver. O problema é facilitar a análise de protótipos de HMIs no contexto [1] em que estes operam, por forma a possibilitar a identificação de possíveis falhas no *design* que, por sua vez, poderão ter repercussões significativas. O foco principal será no domínio automóvel. Painéis de instrumentos são um dos componentes vitais presentes em todos os veículos que permitem que os condutores possam controlar, dentro das suas melhores capacidades, o veículo e deslocar-se para onde pretendem. Para tal acontecer, estes painéis providenciam um conjunto de indicações sobre o estado atual do veículo. Sempre que uma ação do condutor é executada, os resultados dessa ação devem ser apresentados aos condutores, segundo padrões e normas definidos na indústria automóvel. Por exemplo, um novo valor de velocidade, após a aceleração, deverá surgir no velocímetro ou no ecrã táctil correspondente com a devida métrica visível, que para este exemplo poderá ser quilómetros por hora ou milhas por hora, dependendo da métrica em vigor no país do condutor.

Para minimizar o risco de acidentes relacionados com as tarefas auxiliares executadas pelo condutor, que são necessárias para executar a tarefa principal (condução), é importante que os painéis de instrumentos tenham uma boa *Human-Machine Interface (HMI)* e que não perturbem a condução, mas sim que auxiliem a mesma. Uma boa HMI de painéis de instrumentos deve ser fácil de usar, deve ter os seus componentes bem posicionados e organizados. Deve também fornecer um *feedback* conciso e assertivo sempre que o estado do veículo mudar, sem perturbar a condução.

Os fabricantes de automóveis continuam a produzir e a vender veículos com falhas de *design* de HMIs que em determinados contextos de utilização podem, potencialmente, ser prejudiciais ou até mesmo fatais. Então, é necessário melhorar o processo de teste que infere a qualidade dos *designs* de HMI de painéis de instrumentos, uma vez que cada segundo que um condutor utiliza a realizar uma tarefa auxiliar dentro do veículo, como tocar em ecrãs tácteis (*touch screens*) ou a tocar num conjunto de botões, é um segundo em que não está totalmente concentrado no que está a acontecer à sua volta, e que representa um grave risco não só para o condutor, mas também para todas as pessoas que estão nas imediações.

Recentemente tem havido um aumento da preocupação dos fabricantes de automóveis em produzir painéis de instrumentos com a melhor qualidade possível e que tenham em consideração o impacto que os contextos de utilização têm nas interfaces que os compõem.

Na indústria automóvel o foco tem sido a contratação de pessoas de diversas valências académicas para mitigar problemas de *design* relacionados com interações homem-máquina em certos ambientes. Por exemplo, o novo *Chief Executive Officer (CEO)* da *Ford Motor Company*, Jim Hackett, criou a equipa *Greenfield Labs*[5] dentro da *Ford Motor Company*, para explorar o futuro de mobilidade através do *design* centrado no ser humano. Essa equipa é constituída por *designers*, psicólogos, antropólogos e cientistas de dados que estão a colaborar para levar o *design* centrado no ser humano para os carros [15].

A prototipagem rápida é útil, pois permite iterar vários *designs* e refiná-los até um *design* final ser alcançado. Assim, é necessário um mecanismo para desenvolver essas interfaces de forma mais rápida, mais barata, que permita realizar modificações rápidas e que facilite a sua análise nos seus diferentes contextos de utilização.

1.3 OBJETIVOS

Esta dissertação visa cumprir dois objetivos. O primeiro objetivo é a criação de uma biblioteca, para a plataforma *PVSio-web*, que permitirá a configuração de diferentes ambientes de condução, como, por exemplo, a definição das imagens que compõem a paisagem, ou a definição da topografia e do perfil de elevação da pista, a definição da posição da câmara, que influencia o campo de visão dos ambientes de condução a renderizar, entre outras definições. Esta biblioteca deve também integrar controladores externos, como *gamepads*, para permitir a realização de testes mais realistas. Os ambientes de condução supracitados servem, posteriormente, como contextos de utilização durante a condução na pista.

O segundo objetivo é a concretização dessa biblioteca com a implementação de um simulador de ambientes de condução, onde será possível interagir com este utilizando o teclado físico do computador ou outros controladores externos, que se ligam via *USB* ou *Bluetooth* ao computador.

Esse simulador permitirá criar ambientes 2D que podem ter melhor ou mais definição dependendo da qualidade das imagens utilizadas. Esta solução utiliza gráficos 2D, protótipos de baixa fidelidade, em vez de gráficos 3D, protótipos de alta fidelidade, devido a altos custos de processamento e a problemas de desempenho inerentes a simulações 3D. Também o factor tempo foi crucial para a escolha de gráficos 2D, uma vez que o tempo de desenvolvimento de ambientes de condução 3D, como os mencionados anteriormente, é muito elevado em comparação com os ambientes 2D. Os protótipos 2D são mais rápidos e consomem menos recursos, visto que possuem bibliotecas menores. Estes protótipos possuem ainda código menor e mais legível. A aparência visual 3D pode ser alcançada com a manipulação do campo de visão e da posição da câmara ao longo do eixo Z, que é o eixo perpendicular ao ecrã de um computador.

5 `https://greenfieldlabs.com/` (*Greenfield Labs*)

PVSio-web é uma plataforma de prototipagem rápida, já utilizada em outros domínios, em particular no domínio dos aparelhos médicos. Os protótipos são construídos utilizando *widgets*. O conceito de *widget* assenta na criação de áreas interativas por cima de uma ou mais imagens do aparelho real, colocadas como plano de fundo no protótipo. Estas áreas interativas permitem executar uma dada ação, cuja especificação formal *PVS* se encontra especificada num ficheiro à parte e que em tempo de execução é invocada, e observar os resultados que advêm de tal ação. Um *widget* é também responsável por criar todo o *Markup HTML/CSS* necessário que contém os botões, os *displays*, entre outros. Um *widget* possui ainda um conjunto de *Application Programming Interfaces (APIs)* que o diferenciam e que vão desde APIs simples como esconder ou visualizar o *widget* até APIs mais avançadas que permitem alterar a sua interface por forma a refletir os comportamentos presentes na sua especificação formal.

Esta plataforma disponibilizou os *widgets* velocímetro e tacómetro funcionais e configuráveis. É possível fazer qualquer modificação relativamente à aparência visual destes *widgets*, como alterar as cores ou o intervalo de valores, facilmente e visualizá-la rapidamente. Estes *widgets* irão compor o painel de instrumentos do caso de estudo, após a definição de algumas opções de configuração no momento da suas instanciações que permitem obter a aparência visual desejada.

Para resumir, a solução consistiu em desenvolver um conjunto de *widgets*, que estendem a plataforma *PVSio-web*, na área automóvel que permitem construir uma simulação interativa, onde se podem simular os ambientes de condução como estradas, com diferente número de faixas de rodagem, com seções rectas e/ou curvas à direita e/ou à esquerda com perfil plano, ascendente ou descendente, e como diferentes objetos a colocar na paisagem, por exemplo, árvores, edifícios, entre outros. Consistiu também no desenvolvimento de *widgets* que permitem a utilização de controladores externos, como *gamepads* de consolas de jogos populares (comando da *PlayStation 4* ou *XBOX One*), em qualquer simulação interativa disponível na plataforma *PVSio-web*, em particular na simulação interativa que compõe a solução proposta.

1.4 CONTRIBUIÇÃO

As contribuições desta dissertação apresentam-se de seguida,

1. Dotar o *toolkit* de prototipagem rápida, *PVSio-web* [12], de um mecanismo que permite criar e visualizar contextos de utilização, no ramo automóvel. É concretizada com a implementação de um simulador 2D de corridas que foca no acto de conduzir e que permite avaliar o painel de instrumentos durante a condução numa estrada com uma dada topografia e com obstáculos a evitar.

2. O desenvolvimento dos protótipos das duas versões dos painéis de instrumentos do veículo do caso de estudo, tendo em vista validar o simulador do ponto 1. As duas versões supracitadas dizem respeito à interface com a falha de *design* reportada pelos proprietários e à interface com a posterior correção pelo próprio fabricante.

1.5 ESTRUTURA DA DISSERTAÇÃO

O documento de dissertação está estruturado da seguinte forma,

- Capítulo 1 - *Introdução*: enuncia a motivação, a contextualização, os objetivos e a contribuição para a solução proposta;
- Capítulo 2 - *Background*: apresenta a definição de painéis de instrumentos e discute ferramentas de prototipagem relacionadas com a solução proposta;
- Capítulo 3 - *Simulador de Condução*: apresenta a abordagem proposta utilizada para a prototipagem rápida de ambientes de condução. A arquitetura do simulador de condução e as características do *toolkit* de prototipagem rápida *PVSio-web* são também apresentadas. Também são apresentadas as tecnologias utilizadas na implementação do simulador de condução;
- Capítulo 4 - *Implementação do Simulador de Condução*: apresenta todos os *widgets* desenvolvidos que permitem a construção de diferentes ambientes de condução (paisagem e pistas) no simulador de condução;
- Capítulo 5 - *Protótipos Desenvolvidos*: apresenta os protótipos e a especificação formal (modelo *PVS*) desenvolvidos para o caso de estudo;
- Capítulo 6 - *Conclusão*: apresenta um resumo das contribuições desta dissertação e fornece algumas possibilidades de trabalho futuro.

2

BACKGROUND

Nesta secção, os conceitos "prototipagem", "prototipagem de ambientes" e "prototipagem rápida" serão discutidos. Também são apresentadas *frameworks/toolkits* que podem ser utilizadas para concretizar soluções que assentam em tais conceitos. É ainda apresentado em que constitui um painel de instrumentos de veículos e alguns dos seus principais componentes.

> *"USER INTERFACE prototype - Functional, computer-based simulation of the USER INTERFACE that enables representative USERS to perform realistic tasks."*
> *"Computerized prototypes are especially valuable for testing alternative USER INTERFACE designs prior to the production of operational models."*
>
> *Cited in: BS EN 62366:2008 - ICS 11.040.01*
> *BRITISH STANDARD - Medical devices*
> *Application of usability engineering to medical devices*

De acordo com a norma *BS EN 62366: 2008* [3], os protótipos de UI permitem criar representações virtuais de um novo produto, nas quais tarefas realistas podem ser realizadas. Os protótipos virtuais são particularmente úteis para testar de forma contínua e rigorosa *designs* alternativos até que um *design* final seja alcançado.

A prototipagem pode ser realizada fisica ou virtualmente. Os protótipos virtuais são mais baratos e permitem realizar modificações no *design* mais rapidamente, mitigando o atraso associado à aplicação de tais modificações no respectivo protótipo físico.

A prototipagem de ambientes é utilizada para criar os contextos de utilização virtuais do produto, que se aproximam dos ambientes reais onde o produto final irá operar. Esta prototipagem quando combinada com os protótipos, que operam nesses ambientes, é útil porque permite testar e analisar o impacto que os contextos de utilização têm no *design* das interfaces criadas. Quando a prototipagem não tem em consideração os ambientes onde o produto irá operar, verifica-se, às vezes, que ao executar uma dada tarefa numa dada interface, para um mesmo conjunto de etapas, os resultados obtidos são diferentes. O caso de estudo do veículo Lincoln MKC de 2015, apresentado na secção 1.1, no capítulo 1, é um exemplo em que o contexto tem influência na execução da tarefa. Na interface do painel

de instrumentos deste veículo quando o condutor pretendia ativar o modo desportivo do veículo, sem desviar o olhar da sua vizinhança, acabava por clicar no botão que desligava o veículo. É mais provável este problema acontecer durante a condução do que quando os condutores estão parados, uma vez que os condutores estão atentos à estrada. Para resolver tal situação, o fabricante teve que modificar a interface do painel de instrumentos.

> *"The faster that a prototype can be created and modified, the more likely it is that USER test results have a real impact on the product's design."*
> *"It is useful to employ software tools that facilitate rapid prototyping to allow changes at low cost."*
>
> ---
>
> *Cited in: BS EN 62366:2008 - ICS 11.040.01*
> *BRITISH STANDARD - Medical devices*
> *Application of usability engineering to medical devices*

De acordo com a norma *BS EN 62366: 2008* [3], a prototipagem rápida permite aos desenvolvedores criar representações de produtos, facilmente modificadas, em várias iterações, nas quais o *design* do produto é refinado. A prototipagem rápida também produz resultados visíveis mais cedo, permitindo uma análise do *design* atual mais cedo, que culmina, possivelmente, com a criação de HMIs com melhor qualidade. Embora a norma citada em ambos pertença ao domínio de dispositivos médicos, é possível transpor e aplicar os mesmos princípios no domínio automóvel.

2.1 PAINÉIS DE INSTRUMENTOS

Painéis de instrumentos são um dos elementos presentes nos veículos que auxiliam o processo de condução. Os painéis de instrumentos são cada vez mais completos, com interfaces que permitem concretizar mais ações, como por exemplo, botões para regular a temperatura no interior e nos bancos do veículo (ar condicionado), ecrãs tácteis com menus de navegação *Global Positioning System (GPS)*, câmaras e sensores de estacionamento, botões para alterar a potência do veículo (modos desportivos), entre outras funcionalidades.

A Figura 8 mostra a parte do painel de instrumentos do veículo do caso de estudo, que é relevante ao processo de condução, uma vez que possui as interfaces com que o condutor interage para controlar adequadamente o veículo e para que o código da estrada em vigor seja respeitado, evitando, portanto, contra-ordenações graves relacionadas com o posicionamento do veículo e com a sua velocidade excessiva. As interfaces mais relevantes durante a condução são o volante, o velocímetro e o tacómetro. O velocímetro utiliza um sensor eletrónico para medir a velocidade das rodas, contudo a sua precisão pode ser afetado pelo tamanho dos pneus do veículo. Se os pneus tiverem um diâmetro maior que o equipamento original, o velocímetro medirá valores menores que a realidade. As métricas

Figura 8.: **Painel de Instrumentos Simples do Veículo *Lincoln MKC* de 2015**

para a velocidade variam consoante as métricas em vigor no país onde o veículo circulará. O tacómetro mede a rapidez com que o motor está a girar. O volante é um componente circular ou oval, na maioria dos veículos, que permite alterar a direção do veículo em movimento ao alterar a direção das rodas.

2.2 PROTOTIPAGEM

Esta secção fornece uma visão geral de *frameworks* e trabalhos relacionados que suportam o processo de prototipagem rápida para protótipos e para protótipos de ambientes.

2.2.1 *Interfaces e Interações Homem-Máquina*

HMI, é toda a interface que traduz os *inputs* dos utilizadores para uma linguagem máquina e os *outputs* dessa linguagem máquina para a linguagem do utilizador, por forma a que este consiga perceber o resultado da ação executada.

Interação homem-máquina (do inglês *Human-Computer Interactions (HCI)*) é um campo multidisciplinar de estudo e de pesquisa que visa desenvolver HMIs seguras, fáceis de utilizar e que sejam criadas tendo em consideração a eficiência e o conforto durante a sua utilização num ambiente de trabalho[1] [9]. Esta área foca-se na descoberta de métodos e técnicas que dão suporte às atividades que as pessoas realizam nas diversas máquinas com que interagem.

1 `https://www.interaction-design.org/literature/topics/human-computer-interaction` (*Human-Computer Interaction (HCI)*)

A prototipagem de papel tem sido usada como uma técnica de prototipagem rápida, o que permite que as mudanças no *design* da UI sejam feitas com facilidade [10]. A fidelidade inerente a esta abordagem é muito baixa, o que é um problema para o processo de análise de *design*.

2.2.2 *Ferramentas de Prototipagem*

Prototipagem APEX

A *framework Agile Prototyping for user EXperience (APEX)* é uma plataforma de prototipagem rápida de ambientes de computação ubíquos conduzidos por um modelo do meio ambiente[2][3]. No entanto, APEX não é particularmente adequado para a criação de protótipos de UI, uma vez que se concentra mais no espaço onde os sensores são implantados, em prototipagem de sistemas ubíquos.

Um caso de estudo que utilizou esta *framework* foi um lar desenvolvido em [18]. O objetivo final foi desenvolver um sistema, que poderia ser utilizado para gerir melhor o espaço e para fornecer serviços relevantes aos seus utilizadores.

A *framework* APEX integra o servidor aplicacional 3D, *OpenSimulator*, com a ferramenta de modelação, *CPN tools* e com *Smartphones Android*.

OpenSimulator[4] é um servidor aplicacional 3D multi-plataforma, *open source*, e pode ser utilizado para criar e simular ambientes virtuais semelhantes a *Second Life*[5]. *OpenSimulator* é um mundo virtual online, onde os utilizadores criam representações virtuais de si próprios ou de alguém que gostariam de ser, também conhecido como avatares[6]. Estes avatares podem explorar esse mundo virtual e interagirem com lugares, objetos e outros avatares.

CPN Tools[7] é uma ferramenta de modelação[8] que permite editar, simular e analisar redes *Petri*.

Prototipagem PVSio-web

A plataforma *PVSio-Web*[9] [5] [4] é uma aplicação *open source*[10] que permite que qualquer indivíduo desenvolva e mantenha novos recursos como acontece nesta dissertação.

---

2 `http://ivy.di.uminho.pt/apex/` (*APEX Framework*)
3 `http://wiki.di.uminho.pt/twiki/bin/view/Research/APEX/ProjectSummary` (*The project in a Nutshell*)
4 `http://opensimulator.org/wiki/Main_Page` (*OpenSimulator*)
5 `http://secondlife.com/` (*Second Life*)
6 `http://opensimulator.org/wiki/Avatar` (*Create a meshed human avatar for OpenSimulator*)
7 `http://cpntools.org/` (*CPN Tools*)
8 `https://www.scmagazine.com/cpn-tools/review/6142/` (*Product Information CPN Tools, P. Stephenson, November 06, 2009*)
9 `http://www.pvsioweb.org` (*PVSio-Web*)
10 `https://github.com/thehogfather/pvsio-web` (Repositório *GitHub*)

O foco principal desta plataforma é a simulação de UI de sistemas reais, de diferentes áreas, como as áreas automóvel e de dispositivos médicos, aplicando o conceito de prototipagem rápida. *PVSio-web* é um ambiente gráfico para desenvolver sistemas interativos, realistas e sofisticados, com base em animações especificadas em modelos formais, escritas com a linguagem formal *PVS* [17], onde os utilizadores podem experimentar esses sistemas, sem qualquer risco e visualizando as modificações no *design* rapidamente. Tem sido amplamente utilizado, com sucesso, na aprendizagem e treino de dispositivos médicos críticos, facilitando a interação homem-máquina, que, eventualmente, diminui os erros humanos que ocorrem nos sistemas reais.

Este *toolkit* permite a criação de protótipos com diferentes níveis de fidelidade. *PVSio-web* está totalmente implementado em *JavaScript*, com uma arquitetura distribuída, baseada num servidor *web* e num cliente *lightweight*. *PVSio-web* executa os processos *PVS* e *PVSio* num servidor *web*, utilizando *websockets* para suportar a comunicação. A Figura 9 demonstra a arquitetura de alto nível de *PVSio-web*. Nesta arquitetura as funções definidas pelo modelo formal *PVS* são disponibilizadas, através de *widgets*, no *Frontend*.

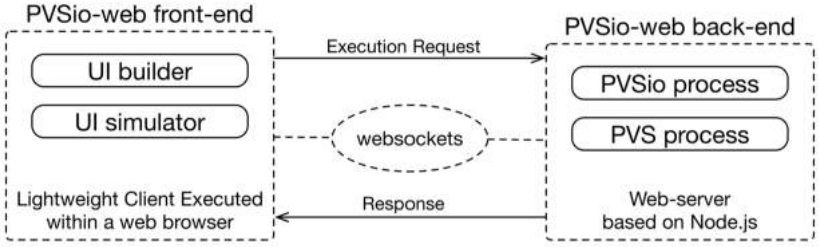


Figura 9.: **Arquitetura PVSio-web**

Fonte: Page 7-[11]

A Figura 10 mostra alguns dos protótipos, na área dos dispositivos médicos, que foram construídos com *widgets*, disponibilizados pela plataforma *PVSio-web*. A ferramenta *Prototype Builder* fornece uma interface para construir os protótipos (*Builder View*) e outra para os animar (*Simulator View*). Um protótipo na área do dispositivo médico que foi implementado, usando o *toolkit PVSio-web*, é a bomba de insulina *medtronic 530g* [11], que ajuda os diabéticos a controlar os seus níveis de glicose. O protótipo *medtronic 530g* pode ser consultado e/ou modificado em *Builder View*, na ferramenta *Prototype Builder*, como se observa na Figura 11. Pode-se observar que este protótipo possui apenas três *widgets*, sendo esses dois *widget Button*, *UP* (seta azul) e *DOWN* (seta amarela), que adiciona um botão interativo sobre a imagem do aparelho médico, e um *widget Numeric Display*, *display* (seta vermelha), que adiciona um *display*, onde aparecem os valores atuais da simulação, obtidos a partir

11 `https://www.medtronicdiabetes.com/products/minimed-530g-diabetes-system-with-enlite` (Medtronic 530g Insulin Pump)

do estado atual *PVS* da especificação formal. As Figuras 12 e 13 mostram o processo de alteração das configurações dos *widgets button DOWN* e *Numeric Display display*. No caso do botão *DOWN* é possível verificar que foi selecionada a opção de capturar eventos *"Click"* em vez de *"Press/Release"*, o que implica que a especificação formal terá função *click_DOWN* em vez de duas funções *press_DOWN* e *release_DOWN*, que determina o comportamento do protótipo perante o clique do botão *DOWN*. No caso do *display* pode-se verificar que se escolheu uma fonte com tamanho *32px*, com fundo transparente. Estes menus de configuração permitem fornecer os campos opcionais aos construtores desses *widgets* de uma forma bastante simples e intuitiva. Esses campos opcionais são os elementos que permitem customizar os *widgets*.

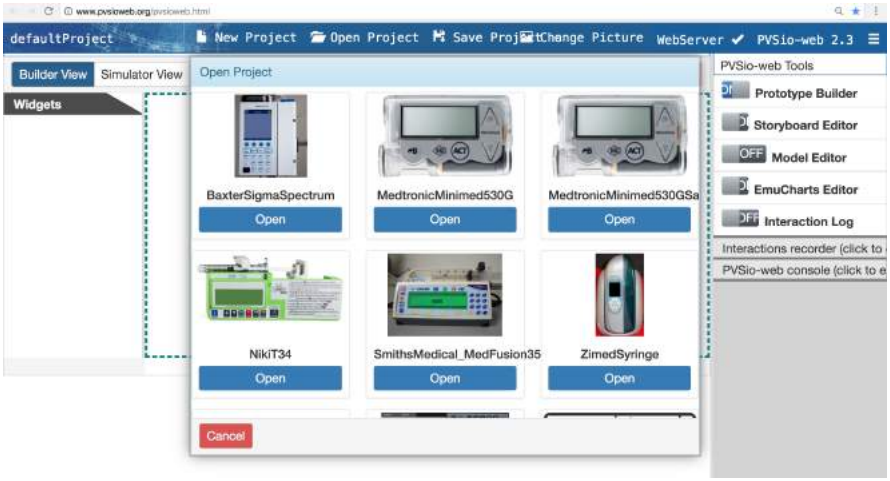


Figura 10.: **Alguns Projetos disponibilizados pela Interface Gráfica de PVSio-web**

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

Figura 11.: **Ferramenta *Prototype Builder* no Projeto *Medtronic530G*** (A seta vermelha indica o *widget Numeric Display*. As setas azul e amarelo indicam os *widget Button*.)

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

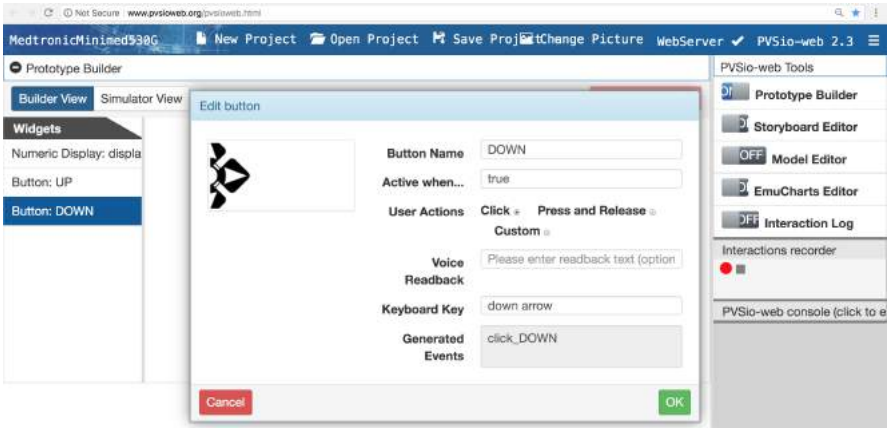


Figura 12.: **Configuração de um *widget button* no Projeto *Medtronic530G***

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

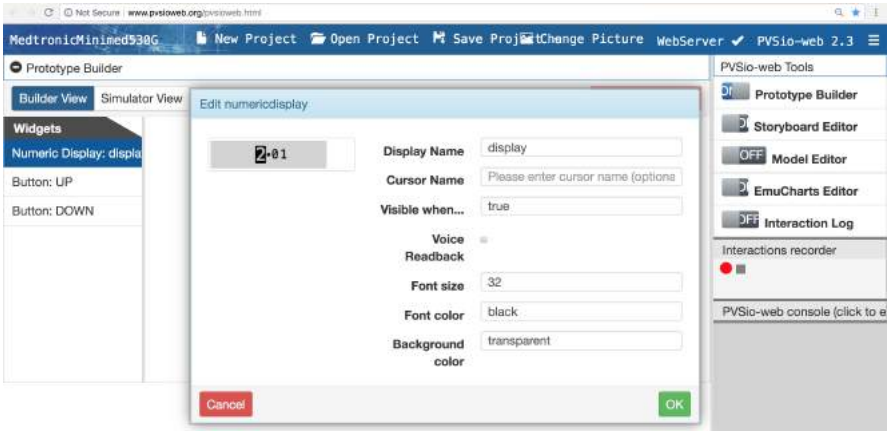


Figura 13.: **Configuração de um *widget display* no Projeto *Medtronic530G***

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

A Figura 14 apresenta o editor que permite alterar a especificação formal *PVS* de forma gráfica, o que facilita o processo para os utilizadores com menores conhecimentos na área de especificação e verificação formal. É possível observar a definição dos comportamentos das funções *click_UP* e *click_DOWN* e quais os resultados das respectivas interações no *widget display* do protótipo.

A Figura 15 permite visualizar e modificar os ficheiros com as especificações formais para o protótipo *medtronic 530g*. Este editor é útil para alterar as especificações e observar a mudança imediata nos comportamentos do protótipo. A plataforma *PVSio-web* suporta especificações formais *PVS*[12].

12 `http://pvs.csl.sri.com/download.shtml` (*PVS(Prototype Verification System) - Specification and Verification System*)

Figura 14.: **Ferramenta *Emucharts Editor* no Projeto *Medtronic530G***

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

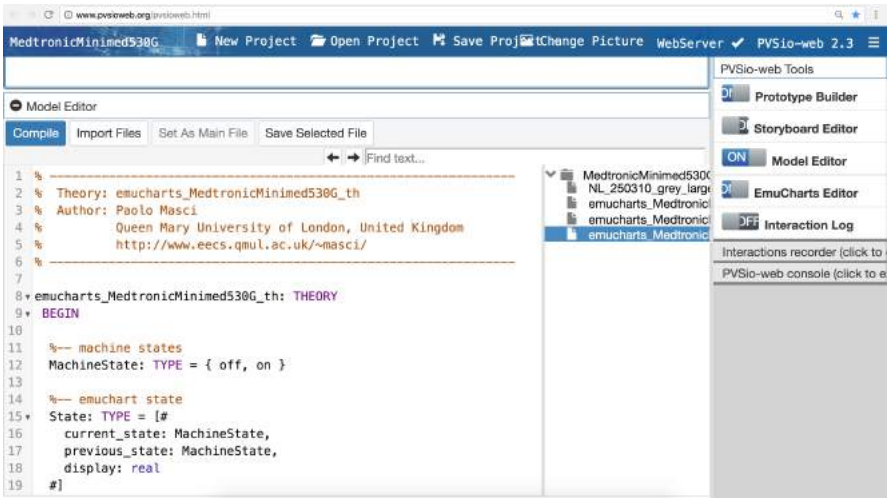


Figura 15.: **Ferramenta *Model Editor* no Projeto *Medtronic530G***

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

Para adicionar novos *widgets* num protótipo o utilizador terá de escolher o tipo de *widget*, Figura 17, sendo que os tipos disponíveis são:

- *Button* que é um *widget* clicável, que permite executar acções, invocando funções definidas na especificação formal do modelo *PVS*,
- *Basic Display* que é um *widget* que permite visualizar texto na interface do protótipo,
- *Numeric Display* que é um *widget* que permite visualizar valores numéricos na interface do protótipo,
- *Touchscreen Display* que é um *widget*, orientado a ecrãs tácteis, que permite visualizar texto na interface do protótipo,

- *Touchscreen Button* que é um *widget* clicável, orientado a ecrãs tácteis, que permite executar acções, invocando funções definidas na especificação formal do modelo *PVS*,
- *LED* que é um *widget* que permite acender ou desligar um *LED*, de acordo com o estado do modelo *PVS*, na interface do protótipo.

Para cada *widget* selecionado será sempre necessário colocar o identificador do mesmo, por exemplo, no campo *Button Name* coloca-se o identificador do novo *widget Button*. É também necessário colocar para os *widget Button*, o evento que será capturado (*"Click"* ou *"Press/Release"*) e colocar uma tecla física do teclado que estará associada a esse evento (campo opcional *Keyboard Key*). Para os restantes *widgets* procede-se de igual forma para que se estabeleça a ligação entre o evento a ser capturado e a especificação formal do comportamento do dispositivo. O pressionar/libertar ou o clique ativa uma função definida na especificação formal do modelo *PVS* de acordo com a ligação estabelecida.

A Figura 16 permite realizar a simulação interativa do protótipo *medtronic 530g*, na janela *Simulator View* da ferramenta *Prototype Builder*. Esta simulação permite observar os comportamentos especificados formalmente para cada um dos *widgets* que compõem o protótipo.

Figura 16.: **Ferramenta *Prototype Simulator* no Projeto *Medtronic530G***

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

A plataforma *PVSio-web* possui, tal como já se apresentou na Figura 10, outros protótipos mais complexos que o protótipo *medtronic 530g* explicado acima, como é o caso do protótipo *AlarisPC_PumpModules*, na Figura 18. É possível verificar que este protótipo possui mais *widgets* colocados sobre a imagem do dispositivo médico (quadrados verdes), que produzem diferentes resultados nos diferentes *displays* desse protótipo, o que permite concluir que esta plataforma permite construir simulações mais ou menos simples, com especificações mais ou menos complexas.

A plataforma *PVSio-web* para além da interface gráfica abordada acima e apresentada na Figura 10 permite criar um protótipo criando os ficheiros de configuração e instanciando os

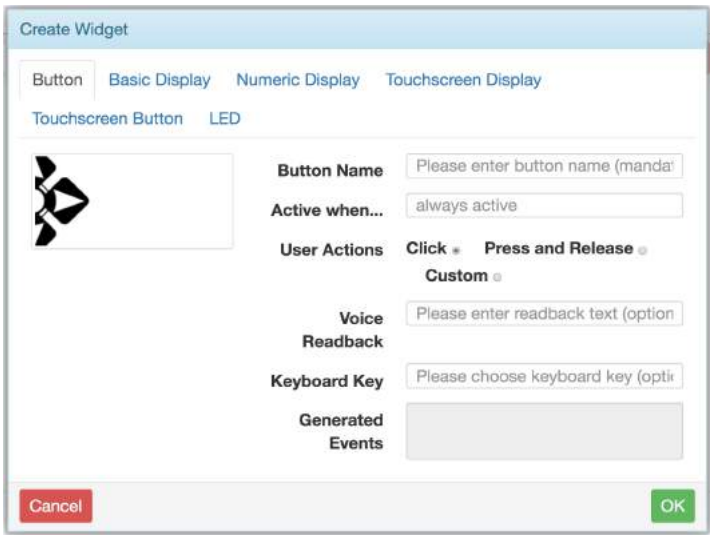


Figura 17.: **Criação de um *Widget* na Interface Gráfica de PVSio-web**

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

widgets manualmente. Consiste, então, na criação de uma pasta com a seguinte hierarquia e conteúdo:

- Imagem de Fundo, que corresponde à imagem do dispositivo cujo protótipo se pretende criar.
- Ficheiro *index.html* com todo o *Markup HTML* que compõe a página *web* da simulação interativa. Os *widgets* instanciados irão adicionar os seus *Markups HTML* a este *Markup HTML*. Por exemplo, a instanciação do *widget button* irá adicionar uma *div* com o *Markup HTML* que permite colocar um botão interativo sobre a imagem de fundo (definir as coordenadas e a posição).
- Pasta *css* com *Markup CSS* para a página *web* da simulação interativa. Esta pasta tem os ficheiros *960.css* e *style.css*, que são os ficheiros colocados no cabeçalho da página *web*, descrita no ficheiro *index.html*.
- Pasta *pvs* com a especificação formal (modelo *PVS*) no ficheiro *main.pvs* do protótipo a criar.
- Ficheiro *markdown* com a descrição do protótipo e das teclas que permitem ao utilizador interagir com a interface da simulação interativa.
- Pasta *js* com a instanciação de todos os *widgets* que compõem o protótipo a criar, bem como a comunicação com o servidor no ficheiro *index.js*. Essa comunicação será responsável por enviar o estado atual do modelo *PVS* para os *widgets* instanciados.

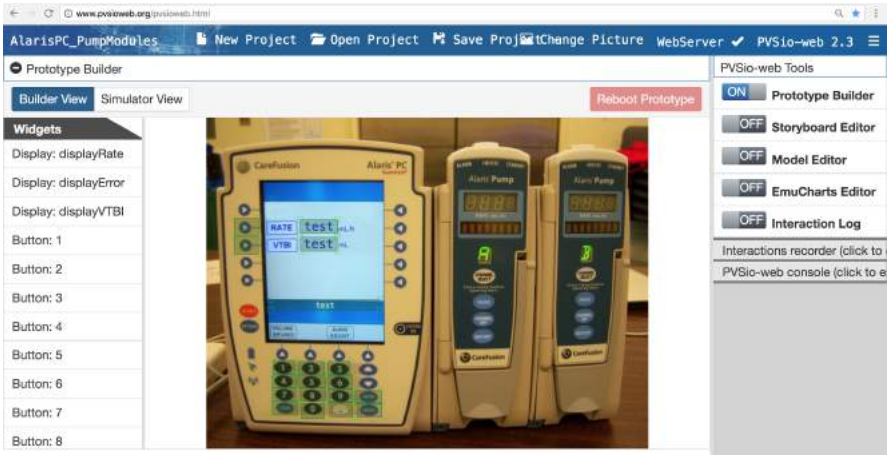


Figura 18.: **Projeto *AlarisPC_PumpModules***

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

De seguida apresenta-se a explicação do protótipo de um dispositivo médico, *Radical7*. A Figura 19 é a imagem de fundo, sobre a qual todos os *widgets* serão colocados, por forma a criar a simulação interativa desejada. O dispositivo médico *Radical-7 Pulse CO-Oximeter*[13] é um aparelho que realiza a medição da saturação de oxigénio, do ritmo do pulso, do índice de perfusão, do índice de variabilidade *Pleth*, da hemoglobina total, da carboxihemoglobina, da metemoglobina e da taxa de respiração acústica. O protótipo deste dispositivo visa simular a interface do dispositivo médico *Radical-7 Pulse CO-Oximeter*, com base em dois sensores, o sensor *Masimo SET SpO2*, que mede a quantidade de oxigénio no sangue, e o sensor *rainbow SET* atualizável *RRa (Acoustic Respiration Rate)*, que fornece a monitorização não invasiva e contínua da frequência respiratória[14]. *SpO2* é a percentagem de hemoglobina que contém oxigénio em relação à quantidade total de hemoglobina no sangue.

A Figura 20 mostra o protótipo inicial. A Figura 21 mostra uma interação homem-máquina no protótipo *Radical7*, onde o utilizador inseriu os valores 220 e 100, para os sensores *SPO2* e *RRa*, e onde se observa a alteração dos valores apresentados nos *displays* colocados na parte direira do ecrã do dispositivo médico. O novo valor do sensor *RRa* lançou um alerta sonoro, sendo este indicado pelo sino vermelho.

A instanciação dos *widgets Button btn_on*, *Basic Display spo2_display* e *Basic Display rra_-display*, no ficheiro *index.js*, no Excerto de Código 1, do protótipo *Radical7*, adiciona o *Markup HTML* presente parcialmente nas Figuras 22 e 23, respectivamente, ao ficheiro *index.html*. A Figura 24 mostra parte do *Markup HTML* relacionado com os campos de texto, onde o utilizador insere os novos valores das medições dos sensores *SPO2* e *RRa*, que

13 `http://www.masimo.com/products/continuous/radical-7/` (*Radical-7 Pulse CO-Oximeter*)
14 `http://www.masimo.com/technology/oxygenation/rra/` (*RRa - Acoustic Respiration Rate*)

Figura 19.: **Imagem de Plano de Fundo do Aparelho *Radical7* Real**

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

Figura 20.: **Página Inicial do Protótipo *Radical7***

Fonte: `http://www.pvsioweb.org/demos/Radical7/`

corresponde à interação homem-máquina neste protótipo, que foi também adicionado ao ficheiro *index.html*.

```
var radical = {};
radical.spo2_display = new PatientMonitorDisplay("spo2_display",
    {top: 56, left: 150, height: 34, width: 160},
    {parent: "prototype", label: "%SpO2"});

radical.rra_display = new PatientMonitorDisplay("rra_display",
    {top: 102, left: 150, height: 34, width: 160},
    {parent: "prototype", label: "RRa", fontColor: "aqua"});

radical.btn_on = new Button("btn_on",
  {top: 112, left: 364},
  { callback: onMessageReceived });
```

Excerto de Código 1: **Instanciação de *widgets* do Protótipo Radical7**

Figura 21.: **Exemplo de uma Interação do Protótipo *Radical7***

Fonte: `http://www.pvsioweb.org/demos/Radical7/`

```
▼<div style="height: 0px; width: 0px;">
  ▶<div id="btn_on" style="position: absolute; width: 32px; height: 32px; top: 112px; left: 364px;
  z-index: 1; display: block;" class="ButtonEVO noselect">…</div>
  </div>
```

Figura 22.: ***Markup do Widget button* do Protótipo *Radical7***

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

```
▼<div id="prototype">
    <img src="radical-7.jpg" style="width:580px;">
    <div style="margin:20px; margin-left:30px;">Simulation of Masimo Radical 7 PulseOximeter</div>
  ▼<div id="spo2_display" class="spo2_display" style="position: absolute; top: 56px; left: 150px;
  width: 160px; height: 34px; margin: 0px; padding: 0px; background-color: rgb(0, 0, 0); display:
  block;">
    ▼<div style="height: 0px; width: 0px;">
      ▶<div id="spo2_display_value" style="position: absolute; width: 38.4px; height: 20.4px; top:
      0.2px; left: 110.4px; z-index: 0; display: block;" class="BasicDisplayEVO noselect">…</div>
      </div>
    ▶<div style="height: 0px; width: 0px;">…</div>
    ▶<div style="height: 0px; width: 0px;">…</div>
    ▶<div style="height: 0px; width: 0px;">…</div>
```

Figura 23.: ***Markup do Widget display* do Protótipo *Radical7***

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

```
  ▶<div id="rra_display" class="rra_display" style="position: absolute; top: 102px; left: 150px;
  width: 160px; height: 34px; margin: 0px; padding: 0px; background-color: rgb(0, 0, 0); display:
  block;">…</div>
  </div>
▼<div id="spo2sensor" style="padding: 10px 28px 10px 28px; border: lightgreen; border-style:
solid;">
    "
                  Simulation of the SPO2 Sensor:
                  "
    <input type="text" id="spo2_sensor_data" name="spo2reading" placeholder="Please enter a new
    SPO2 reading..." size="44">
    <input type="submit" id="submit_spo2_sensor_data" value="Submit new SPO2 reading">
  </div>
▶<div id="rrasensor" style="padding: 10px 28px 10px 28px; border: steelblue; border-style: solid;
```

Figura 24.: ***Markup* dos Campos que Simulam os Sensores do Protótipo *Radical7***

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

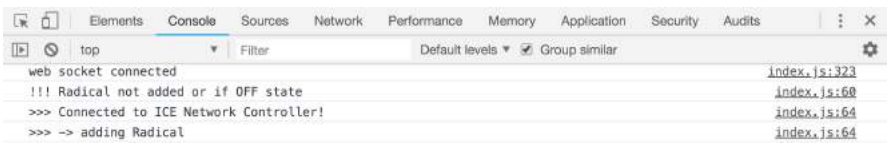


Figura 25.: **Exemplo da Consulta do Estado *PVS* do Protótipo *Radical7***

Fonte: `http://www.pvsioweb.org/pvsioweb.html`

Os protótipos na plataforma *PVSio-web* estão constantemente a atualizar o estado do modelo *PVS* na consola do *browser*, na Figura 25, caso a função *tick* esteja a ser invocada.

O estado do modelo *PVS* é lido, interpretado e enviado como argumento aos *widgets* como demonstra o Excerto de Código 2. Os *widgets* representam a informação do modelo *PVS* ao utilizar esse estado para atualizar o seu próprio estado e agir consoante o mesmo.

```
function onMessageReceived(err, event) {
  // ...
  var res = stateParser.parse(event.data.toString()); // estado PVS atual
  if (res) {
    render_spo2(res); // renderiza o \textit{widget} do sensor SPO2
    render_rra(res);  // renderiza o \textit{widget} do sensor RRA
    render_alarms(res);  // renderiza os \textit{widgets} de alarmes
    radical.btn_on.render(res); // renderiza o \textit{widget} Button btn_on
  }
  // ...
}
```

Excerto de Código 2: **Renderização dos *widgets* do Protótipo Radical7 com base no estado *PVS* atual**

O Excerto de Código 28, página 109 (Anexo A), apresenta a especificação formal do protótipo do dispositivo médico *Radical-7 Pulse CO-Oximeter*. Por exemplo, quando o utilizador inserir um novo valor no sensor *SPO2*, o modelo de *PVS* atualiza o valor dos atributos *spo2* e *spo2_fail* do seu estado (linhas 8 e 13) com a função *spo2_sensor_data* (linhas 98 e 99). Na interface do protótipo, o *widget Basic Display spo2_display* irá apresentar o novo valor inserido pelo utilizador, uma vez que este representa a informação do atributo *spo2* do modelo *PVS*. A inserção de um novo valor para o sensor *RRa* decorre de igual forma.

Seria útil para este *toolkit* permitir a criação e o desenvolvimento de protótipos de ambientes para que seja possível testar os protótipos no seu contexto de utilização por oposição ao teste de protótipos de forma isolada, que é o que neste momento é possível. Esta combinação facilitaria o processo de análise do *design* das suas interfaces, para garantir que o *design* final seja de maior qualidade e possibilitaria o aumento da consciencialização acerca da importância da utilização de contextos de utilização no processo de *design* de qualquer interface.

2.3 Síntese

Neste capítulo são apresentadas as definições dos conceito "prototipagem", "prototipagem de ambientes" e "prototipagem rápida", que são a base para alcançar a solução proposta. É apresentada a importância de combinar os conceitos "prototipagem" e "prototi-

pagem de ambientes", uma vez que os ambientes de condução afetam o desempenho de algumas tarefas, nas respectivas UI.

São apresentados os componentes dos painéis de instrumentos mais importantes no processo de condução. São apresentadas as *frameworks* relacionadas, APEX e *PVSio-web*, que podem ser utilizadas para desenvolver esses protótipos. Foi ainda apresentado um caso de estudo, onde a *framework* APEX foi utilizada.

E, por fim, é apresentada a arquitetura da plataforma *PVSio-web* e o protótipo *PVSio-web* do dispositivo médico *Radical-7 Pulse CO-Oximeter*.

3

SIMULADOR DE CONDUÇÃO

Neste capítulo, descreve-se a abordagem proposta, nesta dissertação, cujo objectivo é expandir os protótipos de painéis de instrumentos de *PVSio-web*, que atualmente consistem em alguns *widgets*, velocímetro, tacómetro, caixas de velocidades, termómetros e relógios sem ter em consideração o contexto de condução. O problema em si está relacionado com a análise do *design* dos componentes do painel de instrumentos de veículos. Pretende-se que esses protótipos tenham em consideração o contexto de utilização para evitar a ocorrência de resultados indesejáveis na execução de tarefas no painel de instrumentos durante a condução.

A solução passa pela implementação de um simulador de condução e que para tal foram criados os *widgets TrackGenerator* e *Arcade*. O *widget TrackGenerator* permitirá a criação e o desenvolvimento de contextos de condução realistas. O *widget Arcade* permitirá a visualização desses contextos de condução sob a forma de um jogo de corridas de acordo com a personalização desejada pelo utilizador, como a alteração da imagem do veículo usada durante a simulação. Para demonstrar a utilidade dos *widgets* desenvolvidos nesta dissertação será utilizado o painel de instrumentos do veículo *Lincoln MKC* de 2015 como caso de estudo.

Outro objetivo que a solução proposta deverá resolver é a integração de controladores externos, inicialmente os mais simples, *gamepads* como o da *PlayStation 4*, lado esquerdo na Figura 26 e o da *XBOX One*, lado direito na Figura 26, e, posteriormente, mais complexos, como o volante e os pedais *Logitech G29*, Figura 27.

Figura 26.: **Controlador Externo (lado esquerdo: *PlayStation 4*; lado direito: *XBOX One*)**

Figura 27.: **Controlador Externo - *Logitech G29* - Pedais e Volante**

3.1 ABORDAGEM PROPOSTA

A solução proposta, nesta dissertação, implicou a construção de um simulador 2D de corridas, desenvolvido em *JavSscript*, *HTML5* e *CSS3*. Este simulador fornece um mecanismo de criação e de desenvolvimento de ambientes de condução, que servirão de contextos de utilização a aplicar aos protótipos de painéis de instrumentos.

Em particular, o simulador deverá receber um conjunto de imagens, independentemente da sua qualidade, e deverá colocar as mesmas numa dada posição, por forma a que o conjunto dessas imagens, a topografia e o perfil de elevação da estrada, parametrizada pelo utilizador, componha os ambientes de condução requeridos. Os ambientes de condução poderão, então, ter diferentes níveis de qualidade da aparência em função das imagens utilizadas.

A solução proposta deverá ainda permitir que para o processo de condução no simulador de corridas supracitado, o utilizador possa utilizar diferentes mecanismos de controlo, como o teclado de setas físico, o teclado de setas virtual ou um qualquer controlador externo.

Assim, o problema, explorado nesta dissertação, exige que a solução permita que o utilizador interaja com os *widgets* do painel de instrumentos ao conduzir, seja com um teclado de setas físico ou com um teclado de setas virtual, seja em dispositivos tácteis, como *tablets*, com os seus contextos de utilização e com uma "vista pára-brisas". A vista supracitada permitirá que o utilizador interaja com a simulação como se estivesse imediatamente por trás do *widget* volante, com os ambientes de condução criados na sua vizinhança.

3.2 ARQUITETURA DA SOLUÇÃO PROPOSTA EM PVSIO-WEB

As questões relacionadas com a arquitetura da solução proposta e a sua integração no *toolkit PVSio-web* serão abordadas nesta secção. A plataforma *PVSio-web* permite criar

um *design* do painel de instrumentos desejado, onde a linguagem formal *PVS* define o comportamento da interface e os *widgets* definem o aspecto da interface. A Figura 28 demonstra como a plataforma *PVSio-web* separa o comportamento e o aspecto visual da interface de um protótipo durante a simulação interativa.

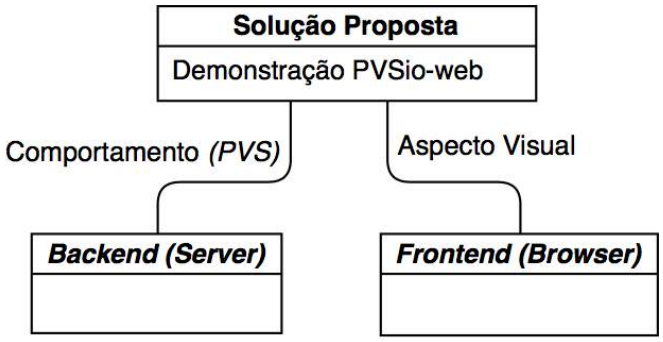


Figura 28.: **Arquitetura de um Protótipo**

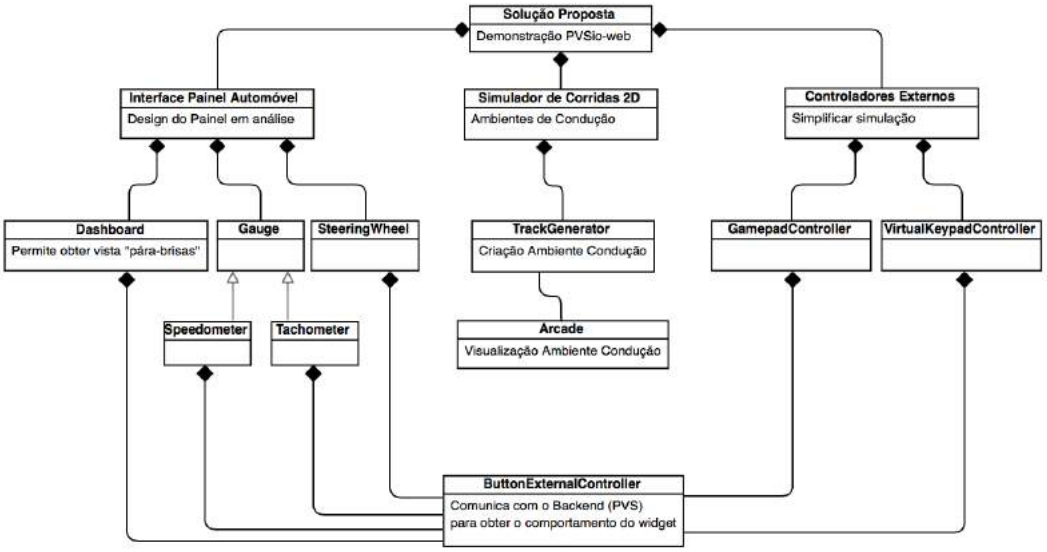


Figura 29.: **Arquitetura da Solução Proposta**

A solução proposta integrará os *widgets Gauge*, *Speedometer* e o *Tachometer*, que já existem na plataforma *PVSio-web*, e integrará *widgets* novos. Os *widgets* novos a implementar são *Dashboard*, *SteeringWheel*, *TrackGenerator*, *Arcade*, *GamepadController*, *VirtualKeypadController* e *ButtonExternalController*. A Figura 29 demonstra como os *widgets* compõem o protótipo da solução proposta. De seguida apresenta-se uma breve descrição da finalidade de cada um desses *widgets* no contexto da solução proposta.

A interface do painel de instrumentos será composta pelos *widgets Gauge*, *Speedometer*, *Tachometer*, *SteeringWheel* e *Dashboard*. O *widget Speedometer* é responsável pela construção da interface que apresentará o valor da velocidade atual, que corresponde ao valor da velocidade que o veículo possui no simulador supracitado. O *widget Tachometer* é responsável pela construção da interface que apresentará o valor atual das rotações por minuto (rpm), que corresponde ao valor das rotações por minuto que o veículo possui no simulador supracitado. O *widget Gauge* permite criar a interface para um qualquer indicador circular do painel de instrumentos, que apresenta o valor atual, num intervalo de valores, para uma dada métrica. Os *widgets Speedometer* e *Tachometer* utilizam o *widget Gauge* para a criação da interface de um tipo de indicador circular específico do painel de instrumentos. O *widget SteeringWheel* será responsável por construir o volante na interface, cuja ação será a rotação do mesmo, com base na direção do veículo no simulador supracitado. Por exemplo, quando o veículo for para a direita, este *widget* deverá reproduzir esse efeito rodando o volante para a direita. O *widget Dashboard* será responsável por criar a interface com um qualquer painel de instrumentos colocado numa dada posição por forma a obter a vista "pára-brisas". Este *widget* será depois especializado para cada caso concreto.

O simulador de condução será composto pelos *widgets TrackGenerator* e *Arcade*. O *widget TrackGenerator* será responsável pela criação dos ambientes de condução e pela criação da pista com base num conjunto de parâmetros que serão fornecidos pelo utilizador. Esses parâmetros incluirão as definições da topografia e dos perfis de elevação da pista. O *widget Arcade* será responsável pela visualização dos ambientes de condução com base em parâmetros que serão também fornecidos pelo utilizador.

A integração dos controladores externos possuirá os *widgets GamepadController* e *VirtualKeypadController*. O *widget GamepadController* permitirá controlar o simulador supracitado com um qualquer controlador externo, como acontece nas consolas de jogos atualmente. O *widget VirtualKeypadController* permitirá controlar o simulador supracitado com as teclas de setas virtuais da mesma forma que poderá controlar com um teclado físico. Este *widget* facilitará a simulação do protótipo em dispositivos tácteis, como *tablets*, onde o teclado físico e o sensor giroscópio não estão presentes, nem uma entrada *USB*, para que se possa conectar um controlador externo.

Os comportamentos dos *widgets* serão especificados formalmente utilizando a linguagem formal *PVS* e utilizarão o *widget ButtonExternalController* para capturar os eventos e estabelecer a ligação com o modelo formal *PVS* para obter o comportamento a reproduzir na interface. O *widget ButtonExternalController* será uma extensão do *widget Button*, existente na plataforma *PVSio-web*, que irá permitir a colocação de *icons* no *background* de um botão.

O Anexo B apresenta o *link* da documentação *online* relacionada com a instalação local da plataforma *PVSio-web*.

3.3 TECNOLOGIAS

3.3.1 *JavaScript*

A tecnologia *JavaScript*[1] [7] [16] é uma linguagem que suporta a programação orientada a eventos, por exemplo, permite controlar o comportamento de páginas *web* em resposta a um ou mais eventos, de acordo com o *Document Object Model (DOM)* dessas páginas.

A sintaxe é muito parecida com a sintaxe de linguagens orientadas a objetos como *Java* e *C++* o que acelera o tempo de aprendizagem. É uma das três tecnologias basilares de soluções *web*, e, como tal, existe uma vasta coleção de bibliotecas e *frameworks* que podem ser utilizadas para acelerar o processo de desenvolvimento de soluções. Atualmente, as *frameworks* mais potentes no mercado de soluções *web* são *AngularJS*[2] [8], *ReactJS*[3] [13] e *VueJS*[4] [6]. As bibliotecas utilizadas em soluções *web* para facilitar a modificação dos elementos DOM são *jQuery*[5] e *D3.js*[6].

3.3.2 *Spritesheets*

Uma *spritesheet*[7] é uma imagem que contém um conjunto de imagens individuais, *sprites*, agrupadas. Este mecanismo permite aumentar a eficiência de uma página *web* visto que o número de pedidos ao servidor é muito menor. Por exemplo, se uma página *web* necessitar de cem imagens, em vez de pedir uma imagem de cada vez, é realizado apenas um pedido, que corresponde à leitura da *spritesheet* que agrupa essas cem imagens numa só imagem.

A leitura das imagens pode ser realizada de duas formas. A primeira consiste em aceder a cada uma dessas imagens com base nas coordenadas *(x,y)* que a *sprite* tem na *spritesheet*. A segunda consiste em aceder a cada uma dessas imagens com base em propriedades *CSS* que indicam a posição relativa da *sprite*. O acesso à imagem via *Markup CSS* (*background-position*, *width*, *height*, etc) é aconselhado para a leitura e colocação de imagens diretamente pelo *Markup HTML*. Para aceder, processar e colocar as imagens numa dada posição no ecrã, em *JavaScript*, é aconselhado aceder às imagens com base nas coordenadas, para que depois se possa utilizar o método, da *Application Programming Interface (API) HTML5 Canvas*, que permite desenhar a imagem desejada na *Canvas* (*CanvasRenderingContext2D.drawImage()*).

1 `https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript` (*About JavaScript*)
2 `https://angularjs.org/` (*AngularJS*)
3 `https://reactjs.org/` (*ReactJS*)
4 `https://vuejs.org/` (*VueJS*)
5 `https://jquery.com/` (*jQuery*)
6 `https://d3js.org/` (*d3JS*)
7 `https://www.w3schools.com/css/css_image_sprites.asp` (*CSS Image Sprites*)

Existem várias ferramentas que permitem fazer o mapeamento das imagens presentes numa dada *spritesheet*. A ferramenta *TexturePacker*[8] é útil quando se pretende criar uma nova *spritesheet*, uma vez que também fornece um ficheiro *JSON* com o mapeamento de coordenadas das imagens agrupadas. Esta ferramenta é muito restritiva na versão gratuita, na medida em que impede a utilização de funcionalidades como a colocação de imagens por categorias. Nesta versão a colocação de imagens é realizada aleatoriamente. A ferramenta *online GetSpriteXY*[9] é útil quando se pretende mapear as imagens presentes numa *spritesheet* que já havia sido criada.

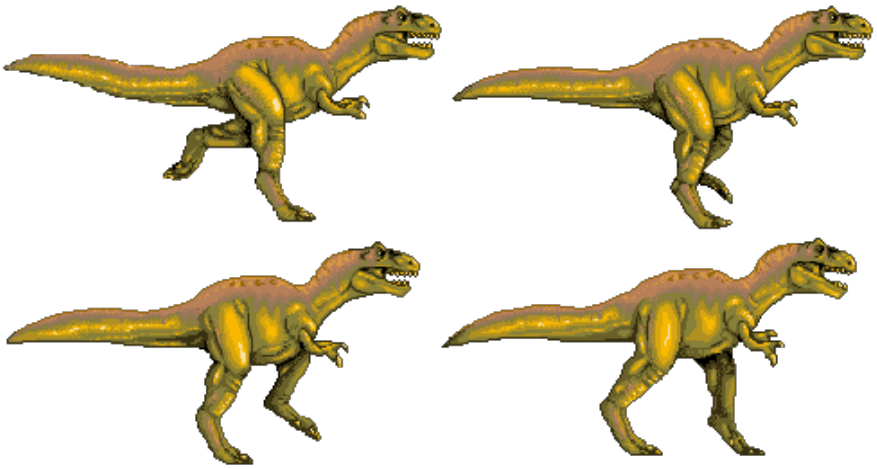


Figura 30.: ***Spritesheet* Criada com a Ferramenta *TexturePacker***

A Figura 30 mostra a *spritesheet* criada com a ferramenta *TexturePacker*, onde se pode observar que se uniram quatro *sprites* diferentes. A criação da *spritesheet* com essa ferramenta culmina com a criação de dois ficheiros. O primeiro ficheiro é a imagem *.png* com a *spritesheet* final. O segundo ficheiro é o ficheiro *JSON* com o mapeamento das coordenadas de cada uma das imagens que compõem a *spritesheet* final. O Excerto de Código 3 mostra o conteúdo do segundo ficheiro criado. Neste excerto é possível observar que o objeto *frames* é o objeto que contém todas as imagens agrupadas, cujos campos mais importantes são o *filename* e *frame*. O campo *filename* é útil para filtrar o objeto *frames* que se pretende desenhar. Por exemplo, numa solução *web*, desenvolvida em *JavaScript*, para desenhar a primeira imagem da segunda fila da Figura 30 será apenas necessário fornecer a *spritesheet* final (Figura 30), acompanhada das coordenadas obtidas no campo *frame*, neste caso, *"x":0,"y":127,"w":244,"h":132*. O Excerto de Código 4 demonstra a utilização do método supracitado para desenhar essa imagem.

8 `https://www.codeandweb.com/texturepacker` (*TexturePacker*)
9 `http://getspritexy.com/` (*GetSpriteXY*)

```
{ "frames": [
  {
  "filename": "tyrannosaur2_1.png",
  "frame": {"x":0,"y":0,"w":252,"h":125},
  "rotated": false,
  "trimmed": false,
  "spriteSourceSize": {"x":0,"y":0,"w":252,"h":125},
  "sourceSize": {"w":252,"h":125}
  },
  {
  "filename": "tyrannosaur2_2.png",
  "frame": {"x":252,"y":0,"w":240,"h":127},
  "rotated": false,
  "trimmed": false,
  "spriteSourceSize": {"x":0,"y":0,"w":240,"h":127},
  "sourceSize": {"w":240,"h":127}
  },
  {
  "filename": "tyrannosaur2_3.png",
  "frame": {"x":0,"y":127,"w":244,"h":132},
  "rotated": false,
  "trimmed": false,
  "spriteSourceSize": {"x":0,"y":0,"w":244,"h":132},
  "sourceSize": {"w":244,"h":132}
  },
  {
  "filename": "tyrannosaur2_4.png",
  "frame": {"x":244,"y":127,"w":238,"h":130},
  "rotated": false,
  "trimmed": false,
  "spriteSourceSize": {"x":0,"y":0,"w":238,"h":130},
  "sourceSize": {"w":238,"h":130}
  }],
  "meta": {
    "app": "https://www.codeandweb.com/texturepacker",
    "version": "1.0",
    "image": "spritesheet_t_rex.png",
    "format": "RGBA8888",
    "size": {"w":492,"h":259},
    "scale": "1",
    "smartupdate": "$TexturePacker:SmartUpdate:
    f8d3d463f5d074a4973ecb99bfcd7380:767090008690d487ee5bb42ca248c362:5286
    d6ab01f3e20baf75644d41b412d4$"
  }
}
```

Excerto de Código 3: **Mapeamento de Coordenadas para das Imagens da *Spritesheet* da Figura 30**

```
spritesheet_t_rex = new Image();
spritesheet_t_rex.src = "spritesheet_t_rex.png";

CanvasRenderingContext2D.drawImage(
  spritesheet_t_rex, // Objeto \textit{Image JavaScript} com a imagem "
    spritesheet_t_rex.png"
  0, // Coord X Sprite na Spritesheet da Figura ~\ref{fig:spritesheet_criada}
  127,  // Coord Y Sprite na Spritesheet da Figura ~\ref{fig:
    spritesheet_criada}
  244,  // Largura do Sprite na Spritesheet da Figura ~\ref{fig:
    spritesheet_criada}
  132,   // Altura do Sprite na Spritesheet da Figura ~\ref{fig:
    spritesheet_criada}
  100, // Coord X Ecra
  100, // Coord Y Ecra
  244, // Largura do Sprite na Spritesheet da Figura ~\ref{fig:
    spritesheet_criada}
  132  // Largura do Sprite na Spritesheet da Figura ~\ref{fig:
    spritesheet_criada}
);
```

Excerto de Código 4: **Exemplo de Invocação do Método *CanvasRenderingContext2D.drawImage()* para Desenhar a Primeira Imagem da Segunda Fila da *Spritesheet* da Figura 30**

A utilização da ferramenta *online GetSpriteXY* pode ser observada na Figura 31 e consiste em selecionar individualmente a imagem pretendida, neste caso, do dinossauro, e a ferramenta preencherá os menus do lado direito com as coordenadas da imagem selecionada. Esta ferramenta fornece ainda o *Markup CSS* que constitui a segunda forma de acesso à imagem selecionada na *spritesheet*.

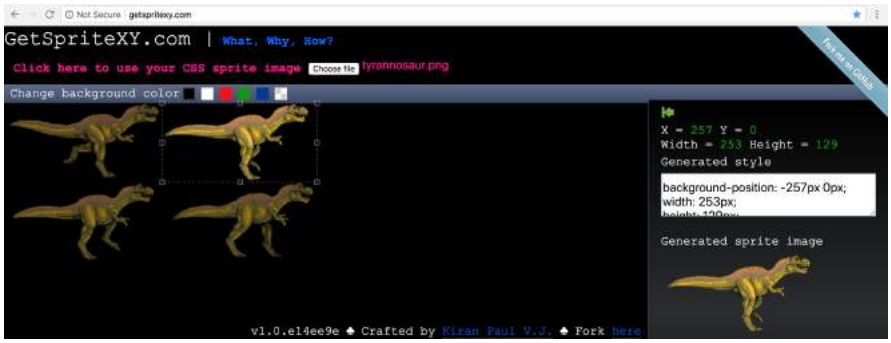


Figura 31.: **Mapeamento de Coordenadas para uma das Imagens da *Spritesheet* da Figura 30**

As ferramentas *Spritegen*[10] e *Spritecow*[11] são dois exemplos que criam *spritesheets* e que fornecem o mapeamento das imagens via *Markup CSS*.

3.4 SÍNTESE

Este capítulo introduziu o problema, que esta dissertação pretende resolver, dentro do *toolkit PVSio-web*, e apresentou a arquitetura da solução proposta e as tecnologias que serão utilizadas na implementação da solução proposta.

10 `http://css.spritegen.com/` (*Spritegen*)
11 `http://www.spritecow.com/` (*Spritecow*)

4

IMPLEMENTAÇÃO DO SIMULADOR DE CONDUÇÃO

Neste capítulo são abordados os *widgets* utilizados para desenvolver a solução proposta. A aparência visual e outros aspectos relacionados com as respectivas interfaces dos *widgets* são definidos por um conjunto de configurações fornecidas, nos respectivos campos opcionais de cada *widget*, nas suas instanciações.

Os *widgets* existentes que permitem construir uma aparência semelhante ao painel de instrumentos do caso de estudo são os *widgets Gauge*, *Speedometer* e *Tachometer* (cor verde na Figura 32). Para mais informações relacionadas com a criação de diferentes *designs* de indicadores circulares que compõem um painel de instrumentos pode consultar a dissertação que explica o processo de criação dessa biblioteca [14].

A Figura 32 apresenta a arquitetura final da solução proposta. A arquitetura possui quatro *widgets* extra, que dizem respeito aos *widgets Sound*, *GyroscopeController*, *DrawGamepad* e *Customization* (cor cinza na Figura 32). O *widget Sound* surge da necessidade de existir um componente que permita reproduzir ficheiros de áudio durante uma qualquer simulação interativa em *PVSio-web*. O *widget GyroscopeController* surge da necessidade de permitir a utilização do simulador de condução em dispositivos móveis, cujos movimentos são detectados pelo sensor giroscópio. O *widget Customization* surge da necessidade de simplificar a utilização do simulador de condução a utilizadores com poucos conhecimentos de programação. O *widget DrawGamepad* surge da necessidade de criar um controlador externo virtual para controlar o veículo no simulador de condução em dispositivos móveis que não possuam sensor giroscópio. O *widget LincolnMKCDashboard* é uma especificação do *widget Dashboard* apresentado na secção 3.2 do capítulo 3 para o painel de instrumentos do caso de estudo.

O maior desafio na implementação do simulador de condução passou pela compreensão do processo de especificação formal das respostas e pela alteração das interfaces dos *widgets* face às ações realizadas pelo utilizador.

Os *widgets* foram desenvolvidos em *JavaScript* como módulos, cujas implementações estão assentes em *prototypes JavaScript*, que são mais tarde exportados. Este processo permite que cada um desses módulos seja facilmente requerido por outras bibliotecas, através de *RequireJS*, e permite instanciar mais que um *widget* facilmente sem ocorrer problemas re-

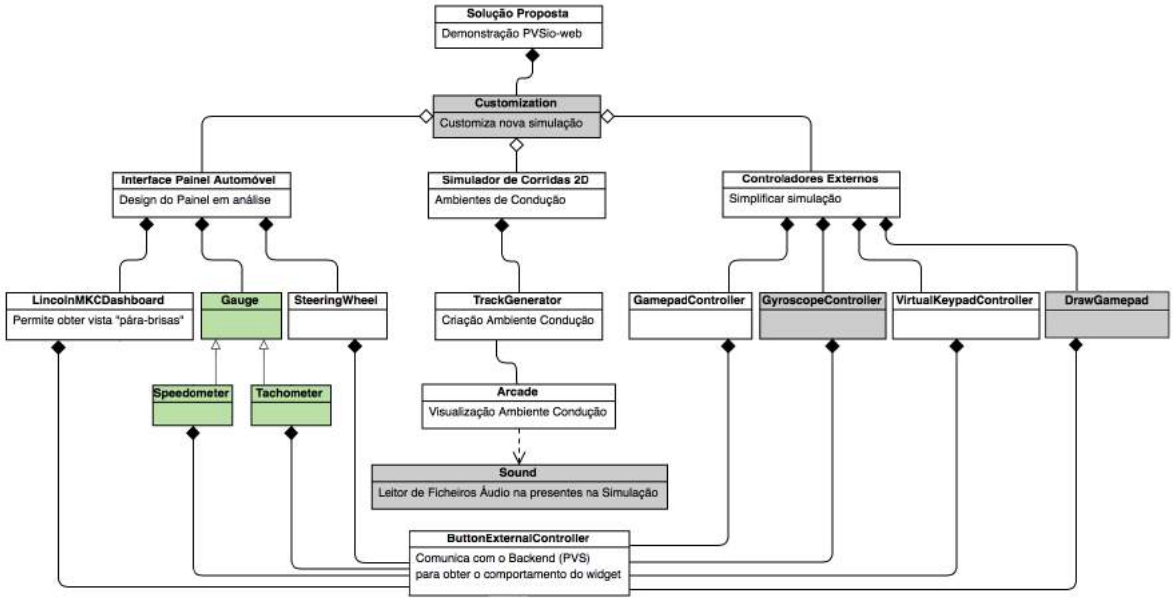


Figura 32.: **Arquitetura da Solução Desenvolvida**

lacionados com variáveis partilhadas. As APIs de todos os *widgets* foram desenvolvidas por forma a devolverem sempre o último estado do *widget* para que seja possível realizar um encadeamento de invocações. Por exemplo, este processo permite realizar o encadeamento *w.a().b()* que corresponde à invocação dos métodos *a()* e *b()* de um *widget w*. A documentação, em inglês, detalhada de todos os *widgets* pode ser consultada *online*[1].

4.1 DESCRIÇÃO DE UM WIDGET PVSIO-WEB GENÉRICO

Para a criação de um protótipo será necessário instanciar o construtor de cada um dos *widgets* que o compõem com três argumentos. O primeiro argumento é o identificador do *widget* utilizado para identificar o *Markup HTML/CSS* criado. O segundo argumento são as coordenadas onde o *widget* será colocado. E o terceiro argumento são os campos opcionais que permitem configurar o *widget*. A definição do construtor é sempre realizada com esta estrutura para todos os *widgets* da plataforma e foi escolhida pelos próprios desenvolvedores de *PVSio-web*. Para a implementação de *widgets* que utilizem imagens recorreu-se à tecnologia *Scalable Vector Graphics (SVG)* [2], que fornece um conjunto de funções de manipulação de imagens.

Os campos opcionais de cada *widget* são um conjunto de parâmetros que permitem configurar as interfaces que serão criadas. A especificação formal *PVS* possui um conjunto de métodos responsáveis por determinar o comportamento dos *widgets* em resposta a uma

1 `https://zecarlos94.github.io/pvsio_web_2D_driving_simulator/car_docs/` (*Documentação*)

dada interação do utilizador e possui o estado da simulação interativa do protótipo. As interfaces criadas irão ser modificadas com base em valores e atributos do estado e com base em alguns dos métodos dessa especificação formal. Cada um dos *widgets* desenvolvidos irá possuir um conjunto de campos opcionais diferentes que irá permitir a criação de diferentes interfaces com diferentes propósitos. O Excerto de Código 5 apresenta a instanciação de um construtor exemplo de um *widget* genérico de *PVSio-web* com um campo opcional preenchido. A instanciação de um qualquer *widget* de *PVSio-web* irá ser igual à instanciação apresentada nesse excerto, sendo que o conteúdo dos argumentos será preenchido de acordo com o *widget* a instanciar.

```
define(function (require, exports, module) {
  "use strict";
  // Require the Generic Widget module
  require("widgets/car/Generic");
  function main() {
    // After Generic Widget module was loaded, initialize it
    let g = new Generic(
      "generic_widget_example",  // id of the Generic Widget HTML element that
     will be created
      {top: 220, left: 20, width: 600, height: 600},  // coordinates object
      { callback: onMessageReceived, // callback is the optional field that
    has the callback function that parses PVS state on client side (in index.
    js file of the PVSio-web prototype)
          .... // add more optional fields according to the Widget
      }
    );
    // After Generic Widget module was initialized, invoke its methods
    // Renders Generic Widget
    g.render();
  }
});
```

Excerto de Código 5: **Instanciação de um *Widget* Genérico de *PVSio-web***

Um *widget PVSio-web* irá possuir uma API comum que diz respeito à manipulação do *Markup HTML/CSS* criado nas suas instanciações e à interpretação do estado *PVS* fornecido pela especificação formal. A API comum estabelecida por *PVSio-web* contém métodos que permitem esconder e visualizar o *widget* rapidamente e que permitem utilizar o estado da especificação formal mais recente. Os métodos comuns são *hide()*, *reveal()*, *show()*, *render()* e *render(res)*. Um *widget* irá também possuir um conjunto de métodos específicos que o diferencia.

O método *hide()* permite esconder o *widget*, no protótipo durante a simulação interativa, ao modificar a propriedade *display* ou *visibility* do *Markup HTML/CSS* para *none* ou *hidden*. O método *reveal()* permite visualizar o *widget*, no protótipo durante a simulação interativa,

Figura 33.: **Tecla com a Seta Norte de um Teclado Apple (lado esquerdo) e Renderização do *Widget ButtonExternalController* com a Seta Norte (lado direito)**

ao modificar a propriedade *display* ou *visibility* do *Markup HTML/CSS* para *block* ou *visible*. O método *show()* permite visualizar o *Markup HTML/CSS* do *widget*, no protótipo durante a simulação interativa, ao devolver a *div* pai do *widget*. Esta *div* é criada na instanciação do seu construtor e é atualizada sempre que forem invocados métodos de manipulação de *Markup HTML/CSS*. O método *render()* permite visualizar o *widget*. Na maior parte dos *widgets* este método apenas invoca o método *reveal()* do próprio *widget*. Existe uma variação deste método que é *render(res)*, que recebe o estado *PVS* fornecido pelo servidor. Este estado irá servir para o *widget* reagir e modificar a interface que criou com base no conteúdo desse estado, que é alterado com base nas interações do utilizador na simulação interativa. O Excerto de Código 5 apresenta também a utilização de um dos métodos comuns supracitados, *render()*, do *widget* genérico instanciado. De seguida serão apresentados mais informações sobre cada um dos *widgets* desenvolvidos.

4.2 Widgets de Painéis de Instrumentos

Nesta secção apresentam-se os *widgets* desenvolvidos com o intuito de criar painéis de instrumentos mais completos em comparação com os que a plataforma permite criar atualmente.

4.2.1 *ButtonExternalController*

O *widget ButtonExternalController* é uma extensão do *widget Button*, cuja diferença assenta na criação de botões interativos com *icons* da biblioteca *jQuery UI* em oposição à criação de botões interativos sem *icons*. O comportamento deste *widget* em resposta a eventos do tipo *"click"* ou *"press/release"* encontra-se especificado formalmente num modelo *PVS* e associado a uma tecla física do teclado do utilizador. Este *widget* permite, por exemplo, criar um botão interativo, lado direito na Figura 33, que recria a interface da tecla de seta para cima física presente no lado esquerdo na Figura 33.

No simulador de condução o *widget ButtonExternalController* é utilizado para reproduzir um conjunto de botões interativos que compõem um teclado virtual de setas, *widget Virtu-*

alKeypadController, e um conjunto de botões interativos que permitem realizar as ações de *pause*, *resume* e *quit* do simulador de ambientes de condução (*widget Arcade*). No entanto, este *widget* pode ser utilizado para construir um qualquer protótipo que necessite de botões interativos com *icons* no seu *background*. Atualmente os *icons* disponíveis pertencem à biblioteca *jQuery UI*, contudo facilmente poderão ser utilizadas outras coleções de *icons*, como *Material UI*, *Glyphicons* ou *Font Awesome*. Algumas destas bibliotecas apenas oferecem gratuitamente uma pequena coleção de *icons* mais simples.

Os campos opcionais mais relevantes fornecidos como terceiro argumento do construtor deste *widget* que o distingue dos restantes *widgets* apresentam-se de seguida

1. ***keyCode*** define a tecla física associada ao novo botão interativo criado por este *widget*.
2. ***callback*** define a função *callback* que é responsável por realizar o *parsing* do último estado formal *PVS*.
3. ***area*** define o *id* da *div* (*Markup HTML*) dedicada ao novo botão interativo.
4. ***parent*** define o *id* da *div* pai, onde a *div* anterior será *appended*.
5. ***buttonClass*** define a utilização da *tag* <*button*> para a *div* dedicada ao novo botão interativo.
6. ***icon*** define o atributo que permite declarar que este botão interativo possuirá um *icon jQuery UI* no *background*.
7. ***evts*** define o tipo de evento associado ao novo botão interativo. Para um botão que poderá ser clicado o valor que toma é *"click"* e para um botão que poderá ser pressionado e libertado o valor que toma é *"press/release"*.

A apresentação de todos os campos opcionais e os seus valores por omissão podem ser consultados na Figura 34.

Este *widget* possui também um conjunto de métodos específicos que confere a funcionalidade desejada ao *widget*. O método *click()* permite invocar a especificação formal definida no modelo *PVS* do botão interativo com o seguinte formato *"click_"+id*, onde *id* é o identificador deste *widget*. Os métodos *press()* e *release()* permitem invocar as especificações formais definidas no modelo formal *PVS* do botão interativo, respectivamente, com o formato *"press_"+id* e *"release_"+id*. O método *pressAndHold()* permite simular uma ação contínua de pressionar o botão interativo, invocando a respectiva especificação formal definida no modelo *PVS* da mesma forma que os métodos anteriores.

Por exemplo, caso o identificador do botão interativo seja *"accelerate"*, as assinaturas dos métodos que terão de ser implementados, no modelo formal *PVS*, para controlar o seu comportamento, apresentam-se no Excerto de Código 6.

constructor(id, coords, opt) → {ButtonExternalController}

Constructor for the ButtonExternalController widget.

Parameters:

Name	Type	Description
id	String	The id of the widget instance.
coords	Object	The four coordinates (top, left, width, height) of the display, specifying the left, top corner, and the width and height of the (rectangular) display. Default is { top: 1000, left: 100, width: 500, height: 500 }.
opt	Object	Options: *Properties* Name / Type / Attributes / Description: axisX — String — <optional> — Value of left axis of PS4 gamepad. axisY — String — <optional> — Value of right axis of PS4 gamepad. buttonClass — String — <optional> — Constant that allows this widget to use 'button' tags instead of 'area' when defining buttons. arrowName — String — <optional> — allows the visual appearance as on the real keyboard layout, i.e. 3 arrow keys aligned at the bottom and 1 arrow key in the middle above them. That is, when this field is not empty, it means that it is an up arrow key (default is ""). title — String — <optional> — Title of Gamepad's Buttons. icon — String — <optional> — JQuery-UI icon class to define VirtualKeypadController widget buttons, i.e. up, left, down, right arrows.

Figura 34.: **Construtor do *Widget ButtonExternalController***

```
press_accelerate(st: state)
release_accelerate(st: state)
```

Excerto de Código 6: **Assinaturas dos Métodos a Implementar na Teoria *PVS* para Controlar o *Widget ButtonExternalController***

4.2.2 *SteeringWheel*

O *widget SteeringWheel* introduz um mecanismo de simulação de volantes, no contexto de *widgets* na plataforma *PVSio-web*. Este *widget* cria uma interface composta por uma imagem de um volante, presente no painel de instrumentos de qualquer veículo, cujo comportamento seja definido por um estado *PVS*. O utilizador quando interage com esta interface observa que a mesma apenas executa movimentos de rotação tal como se verifica nos volantes dos sistemas físicos.

Para simplificar o processo de definição da imagem do volante que este *widget* utilizará, todas as imagens deverão adicionar *"_steering_wheel"* como sufixo e deverão ter como prefixo apenas uma palavra que definirá o estilo do volante. Será, então, este prefixo que o utilizador fornecerá ao construtor do *widget* para que este utilize a imagem desejada. Por exemplo, a imagem de um volante, de um veículo, da marca *Ferrari* deverá possuir o nome

Figura 35.: **Renderização do *Widget SteeringWheel* (lado esquerdo: Estilo *Sparco*; lado direito: Estilo *Ferrari*)**

"ferrari_steering_wheel.svg", por forma a que seja possível definir na instanciação do *widget* o estilo *"ferrari"*, cujo resultado se encontra na imagem no lado direito na Figura 35.

Este *widget* cria três botões lógicos normais que associam um comportamento do volante a uma tecla física do computador. Esses comportamentos são *left*, *right* e *straight* que, por sua vez, implicam a definição da especificação formal com o seguinte formato *id*+"_comportamento". No simulador de condução, este *widget* é utilizado para simular o comportamento do volante em simultâneo com as interfaces que compõem o painel de instrumentos completo de um veículo, que serão renderizadas por outros *widgets*. Por omissão esta dissertação fornece cinco estilos diferentes um volante básico, um volante da marca *Ferrari*, um volante da marca *Porsche*, um volante da marca *Lincoln* (volante do caso de estudo) e um volante da marca *Sparco* (marca de desportos motorizados, como o *World Rally Championship (WRC)*).

Os campos opcionais mais relevantes fornecidos como terceiro argumento do construtor deste *widget* apresentam-se de seguida

1. ***style*** define a imagem do volante a renderizar. Por exemplo, o estilo *"sparco"* implica que a interface seja renderizada com a imagem do volante cujo nome do ficheiro é *"sparco_steering_wheel.svg"*, cujo resultado pode ser observado na imagem no lado esquerdo na Figura 35.

2. ***z-index*** define o *layer* (camada) onde a imagem será colocada no protótipo, por forma a permitir a sobreposição da imagem do volante em relação a outras imagens.

A apresentação de todos os campos opcionais e os seus valores por omissão podem ser consultados na Figura 36.

Este *widget* possui também um conjunto de métodos específicos que confere a funcionalidade desejada ao *widget*. O método *rotate(val)* permite rodar logicamente o volante, cujo ângulo de rotação é *val*. Este método invoca a especificação formal, no modelo *PVS*, com o formato *id+"_rotate"*, onde *id* é o identificador do *widget*, como se observa no Excerto de Código 7. O método *render(val)* permite rodar visualmente o volante, cujo ângulo de

`constructor(id, coords, opt)` → {SteeringWheel}

Constructor for the SteeringWheel widget.

Parameters:

Name	Type	Description
id	String	The id of the widget instance.
coords	Object	The four coordinates (top, left, width, height) of the display, specifying the left, top corner, and the width and height of the (rectangular) display. Default is { top: 0, left: 0, width: 250, height: 250 }.
opt	Object	Options: *Properties* Name / Type / Attributes / Description parent — String — <optional> — the HTML element where the display will be appended (default is "body"). style — String — <optional> — a valid style identifier, i.e., valid SVG filename (default is "ferrari"). z-index — String — <optional> — value for the CSS property z-index (if not provided, no z-index is applied). imagePath — String — <optional> — the image path with steering wheel to load. (default is "../../client/app/widgets/car/steering_wheels/").

Figura 36.: **Construtor do *Widget SteeringWheel***

rotação é *val*. Este método é um caso particular do método genérico que em vez de receber o estado *PVS* total, recebe apenas o conteúdo do atributo desse estado que está relacionado com o ângulo de rotação do *widget*. Para que esta rotação se observe visualmente este método invoca o método *rotate()*, da tecnologia SVG na propriedade *CSS transform* no *Markup HTML*, com o valor *val*, o que permite rodar a imagem no protótipo.

Caso *id* do *widget* for *"steering_wheel"*, as assinaturas dos métodos que terão de ser implementados, no modelo formal *PVS*, para controlar o comportamento do volante, apresentam-se no Excerto de Código 7.

```
steering_wheel_right(st: state)
steering_wheel_left(st: state)
steering_wheel_straight(st: state)
steering_wheel_rotate(x: real)(st: state)
```

Excerto de Código 7: **Assinaturas dos Métodos a Implementar na Teoria *PVS* para Controlar o *Widget SteeringWheel***

4.2.3 *LincolnMKCDashboard*

O *widget LincolnMKCDashboard* introduz um mecanismo de simulação dos painéis de instrumentos completos, cuja imagem a utilizar não deverá ter o volante, pois esse será adi-

cionado através de um *widget* apropriado. Este *widget* também define dois botões lógicos para simular os botões *Start/Stop* e *S* do veículo do caso de estudo. O posicionamento desses botões lógicos foi realizado manualmente para as imagens com a interface antes e após a correção. Para simplificar o processo de seleção da imagem do painel de instrumentos completo cada imagem deverá possuir o nome com o seguinte formato *2015_lincoln_mkc_+design+_recall_+dashIndex+.svg*, onde *design* pode tomar um de dois valores possíveis, *"before"* ou *"after"*, que permite definir se o painel de instrumentos escolhido para a simulação é a versão com a falha reportada, ou se é a versão com a interface corrigida, e onde *dashIndex* é um número colocado como sufixo na imagem, que permite ter várias imagens diferentes de painéis de instrumentos das versões *"before"* e *"after"* supracitadas. Apesar do formato acima exigir que a primeira parte seja *2015_lincoln_mkc_* é possível que a imagem possua um painel de instrumentos de outro veículo.

No futuro, este *widget* poderá ser melhorado por forma a que seja mais transparente o processo de definição do nome das imagens dos painéis de instrumentos a renderizar e de definição dos botões lógicos, que têm de ser mapeados manualmente de acordo com a imagem que se pretende fornecer ao *widget*.

No simulador de condução o *widget LincolnMKCDashboard* é utilizado para visualizar o painel de instrumentos do veículo do caso de estudo apresentado na secção 1.1. no capítulo 1. Esta visualização tem em consideração o conceito de vista "pára-brisas" por forma a que a interface criada com a imagem do painel de instrumentos completo em conjunto com o simulador de condução forneça a perspectiva de que o utilizador se encontra atrás do volante com os ambientes de condução à sua frente e ao seu redor. O utilizador interage com este *widget* ao clicar no botão lógico *Start/Stop* (botão com uma luz verde) ou ao clicar no botão lógico *S*.

Então, este *widget* cria dois botões lógicos, *startAndStop* e *activateSportMode*, cujos comportamentos se encontram definidos no modelo formal *PVS* com o seguinte formato *"press_-"*+"comportamento". Por exemplo, para o comportamento *startAndStop* é *"press_startAndStop"* e *"release_startAndStop"*. Para o segundo comportamento o processo é igual. A Figura 37 apresenta a imagem do painel de instrumentos completo. A imagem no lado esquerdo na Figura 38 foca a consola central da interface com a falha reportada. A imagem no lado direito na Figura 38 foca a consola central da interface corrigida.

Os campos opcionais mais relevantes fornecidos como terceiro argumento do construtor deste *widget* apresentam-se de seguida

1. ***design*** define o nome da imagem do painel de instrumentos completo a renderizar.
2. ***dashIndex*** define o sufixo numérico da imagem do painel de instrumentos completo a renderizar, caso exista.

Figura 37.: **Renderização do *Widget LincolnMKCDashboard***

Figura 38.: **Renderização do *Widget LincolnMKCDashboard* - Consola Central em Destaque (lado esquerdo: Antes da Recolha do Fabricante; lado direito: Após a Recolha do Fabricante)**

`constructor(id, coords, opt)` → {LincolnMKCDashboard}

Constructor for the LincolnMKCDashboard widget.

Parameters:

Name	Type	Description
id	String	The id of the widget instance.
coords	Object	The four coordinates (top, left, width, height) of the display, specifying the left, top corner, and the width and height of the (rectangular) display. Default is { top: 0, left: 0, width: 250, height: 250 }.
opt	Object	Options: *Properties* Name / Type / Attributes / Description

Name	Type	Attributes	Description
parent	String	<optional>	the HTML element where the display will be appended (default is "body").
dashIndex	Bool	<optional>	is the index of full dashboard placed as a suffix in the filename (default is 1).
design	String	<optional>	the image with lincolnMKCDashboard to load. Only has 2 values: 'before', i.e. the design before General Motors recalled the vehicles, and 'after', i.e. the design after General Motors recalled the vehicles (default is "before").
buttonsPVS	Array	<optional>	the actions defined in main.pvs file, which are used to define all Lincoln MKC Dashboard buttons, ButtonEVO, ids (default is ["startAndStop", "activateSportMode"]).
imagePath	String	<optional>	the image path with lincolnMKCDashboard to load. (default is "../../client/app/widgets/car/lincoln_mkc_dashboard_case_study/").

Figura 39.: **Construtor do *Widget LincolnMKCDashboard***

3. ***buttonsPVS*** define o nome das funções, especificadas formalmente, no modelo *PVS*, que serão associadas, sequencialmente, aos botões lógicos criados para o painel de instrumentos escolhido.

A apresentação de todos os campos opcionais e os seus valores por omissão podem ser consultados na Figura 39.

Este *widget* possui também um conjunto de métodos específicos que confere a funcionalidade desejada ao *widget*. O método *callClickPVS(buttonName)* permite invocar a especificação formal associada ao botão lógico cujo identificador é dado pelo argumento *buttonName* e cujo evento é do tipo *"click"*. O método *callPressReleasePVS(buttonName)* permite invocar a especificação formal associada ao botão lógico cujo identificador é dado pelo argumento *buttonName* e cujo evento é do tipo *"press/release"*.

Caso o utilizador forneça a lista *["startAndStop", "activateSportMode"]* no campo opcional *buttonsPVS*, as assinaturas dos métodos que terão de ser implementados, no modelo formal *PVS*, para controlar o comportamento dos botões do painel de instrumentos, apresentam-se no Excerto de Código 8.

Figura 40.: **Renderização do *Widget DrawGamepad* (lado esquerdo: estilo *PS4*; lado direito: *XBOX*)**

```
press_startAndStop(st: state)
release_startAndStop(st: state)
press_activateSportMode(st: state)
release_activateSportMode(st: state)
```

Excerto de Código 8: **Assinaturas dos Métodos a Implementar na Teoria *PVS* para Controlar o *Widget LincolnMKCDashboard***

4.3 WIDGETS DE CONTROLO

Nesta secção apresentam-se os *widgets* desenvolvidos com o intuito de controlar os protótipos construídos com *widgets* de painéis de instrumentos e de animação (simulador de condução). O controlo dos protótipos tem em consideração os diferentes dispositivos que poderão ser utilizados para os simular. Os dispositivos podem ser móveis ou *desktop* e podem ter ou não sensor giroscópio e entradas *USB*.

4.3.1 *DrawGamepad*

O *widget DrawGamepad* introduz um mecanismo de simulação de controladores externos virtuais (estilos). Este *widget* requer o mapeamento manual de cada botão físico (dispositivo real) para um botão lógico (*widget ButtonExternalController*). O utilizador não irá configurar o posicionamento dos botões lógicos, mas sim configurar o estilo a utilizar e os identificadores dos botões lógicos desse estilo, por forma a coincidir com os métodos implementados na especificação formal que determina os comportamentos desse controlador externo virtual. O posicionamento foi realizado previamente na criação dos estilos disponíveis.

Neste momento estão disponíveis dois controladores diferentes, comando *PlayStation 4* e o comando *XBOX One*, que são os comandos mais utilizados no mercado atual de jogos[2]. Cada controlador externo físico possui um índice associado a cada botão físico de acordo com o mapeamento aplicado pelo seu fabricante. Por exemplo, no controlador externo

2 https://www.pcgamer.com/best-controller-for-pc-gaming/ (*The best controller for PC gaming*)

físico da *PlayStation 4* o botão *"cross"* tem o índice zero. Então, optou-se por realizar a configuração dos identificadores dos botões lógicos através de uma lista, tendo por base esses índices. O índice zero da lista corresponde ao identificador do botão lógico *"cross"*. Para os restantes botões o processo é igual e todos esses são definidos com eventos *"press/release"*, exceto para os *sticks*, que são definidos com eventos *"click"*, para diferenciar o comportamento destes botões, já que no sistemas reais estes também se comportam de forma diferente.

No simulador de condução o *widget DrawGamepad* é utilizado para simular o comportamento de um controlador externo que permite controlar a direção do veículo no simulador de ambientes de condução e controlar o comportamento dos *widgets Speedometer*, *Tachometer* e *SteeringWheel*.

Os campos opcionais mais relevantes fornecidos como terceiro argumento do construtor deste *widget* que o distingue dos restantes *widgets* apresentam-se de seguida

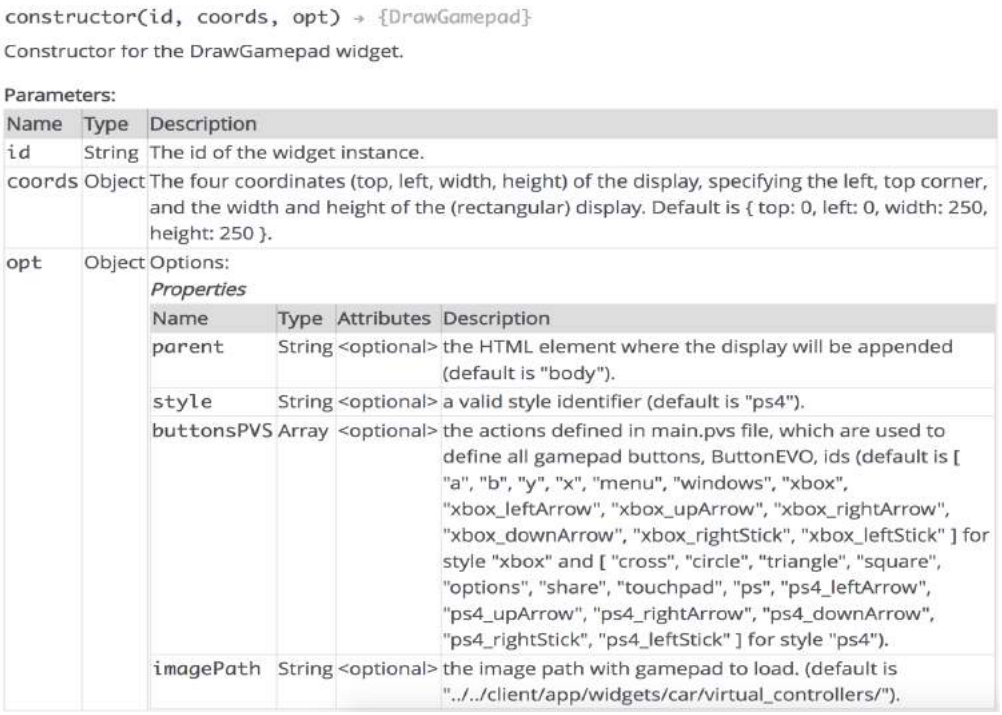
1. ***style*** define a imagem do controlador externo a renderizar. Existem dois valores possíveis, *"xbox"* e *"ps4"*, respectivamente, para o controlador externo virtual *XBOX One*, lado direito na Figura 40 e *PlayStation 4*, lado esquerdo na Figura 40.

2. ***buttonsPVS*** define a lista com o nome das funções definidas na especificação formal (modelo *PVS*) associadas a cada um dos botões lógicos do controlador externo a renderizar.

A apresentação de todos os campos opcionais e os seus valores por omissão podem ser consultados na Figura 41.

Este *widget* possui também um conjunto de métodos específicos que confere a funcionalidade desejada ao *widget*. O método *callClickPVS(buttonName)* permite invocar a especificação formal associada ao botão lógico cujo identificador é dado pelo argumento *buttonName* e cujo evento é do tipo *"click"*. O método *callPressReleasePVS(buttonName)* permite invocar a especificação formal associada ao botão lógico cujo identificador é dado pelo argumento *buttonName* e cujo evento é do tipo *"press/release"*.

Por exemplo, caso a lista dos métodos da especificação formal, fornecida no campo opcional *buttonsPVS*, seja *["cross", "circle", "triangle", "square", "options", "share", "touchpad", "ps", "ps4_leftArrow", "ps4_upArrow", "ps4_rightArrow", "ps4_downArrow", "ps4_rightStick", "ps4_leftStick"]*, as assinaturas dos métodos que terão de ser implementados, no modelo formal *PVS*, para controlar o comportamento do controlador externo virtual, apresentam-se no Excerto de Código 9. Caso o utilizador não pretenda fornecer a lista com os identificadores dos botões lógicos do estilo escolhido, o *widget DrawGamepad* possui uma lista por omissão para cada um dos dois estilos disponíveis. Por omissão, a lista *["cross", "circle", "triangle", "square", "options", "share", "touchpad", "ps", "ps4_leftArrow", "ps4_upArrow", "ps4_rightArrow", "ps4_downArrow", "ps4_rightStick", "ps4_leftStick"]* contém os nomes dos



`constructor(id, coords, opt)` → {DrawGamepad}

Constructor for the DrawGamepad widget.

Parameters:

Name	Type	Description
id	String	The id of the widget instance.
coords	Object	The four coordinates (top, left, width, height) of the display, specifying the left, top corner, and the width and height of the (rectangular) display. Default is { top: 0, left: 0, width: 250, height: 250 }.
opt	Object	Options: *Properties* Name / Type / Attributes / Description

Name	Type	Attributes	Description
parent	String	<optional>	the HTML element where the display will be appended (default is "body").
style	String	<optional>	a valid style identifier (default is "ps4").
buttonsPVS	Array	<optional>	the actions defined in main.pvs file, which are used to define all gamepad buttons, ButtonEVO, ids (default is ["a", "b", "y", "x", "menu", "windows", "xbox", "xbox_leftArrow", "xbox_upArrow", "xbox_rightArrow", "xbox_downArrow", "xbox_rightStick", "xbox_leftStick"] for style "xbox" and ["cross", "circle", "triangle", "square", "options", "share", "touchpad", "ps", "ps4_leftArrow", "ps4_upArrow", "ps4_rightArrow", "ps4_downArrow", "ps4_rightStick", "ps4_leftStick"] for style "ps4").
imagePath	String	<optional>	the image path with gamepad to load. (default is "../../client/app/widgets/car/virtual_controllers/").

Figura 41.: **Construtor do *Widget DrawGamepad***

métodos para o controlador externo virtual *PlayStation 4* e a lista *["a", "b", "y", "x", "menu", "windows", "xbox", "xbox_leftArrow", "xbox_upArrow", "xbox_rightArrow", "xbox_downArrow", "xbox_rightStick", "xbox_leftStick"]* contém os nomes dos métodos para o controlador externo virtual *XBOX One*.

```
press_cross(st: state)
release_cross(st: state)
press_circle(st: state)
release_circle(st: state)
press_triangle(st: state)
release_triangle(st: state)
press_square(st: state)
release_square(st: state)
press_options(st: state)
release_options(st: state)
press_share(st: state)
release_share(st: state)
release_touchpad(st: state)
press_ps(st: state)
release_ps(st: state)
press_ps4_leftArrow(st: state)
release_ps4_leftArrow(st: state)
press_ps4_upArrow(st: state)
release_ps4_upArrow(st: state)
press_ps4_rightArrow(st: state)
release_ps4_rightArrow(st: state)
press_ps4_downArrow(st: state)
release_ps4_downArrow(st: state)
click_ps4_rightStick(st: state)
click_ps4_leftStick(st: state)
```

Excerto de Código 9: **Assinaturas dos Métodos a Implementar na Teoria *PVS* para Controlar o *Widget DrawGamepad***

Figura 42.: **Renderização do *Widget VirtualKeypadController***

O clique do botão lógico *"x"*, na imagem no lado esquerdo na Figura 40, invocará as funções *press_x* e *release_x*, que deverão estar definidas no modelo *PVS*.

4.3.2 *VirtualKeypadController*

O *widget VirtualKeypadController* introduz um mecanismo que cria um teclado de setas virtual, composto por um *icon* teclado e por quatro botões lógicos (*widget ButtonExternalController*) com a mesma disposição das teclas de setas presentes em qualquer teclado físico, acompanhadas de três botões lógicos dedicados às interações com os menus *pause*, *resume* e *quit* do simulador de condução, Figura 42. Os eventos a capturar nesses botões lógicos são do tipo *"press/release"*. Para simular o comportamento dos teclados virtuais presentes em qualquer disposito móvel decidiu-se que os sete botões lógicos apenas são visíveis após o primeiro clique no *icon* teclado. Após o segundo clique passam a estar invisíveis.

A disposição e o tamanho dos botões varia consoante o tipo de aparelho, *mobile* ou *desktop*, e ajusta as posições, a largura e a altura de cada uma das interfaces desses botões lógicos para facilitar o clique em cada um desses botões.

No simulador de condução, o *widget VirtualKeypadController* é utilizado para interagir com o veículo do simulador de condução, com o *widget SteeringWheel* e com os *widgets Speedometer* e *Tachometer* que compõem o protótipo do painel de instrumentos completo do veículo do caso de estudo, e dos seus contextos de condução.

Os campos opcionais mais relevantes fornecidos como terceiro argumento do construtor deste *widget* apresentam-se de seguida

1. ***keyboardLeftMobile*** define a coordenada *left*, num aparelho *mobile*, onde se colocarão as interfaces dos sete botões lógicos, quando a visibilidade tomar o valor "visível"(*visible*).

2. ***keyboardTopMobile*** define a coordenada *top*, num aparelho *mobile*, onde se colocarão as interfaces dos sete botões lógicos, quando a visibilidade tomar o valor "visível"(*visible*).

3. ***keyboardLeftDesktop*** define a coordenada *left*, num aparelho *Desktop*, onde se colocarão as interfaces dos sete botões lógicos, quando a visibilidade tomar o valor "visível"(*visible*).

4. ***keyboardTopDesktop*** define a coordenada *top*, num aparelho *Desktop*, onde se colocarão as interfaces dos sete botões lógicos, quando a visibilidade tomar o valor "visível"(*visible*).

5. ***keyboardHoverInitialTitle*** define o texto que aparecerá por cima da imagem do teclado quando o utilizador realizar um movimento do tipo *hover*, antes do primeiro clique, com o rato sobre essa imagem quando a visibilidade tomar o valor "invisível"(*hidden*).

6. ***keyboardHoverSecondTitle*** permite definir o texto que aparecerá por cima da imagem do teclado quando o utilizador realizar um movimento do tipo *hover*, antes do segundo clique, com o rato sobre essa imagem quando a visibilidade tomar o valor "visível"(*visible*).

7. ***simulatorActions*** define o identificador da *div* que possuirá todo o *Markup HTML* relacionado com as interfaces dos botões dos menus *pause*, *resume* e *quit* do *widget Arcade*.

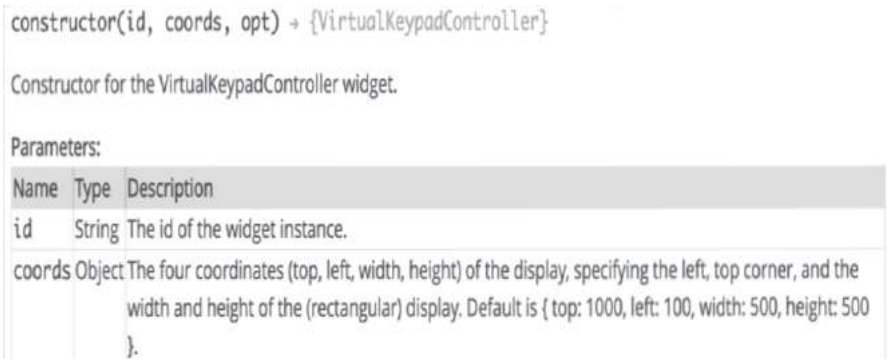
8. ***simulatorArrows*** define o identificador da *div* que possuirá todo o *Markup HTML* relacionado com as interfaces dos botões que compõem o teclado de setas virtual (setas para a esquerda, para a direita, para cima e para baixo).

9. ***floatArrows*** define o identificador da *div* que possuirá todo o *Markup HTML* relacionado com a interface do botão com a seta para cima, uma vez que é a única seta que se encontra no nível superior do teclado de setas físico, que se pretende recriar.

10. ***blockArrows*** define o identificador da *div* que possuirá todo o *Markup HTML* relacionado com as interfaces dos restantes botões que compõem o teclado de setas virtual (setas para a esquerda, para a direita e para baixo), que se encontram no nível inferior do teclado de setas físico, que se pretende recriar.

11. ***buttonClass*** define as classes *jQuery UI* que permitem colocar o *icon* na interface de cada botão lógico criado.

12. ***arrowKeysPVS*** define o nome das funções, especificadas formalmente, no modelo *PVS*, que serão associadas, sequencialmente, aos botões lógicos que correspondem às setas virtuais (setas para cima, para baixo, para a esquerda e para a direita).



constructor(id, coords, opt) → {VirtualKeypadController}

Constructor for the VirtualKeypadController widget.

Parameters:

Name	Type	Description
id	String	The id of the widget instance.
coords	Object	The four coordinates (top, left, width, height) of the display, specifying the left, top corner, and the width and height of the (rectangular) display. Default is { top: 1000, left: 100, width: 500, height: 500 }.

Figura 43.: **Construtor do *Widget VirtualKeypadController* - Parte 1**

13. ***otherKeysPVS*** define o nome das funções, especificadas formalmente, no modelo *PVS*, que serão associadas, sequencialmente, aos botões lógicos dos menus *quit*, *pause* e *resume* do *widget Arcade*.

A apresentação de todos os campos opcionais e os seus valores por omissão podem ser consultados nas Figuras 43 e 44.

Os comportamentos do *widget VirtualKeypadController* são configurados nos campos opcionais *arrowKeysPVS* e *otherKeysPVS*. Nestes campos opcionais o utilizador pode fornecer os identificadores para os sete botões lógicos que compõem o teclado virtual, que correspondem aos identificadores dos métodos da especificação formal. Por omissão, os comportamentos são *steering_wheel_left* e *steering_wheel_right*, *accelerate* e *brake*, *quit*, *pause* e *resume*. Os comportamentos *steering_wheel_left* e *steering_wheel_right* dizem respeito à direção do veículo no simulador de condução e à direção do volante (*widget SteeringWheel*). Os comportamentos *accelerate* e *brake* dizem respeito aos valores da velocidade e das rotações por minuto exibidos no velocímetro (*widget Speedometer*) e no tacómetro (*widget Tachometer*) e que coincidem com os valores da velocidade e das rotações por minuto do veículo no simulador de condução. Os comportamentos *quit*, *pause* e *resume* dizem respeito ao estado da simulação de condução.

Este *widget* apenas possui os métodos da API genérica supracitada. Os nomes das funções, especificadas formalmente no modelo *PVS*, fornecidas no campo opcional *arrowKeysPVS* são atribuídas sequencialmente da seguinte forma, o primeiro nome diz respeito à função que será invocada pelo botão de seta para cima, o segundo nome à função que será invocada pelo botão de seta para baixo, o terceiro nome à função que será invocada pelo botão de seta para esquerda e o quarto nome à função que será invocada pelo botão de seta para direita. Os nomes das funções, especificadas formalmente no modelo *PVS*, fornecidas no campo opcional *otherKeysPVS* são atribuídas sequencialmente da seguinte forma, o primeiro nome diz respeito à função que será invocada pelo botão do menu *quit*, o segundo nome à função que será invocada pelo botão do menu *pause* e o terceiro nome à função que será invocada pelo botão do menu *resume*.

opt Object Options:

Properties

Name	Type	Attributes	Description
buttonsDiv	String	<optional>	id name of the div where to put the virtual buttons image (default is "virtualKeyPad").
keyboardImgDiv	String	<optional>	id name of the div where to put the virtual keyboard image (default is "mobileDevicesController").
keyboardClass	String	<optional>	virtual keyboard div class name (default is "icon keyboard").
keyboardTopMobile	Int	<optional>	virtual keyboard div top position for mobile devices (default is 750).
keyboardLeftMobile	Int	<optional>	virtual keyboard div left position for mobile devices (default is 1350).
keyboardTopDesktop	Int	<optional>	virtual keyboard div top position for desktop devices (default is 735).
keyboardLeftDesktop	Int	<optional>	virtual keyboard div left position for desktop devices (default is 1380).
keyboardUrl	String	<optional>	virtual keyboard image path (default is "widgets/car/configurations/img/keyboard.png").
keyboardHoverInitialTitle	String	<optional>	initial text on hover in virtual keyboard image (default is "Click to open virtual keypad controller").
keyboardHoverSecondTitle	String	<optional>	second text(after click) on hover in virtual keyboard image (default is "Click to close virtual keypad controller").
keyboardOnclickAction	String	<optional>	method to execute after clicking the virtual keyboard image (default is "").
keyboardImageWidthMobile	Int	<optional>	virtual keyboard image width for mobile devices (default is 80).
keyboardImageHeightMobile	Int	<optional>	virtual keyboard image height for mobile devices (default is 60).
keyboardImageWidthDesktop	Int	<optional>	virtual keyboard image width for desktop devices (default is 50).
keyboardImageHeightDesktop	Int	<optional>	virtual keyboard image height for desktop devices (default is 30).
parent	String	<optional>	the HTML element where the display will be appended (default is "body").
simulatorActions	String	<optional>	the HTML element where the action buttons(pause, resume and quit) will be appended (default is "simulatorActions").
simulatorArrows	String	<optional>	the HTML element where the arrow buttons(left, up, right, down) will be appended (default is "simulatorArrows").
floatArrows	String	<optional>	the HTML element where the up arrow button will be appended (default is "floatArrows").
blockArrows	String	<optional>	the HTML element where the arrow buttons(left, right, down) will be appended (default is "blockArrows").
buttonClass	String	<optional>	the constant string that allows ButtonExternalController widget to use 'button' tags with JQuery-UI images instead of areas as Button widget implements.(default is "buttonClass").
title	String	<optional>	the button's title(default is 'title').
arrowKeysPVS	Array	<optional>	array with the IDs of the ButtonExternalController widgets to create, that is, the names of the formal specifications in the main.pvs file for arrow keys (default is ["accelerate", "brake", "steering_wheel_left", "steering_wheel_right"]).
otherKeysPVS	Array	<optional>	array with the IDs of the ButtonExternalController widgets to create, that is, the names of the formal specifications in the main.pvs file for the other buttons (default is ["quit", "pause", "resume"]).

Figura 44.: **Construtor do *Widget VirtualKeypadController* - Parte 2**

Caso o utilizador forneça a lista *["accelerate", "brake", "steering_wheel_left", "steering_wheel_right"]* no campo opcional *arrowKeysPVS* e forneça lista *["quit", "pause", "resume"]* no campo opcional *otherKeysPVS*, as assinaturas dos métodos que terão de ser implementados, no modelo formal *PVS*, para controlar o comportamento do teclado virtual, apresentam-se no Excerto de Código 10.

```
press_accelerate(st: state)
release_accelerate(st: state)
press_brake(st: state)
release_brake(st: state)
steering_wheel_left(st: state)
steering_wheel_right(st: state)
press_quit(st: state)
release_quit(st: state)
press_pause(st: state)
quit_pause(st: state)
press_resume(st: state)
release_resume(st: state)
```

Excerto de Código 10: **Assinaturas dos Métodos a Implementar na Teoria *PVS* para Controlar o *Widget VirtualKeypadController***

4.3.3 *GyroscopeController*

O *Widget GyroscopeController* introduz um mecanismo que permite controlar o simulador de condução, *widget Arcade*, o volante (*widget SteeringWheel*) e os *widgets Speedometer* e *Tachometer* em dispositivos móveis com sensor Giroscópio. O Giroscópio é um mecanismo que permite medir a orientação do aparelho.

A Figura 45 descreve o sistema de coordenadas e os eixos de rotação de um *smartphone* que possui um sensor giroscópio. Este sistema de coordenadas é composto por três eixos, x, y e z, alinhados no centro do *smartphone/tablet* ou computador. A orientação do *smartphone/tablet* baseia-se na orientação do ecrã desses aparelhos. A orientação do computador (*desktop/laptop*) baseia-se em relação ao teclado físico desses aparelhos[4].

Para utilizar o sensor giroscópio é necessário utilizar um *event listener* que fica à escuta de eventos *deviceorientation*, que possuem os valores de *alpha*, *beta* e *gamma*. Esses eventos são lançados sempre que ocorre uma alteração da orientação do aparelho. É necessário

3 https://developers.google.com/web/fundamentals/native-hardware/device-orientation/ (*Device Orientation and Motion*)

4 https://developer.mozilla.org/en-US/docs/Web/API/Detecting_device_orientation (*Detecting device orientation*)

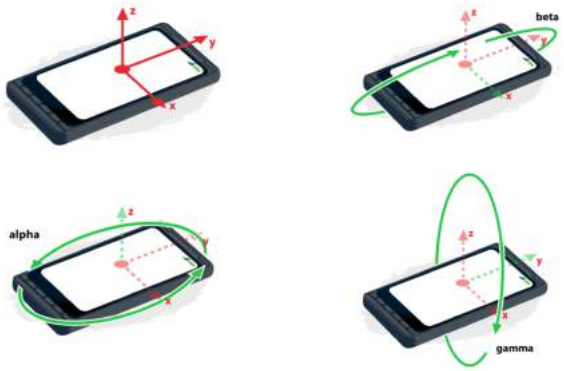


Figura 45.: **Sistema de Coordenadas e Rotação dos Eixos X,Y e Z de um Aparelho que possua Giroscópio**

Fonte: [3]

implementar um método que seja responsável por receber esses eventos e processar os mesmos.

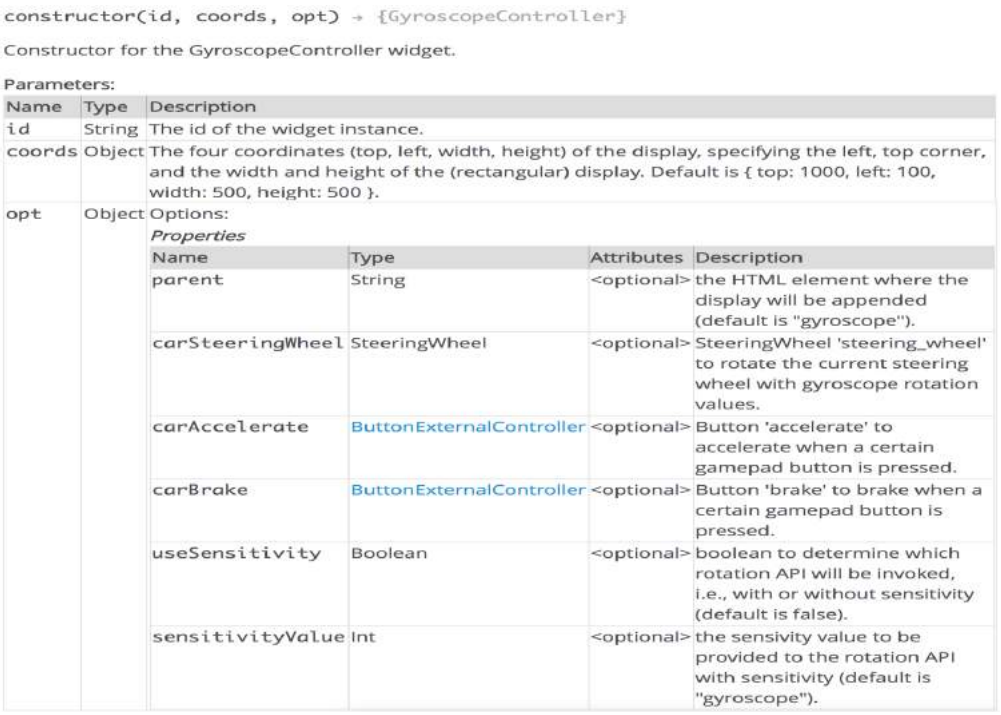
No simulador de condução o *widget GyroscopeController* é utilizado para acelerar e travar, para alterar a direção do veículo no simulador de condução e para rodar o volante (*widget SteeringWheel*) do painel de instrumentos do protótipo. A rotação sobre o eixo z implica a alteração dos valores *alpha*, a rotação sobre o eixo x implica a alteração dos valores *beta* e a rotação sobre o eixo y implica a alteração dos valores *gamma*. Assumindo que o dispositivo está em modo *landscape*, para acelerar e travar será necessário rodar o eixo x e para mudar a direção do veículo será necessário rodar o eixo y, que corresponde a rodar o dispositivo da esquerda para a direita e vice-versa. Para acelerar será necessário rodar o dispositivo de frente para a trás e para travar será necessário rodar o dispositivo de trás para a frente.

Os campos opcionais mais relevantes fornecidos como terceiro argumento do construtor deste *widget* apresentam-se de seguida

1. ***carSteeringWheel*** define o volante a controlar (*widget SteeringWheel*).
2. ***carAccelerate*** define o botão lógico responsável por simular a aceleração no simulador de condução e *widgets Speedometer* e *Tachometer*.
3. ***carBrake*** define o botão lógico responsável por simular a travagem no simulador de condução e *widgets Speedometer* e *Tachometer*.

A apresentação de todos os campos opcionais e os seus valores por omissão podem ser consultados na Figura 46.

Este *widget* possui também um conjunto de métodos específicos e privados de utilização interna. Os métodos *rotateSteeringAngle(x,y)* e *rotateSteeringAngleWithSensitivity(x,y,sensitivity)* permitem utilizar os ângulos calculados com base na orientação do dis-



`constructor(id, coords, opt)` → {GyroscopeController}

Constructor for the GyroscopeController widget.

Parameters:

Name	Type	Description
id	String	The id of the widget instance.
coords	Object	The four coordinates (top, left, width, height) of the display, specifying the left, top corner, and the width and height of the (rectangular) display. Default is { top: 1000, left: 100, width: 500, height: 500 }.
opt	Object	Options: *Properties* Name / Type / Attributes / Description

Name	Type	Attributes	Description
parent	String	<optional>	the HTML element where the display will be appended (default is "gyroscope").
carSteeringWheel	SteeringWheel	<optional>	SteeringWheel 'steering_wheel' to rotate the current steering wheel with gyroscope rotation values.
carAccelerate	ButtonExternalController	<optional>	Button 'accelerate' to accelerate when a certain gamepad button is pressed.
carBrake	ButtonExternalController	<optional>	Button 'brake' to brake when a certain gamepad button is pressed.
useSensitivity	Boolean	<optional>	boolean to determine which rotation API will be invoked, i.e., with or without sensitivity (default is false).
sensitivityValue	Int	<optional>	the sensivity value to be provided to the rotation API with sensitivity (default is "gyroscope").

Figura 46.: **Construtor do *Widget GyroscopeController***

positivo móvel, com sensor giroscópio, para alterarem a direção do veículo no simulador de condução e a direção do volante (*widget SteeringWheel*). O valor *beta* é utilizado para invocar os métodos *press()* e *release()* do botão responsável pela aceleração do veículo (*carAccelerate*), e do botão responsável pela travagem do veículo (*carBrake*). O valor *gamma* é utilizado para rodar o volante (*carSteeringWheel*). Esta rotação poderá ser suavizada com base no valor de sensibilidade fornecido pelo utilizador (campo opcional *sensitivityValue*), caso deseje (campo opcional *useSensitivity*). O método *handleOrientation()* é responsável por interpretar os dados recebidos pelo sensor giroscópio.

4.3.4 `GamepadController`

O *Widget GamepadController* introduz um mecanismo que permite controlar o simulador de condução, o volante (*widget SteeringWheel*) e os *widgets* do painel de instrumentos *Speedometer* e *Tachometer*, cujos valores apresentados dependem da aceleração ou travagem realizados no simulador supracitado, com um controlador externo. Um controlador externo é um qualquer aparelho que possua botões e *joysticks* que pode ser conectado ao computador via *USB* ou via *Bluetooth*. Os controladores externos mais comuns são os comandos das consolas *PlayStation 4*, da *Sony*, e *XBOX One*, da *Microsoft*, lados esquerdo e direito na Figura 26, na página 28. Existem ainda controladores externos menos comuns e mais complexos como o controlador externo *Logitech G29*, da *Logitech*, Figura 27, na página 29.

Os fabricantes não aplicam o mesmo *standard* de mapeamento dos botões e dos *sticks*. O desafio deste *widget* passou pela compreensão do mecanismo que permite conectar e utilizar um qualquer controlador externo, para controlar uma dada simulação interativa disponível na plataforma *PVSio-web*, uma vez que estes não possuem um mapeamento genérico. Por exemplo, o identificador do botão *X* do controlador externo da *PlayStation 4* pode não coincidir sempre com o identificador do botão equivalente ao botão *X* num outro controlador externo mais complexo, como o volante e os pedais do controlador externo *Logitech G29*. Para os controladores externos da *PlayStation 4* e da *XBOX One* o mapeamento é bastante semelhante, visto que os seus fabricantes seguem um padrão semelhante, quer na disposição física dos botões e *sticks*, quer na colocação desses identificadores lógicos. O mesmo já não acontece com o volante e os pedais *Logitech G29*, cujo mapeamento, apesar de ser também construído em conjunto com a *Sony*, não segue o mesmo padrão da *PlayStation 4*, e esta situação estende-se a outros fabricantes que possuem controladores externos bastante diferentes, como *joysticks*.

Para colmatar a dificuldade em mapear a estrutura dos diferentes controladores externos, este *widget* terá de receber os índices que indicam a posição do botão no controlador externo físico que estão associados às funcionalidades pretendidas. Por exemplo, para a funcionalidade "acelerar", este *widget* deverá receber o índice que corresponde ao botão físico do controlador externo que se utilizará para acelerar o veículo no simulador de condução. Para as restantes funcionalidades o processo deverá ser igual. Estes índices são os identificadores dos botões definidos pelos próprios fabricantes e que podem ser consultados nos manuais ou através de *event listeners* que detetem eventos *gamepadconnected* e *gamepaddisconnected*[5]. Os valores desses eventos podem ser facilmente visualizados em *html5gamepad*[6], sendo necessário conectar um controlador externo ao computador primeiro e clicar nos seus botões ou mover os seus *sticks* analógicos.

O mapeamento de cada controlador externo é composto por um identificador nominal único do mesmo, um *array buttons*, em que cada índice corresponde a um botão físico, e um *array axes*, em que cada índice corresponde a um *stick* físico. Um *stick*, também designado por *stick* analógico e por *joystick* é o componente dos controladores externos que é utilizado para mover objetos com base num referencial 2D, isto é, que usa as posições x e y, para calcular o ângulo de rotação. Este ângulo de rotação é obtido através da função matemática *atan2* que calcula o valor único da tangente do arco das duas coordenadas x e y, fornecidas pelo *stick* analógico. Visto que os controladores externos possuem referenciais 2D normalizados, entre -1 e 1, cujos ângulos são em radianos, é necessário converter tais ângulos para graus. Para tal será necessário utilizar a fórmula matemática *angleRad * 180 / Math.PI* que multiplica o valor do ângulo em radianos pelo rácio $180/\pi$. Uma vez que os ângulos podem

5 `https://developer.mozilla.org/en-US/docs/Web/API/Gamepad_API/Using_the_Gamepad_API` (*Using the Gamepad API*)

6 `http://html5gamepad.com/` (*HTML5 Gamepad Tester*)

ser elevados este *widget* deverá fornecer a possibilidade de suavizar as rotações. Para tal este *widget* considera um valor de sensibilidade, que será também fornecido pelo utilizador, por forma a obter ângulos, em graus, mais pequenos que promovem transições mais suaves, por exemplo, durante a rotação do volante.

Este *widget* deverá também receber os nomes das especificações formais associadas a cada um dos *widgets* que pretende controlar, os nomes das especificações formais associadas aos menus do simulador de condução (menus de início, pausa e fim) e os nomes das especificações formais associadas às ações do *widget Sound* do simulador de condução, para controlar o leitor de ficheiros áudio.

Este *widget* sempre que capturar eventos no botão físico do controlador externo, cujo índice é fornecido pelo utilizador, invoca os respectivos métodos *press()* e *release()* dos botões lógicos recebidos como campos opcionais que coincidem com a funcionalidade pretendida. O comportamento desses botões lógicos encontram-se especificados formalmente num modelo *PVS*.

Este *widget* possui um estado atual, *gamepadsKnown*, que permite ligar mais que um controlador externo em simultâneo. Sempre que um evento é capturado é necessário interpretar o mesmo e determinar se algum dos índices fornecidos nos campos opcionais (ou valores por omissão) foram pressionados ou clicados, e caso sejam, invocar os respectivos métodos das APIs dos *widgets* associados a esses índice ou invocar diretamentes as especificações formais, com base na escolha do utilizador na instanciação deste *widget*. Este *widget* invoca a função *listGamepads()* a cada 500 milissegundos, por forma a atualizar a lista dos controladores atualmente conectados (*gamepadsKnown*) através da função *JavaScript setInterval()*. Este *widget* invoca a função *JavaScript requestAnimationFrame()* para solicitar ao *browser* que invoque uma função para atualizar o estado do *widget*, que atualiza as *divs* criadas pelo *widget* com o botão clicado ou pressionado e/ou o *stick* analógico movido em cada instante.

Os campos opcionais mais relevantes fornecidos como terceiro argumento do construtor deste *widget* apresentam-se de seguida

1. ***carSteeringWheel*** define o volante a controlar (*widget SteeringWheel*).

2. ***carAccelerate*** define o botão lógico responsável por simular a aceleração no simulador de condução*widgets Speedometer* e *Tachometer*.

3. ***carBrake*** define o botão lógico responsável por simular a travagem no simulador de condução*widgets Speedometer* e *Tachometer*.

A apresentação de todos os campos opcionais e os seus valores por omissão podem ser consultados no Anexo C.

Este *widget* possui também um conjunto de métodos específicos. O método *listGamepadsKnown()* permite listar na consola do *browser* todos os controladores externos conectados até

esse instante. Este *widget* possui ainda um conjunto de métodos privados, que são invocados apenas pelo próprio *widget* que permitem obter a funcionalidade desejada. Os métodos privados *connectGamepad()* e *disconnectGamepad()* permitem adicionar e remover as *div* e atualizar o estado do *widget* (*gamepadsKnown*) com base em cada controlador externo conectado e desconectado. Os métodos privados *calculateRotationAngleWithSensitivity(y, x, sensitivity)* e *calculateRotationAngle(y, x)* permitem calcular os ângulos com base no movimento do *stick* analógico. Os métodos privados *radiansToDegrees(radAngle) degToRadians(degAngle)* permitem converter os ângulos de radianos para graus e vice-versa. O métodos privado *listGamepads()* permite atualizar o estado do *widget*. Os métodos privados *removeGamepad(g)* e *saveGamepad(g)* permitem remover e adicionar o controlador externo desconectado ou conectado. O método privado *updateStatus()* atualiza as *divs* de cada controlador externo e invoca os respectivos métodos, que por sua vez invocarão as especificações formais que definem o comportamento de cada um dos *widgets* recebidos como campos opcionais (volante, botão de aceleração e botão de travagem). Existem ainda outros métodos privados que apenas realizam o mapeamento textual entre os índices e a frase correspondente de acordo com o controlador externo. Por exemplo, quando o botão *square* é clicado ou pressionado em vez de apresentar o índice 2 (*key*), na *div* do controlador externo *PlayStation 4*, será apresentado o valor *'button square'*, uma vez que o método privado *mappingPS4GamepadButtons(key)* possui no seu mapeamento o par chave-valor *key: 2, value: 'button square'*. Para os restantes índices o processo é o mesmo, apenas varia o método, que depende do tipo de controlador externo (*PlayStation3*, *XBOX One*, *Logitech G29*, etc).

Futuramente, este *widget* poderá ser ligeiramente modificado para controlar outros *widgets*, sendo que as únicas mudanças se encontram ao nível dos campos opcionais que passarão a receber e à forma como os métodos desses *widgets* serão invocados quando um evento do controlador externo associado ao botão ou ao *stick* analógico for capturado.

4.4 WIDGETS AUXILIARES

Nesta secção apresentam-se os *widgets* desenvolvidos com o intuito de auxiliar a configuração e a criação de simulações interativas de painéis de instrumentos e de auxiliar a criação de protótipos de ambientes de condução.

4.4.1 *Sound*

O *widget Sound* introduz um mecanismo de reprodução de ficheiros áudio no contexto de *widgets* na plataforma *PVSio-web*. Este *widget* funciona como um leitor de músicas simples. Este *widget* cria *event listeners* que permitem realizar o *toggle* das imagens que dizem respeito às ações reproduzir, lado esquerdo na Figura 47 e silenciar, lado direito na Figura

Figura 47.: **Renderização do *Widget Sound* (lado esquerdo: Ficheiros Áudio em Reprodução; lado direito: Ficheiros Áudio em Pausa)**

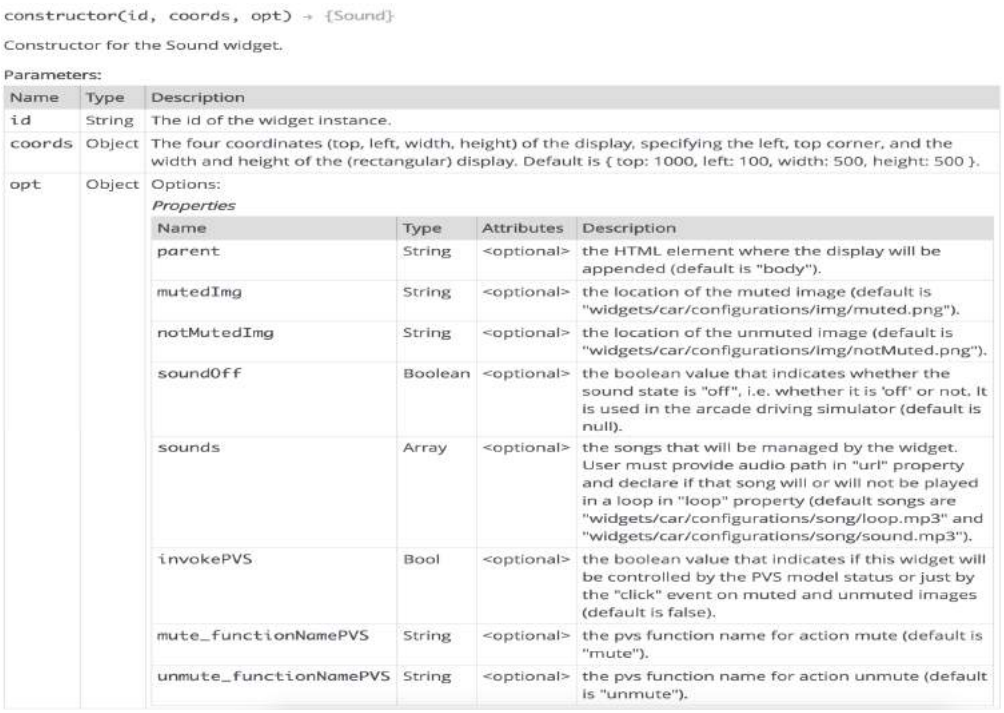
47, os ficheiros áudio, com base numa variável da especificação formal *PVS* ou com base no evento *"click"* associado à imagem. As restantes operações associadas com a reprodução de ficheiros áudio serão garantidas através de um conjunto de métodos.

Para a implementação deste *widget* recorreu-se à *tag HTML5 <audio>* [19], que fornece um conjunto de funções que permitem controlar um ficheiro áudio, das quais é importante salientar que a função que permite definir o novo volume apenas aceita valores compreendidos entre 0.0 e 1.0. O valor 0.0 representa o silêncio e o valor 1.0 representa o volume mais alto possível, que é o valor por omissão. O valor 0.3 implica uma redução de 30% do volume do ficheiro áudio.

No simulador de condução este *widget* é utilizado para reproduzir um conjunto de ficheiros áudios que simbolizam, respectivamente, a aceleração do veículo, o som do veículo em movimento sem aceleração, e um som arranque do veículo, quando se inicia o simulador de condução. Poderá também ser utilizado para simular um leitor de músicas do painel de instrumentos. Para obter um leitor de músicas mais complexo (controlo de faixas, de volume, etc) será necessário criar as interfaces que invoquem os métodos da API que garantem essas operações. Por exemplo, criar uma interface com um botão *up*, que invoca o método que incrementa o volume com uma dada quantidade predefinida.

Os campos opcionais mais relevantes fornecidos como terceiro argumento do construtor deste *widget*, que o distingue dos restantes *widgets*, apresentam-se de seguida

1. ***mutedImg*** define a imagem *mute* a colocar na interface.
2. ***notMutedImg*** define a imagem *unmute* a colocar na interface.
3. ***invokePVS*** define se o comportamento do *widget* é ou não controlado por uma variável no modelo formal *PVS*.
4. ***mute_functionNamePVS*** define o nome do método definido no modelo formal *PVS* que é invocado para silenciar os ficheiros áudio.
5. ***unmute_functionNamePVS*** define o nome do método definido no modelo formal *PVS* que é invocado para reproduzir os ficheiros áudio.
6. ***songs*** define a lista dos ficheiros áudio que se pretende adicionar ao leitor de músicas.



constructor(id, coords, opt) → {Sound}

Constructor for the Sound widget.

Parameters:

Name	Type	Description
id	String	The id of the widget instance.
coords	Object	The four coordinates (top, left, width, height) of the display, specifying the left, top corner, and the width and height of the (rectangular) display. Default is { top: 1000, left: 100, width: 500, height: 500 }.
opt	Object	Options: *Properties* Name / Type / Attributes / Description

Name	Type	Attributes	Description
parent	String	<optional>	the HTML element where the display will be appended (default is "body").
mutedImg	String	<optional>	the location of the muted image (default is "widgets/car/configurations/img/muted.png").
notMutedImg	String	<optional>	the location of the unmuted image (default is "widgets/car/configurations/img/notMuted.png").
soundOff	Boolean	<optional>	the boolean value that indicates whether the sound state is "off", i.e. whether it is 'off' or not. It is used in the arcade driving simulator (default is null).
sounds	Array	<optional>	the songs that will be managed by the widget. User must provide audio path in "url" property and declare if that song will or will not be played in a loop in "loop" property (default songs are "widgets/car/configurations/song/loop.mp3" and "widgets/car/configurations/song/sound.mp3").
invokePVS	Bool	<optional>	the boolean value that indicates if this widget will be controlled by the PVS model status or just by the "click" event on muted and unmuted images (default is false).
mute_functionNamePVS	String	<optional>	the pvs function name for action mute (default is "mute").
unmute_functionNamePVS	String	<optional>	the pvs function name for action unmute (default is "unmute").

Figura 48.: **Construtor do *Widget Sound***

A apresentação de todos os campos opcionais e os seus valores por omissão podem ser consultados na Figura 48.

A API deste *widget* disponibiliza métodos para controlar as suas funções, desde alterar o volume, iniciar e parar o som ou saber o estado do *widget*. Existem versões de métodos para actuar sobre todos os sons, fornecidos no campo opcional *songs*, ou sobre sons específicos (passando o *index* respectivo como argumento).

Para alterar o volume existem os métodos *setVolumeAll(newVolume)* e *setVolume(newVolume, index)* que definem o novo volume (*newVolume*).

Para iniciar e parar o som existem os métodos *playSound(index)*, *playAll()*, *pauseSound(index)* e *pauseAll()*.

Para controlar a reprodução dos ficheiros áudios existem os métodos *unmute()* e *mute()*. Estes métodos também trocam, respectivamente, a imagem da interface atual para a imagem *unmute* e *mute*.

Para definir uma sequência específica de reprodução de sons existe o método *onEndedSound(indexOnEnded, arrayNext)*. Este método permite reproduzir todos os ficheiros áudio fornecidos no argumento *arrayNext* mal acabe de reproduzir o ficheiro áudio com índice *indexOnEnded* na lista de áudios (campo opcional *songs*).

Para saber o estado do som, *muted* ou *unmuted*, no simulador de condução existe o método *getSoundOff()*. Este método acede ao conteúdo de uma *div* criada na instanciação do *widget Sound* pelo construtor do simulador de condução. Essa *div* é preenchida pelos mé-

todos *mute()* e *unmute()*, com os valores *true* e *false*, respectivamente, para que seja possível controlar o áudio individualmente para cada simulação.

Para criar os *event listeners* para as ações de *"click"* nas imagens da interface existe o método *startSound()*.

Por exemplo, caso os valores definidos nos campos opcionais *mute_functionNamePVS* e *unmute_functionNamePVS* sejam *"mute"* e *"unmute"*, as assinaturas dos métodos que terão de ser implementados, no modelo formal *PVS*, para controlar o comportamento do leitor de músicas, apresentam-se no Excerto de Código 11.

```
press_mute(st: state)
release_mute(st: state)
press_unmute(st: state)
release_unmute(st: state)
```

Excerto de Código 11: **Assinaturas dos Métodos a Implementar na Teoria *PVS* para Controlar o *Widget Sound***

4.4.2 *Customization*

O *Widget Customization* introduz um mecanismo que permite configurar e simplificar o processo de criar uma simulação interativa de painéis de instrumentos. Desta forma, os utilizadores que possuam poucos conhecimentos de programação poderão configurar as suas simulações textualmente em vez de instanciarem e configurarem manualmente os *widgets*. No simulador de condução o *widget Customization* é utilizado para instanciar e customizar os campos opcionais dos *widgets Arcade*, *TrackGenerator*, *GyroscopeController*, *VirtualKeypadController*, *GamepadController*, *DrawGamepad*, *SteeringWheel*, *Speedometer* e *Tachometer*, com base nas informações interpretadas nos menus de customização, de seleção de imagens e cores preenchidos pelo utilizador.

As customizações passarão pela,

1. seleção da imagem do volante, num conjunto de imagens disponíveis no *image picker*, que será fornecida ao *widget SteeringWheel*.
2. seleção das cores do simulador de condução, através de um *color picker*.
3. definição da topografia, dos perfis de elevação e dos sinais de trânsito da estrada que será renderizada no simulador de condução.
4. definição do ficheiro de mapeamento de coordenadas *JSON* da *spritesheet* com os sinais de trânsito e as restantes imagens que compõem os ambientes de condução a criar.

5. definição da(s) imagen(s) *PNG* com a *spritesheet*, que possui todas as imagens do mapeamento anterior agrupadas numa só imagem.
6. definição do(s) objeto(s) a colocar na paisagem nos ambientes a criar.
7. definição do(s) obstáculo(s) a colocar dentro da estrada definida anteriormente.
8. definição das configurações da estrada, que incluem o número total de zonas e o tamanho de cada zona, que possibilita a construção de estradas com diferentes tamanhos.
9. definição do tipo de veículo a usar no simulador, caso se pretenda apresentar um veículo, e se esse é ou não realista e o índice colocado como sufixo na imagem definida no mapeamento fornecido anteriormente.
10. configuração dos intervalos dos *widgets Speedometer* e *Tachometer*.
11. configuração do número de faixas que serão renderizadas na estrada definida acima.
12. configuração da frequência dos obtáculos a colocar dentro da estrada
13. configuração do número de voltas a renderizar pelo simulador de condução (*widget Arcade*)
14. definição da utilização ou não da teoria *PVS* para controlar o veículo no simulador de condução. Esta parametrização poderá ser utilizada, no futuro, para verificar o impacto da utilização da especificação formal para definir o comportamento do *widget Arcade*.

Os menus de customização que este *widget* cria podem ser encontrados nas Figuras 49, 50 e 51.

O processo de decisão que definiu as customizações supracitadas baseou-se nas informações fornecidas aos construtores dos *widgets* supracitados que este *widget* pretende instanciar automaticamente que requerem conhecimentos avançados de programação e sobre a arquitetura da plataforma *PVSio-web*. Para a implementação deste *widget* foi necessário recorrer a algumas bibliotecas *JavaScript* que permitiram, respectivamente, a criação de um *image picker* para se obter um menu de seleção de imagens, a criação de um *color picker* para se obter um menu de seleção de cores e a criação de um conjunto de *sliders* que permitem a definição de um valor pertencente a um dado intervalo de valores.

A biblioteca *JavaScript image-picker*[7] é um *plugin jQuery* simples que transforma a *tag HTML <select>* numa interface gráfica mais amigável. Esta biblioteca é utilizada em conjunto com as bibliotecas *JavaScript imagesLoaded* e *masonry*. A biblioteca *imagesLoaded*[8] permite detectar quando as imagens foram carregadas. A biblioteca *masonry*[9] é uma biblioteca

7 `http://rvera.github.io/image-picker/` (*Image Picker*)
8 `https://imagesloaded.desandro.com/` (*Detect when images have been loaded*)
9 `https://masonry.desandro.com/` (*Masonry Cascading grid layout library*)

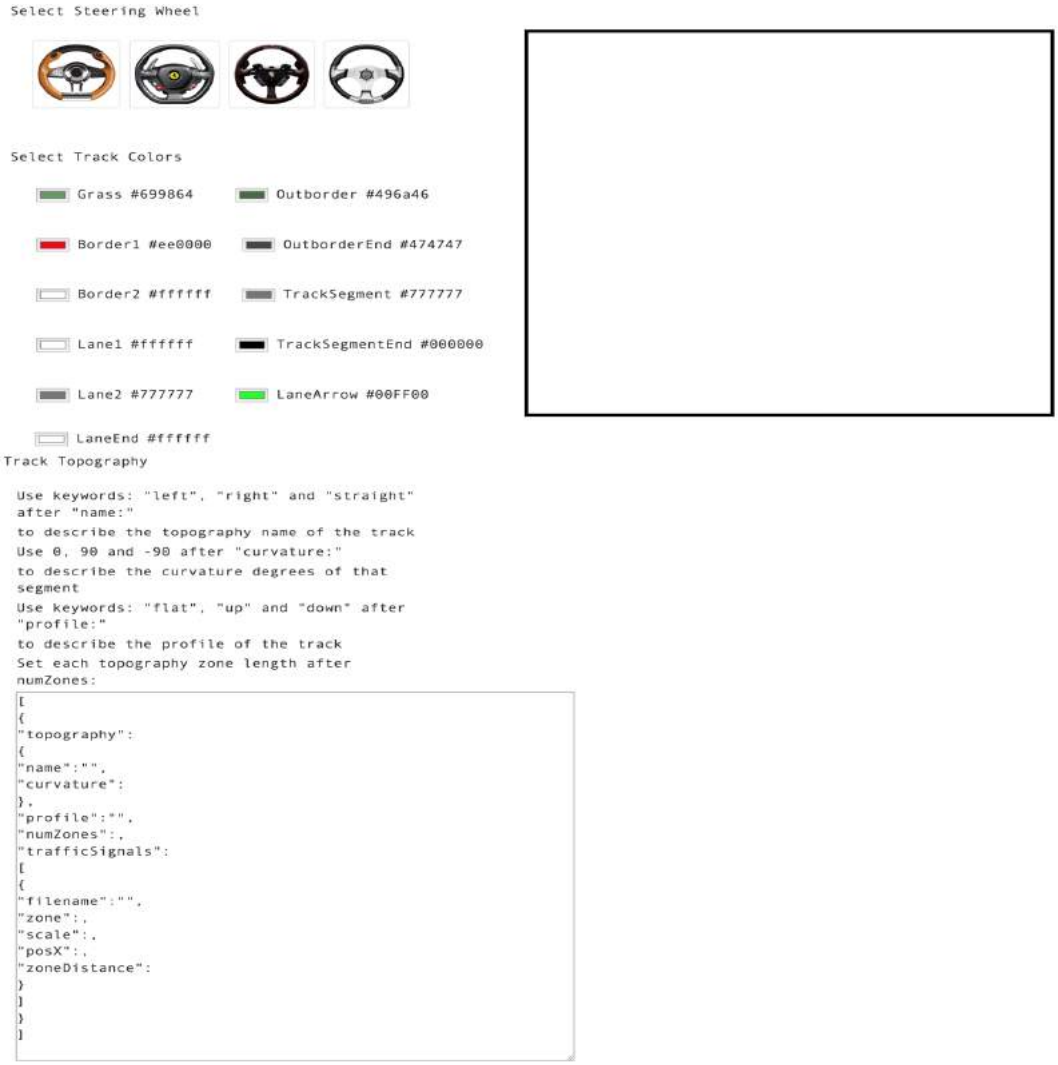


Figura 49.: **UI do *Widget Customization* - Parte 1**

Figura 50.: **UI do *Widget Customization* - Parte 2**

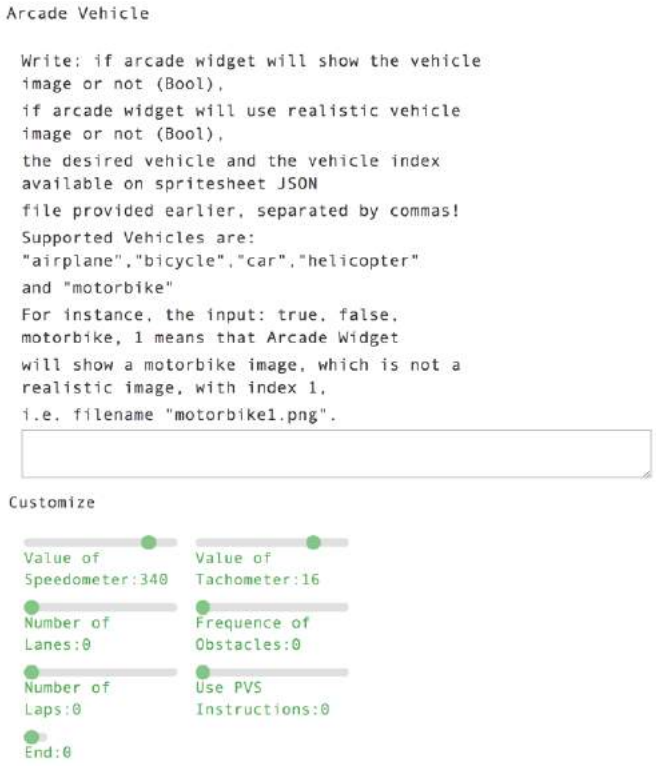


Figura 51.: **UI do *Widget Customization* - Parte 3**

JavaScript quer permite criar *layouts* de grelha de forma dinâmica, com base no espaço vertical disponível. A biblioteca *HTML DOM Input Color Object*[10] é uma biblioteca simples que permite criar um selector de cores recorrendo a uma *tag HTML <input>*, cujo atributo *type* toma o valor *"color"*. Para a implementação dos *sliders HTML*[11] foi necessário criar uma *tag HTML <input>* para cada um desses *sliders*, onde se definiu o atributo *type* com o valor *"range"*.

Para os restantes menus apenas foi necessário recorrer às *tags HTML <textarea>* que permitem que o utilizador possa escrever texto. A interpretação do conteúdo dessas *textareas* é realizada com a utilização de funções como *JSON.parse()* e *parseInt()*.

Os campos opcionais mais relevantes fornecidos como terceiro argumento do construtor deste *widget* apresentam-se de seguida

1. ***imagesSteeringWheels*** define as imagens dos volantes que serão colocadas no menu *Select Steering Wheel* (*image picker*). A definição inclui o caminho e o nome do ficheiro para cada imagem que é adicionada ao menu.

2. ***sliderRanges*** define os *sliders HTML* a criar no menu *Customize*. A definição inclui os limites inferior e superior de cada intervalo e o valor por omissão que irá aparecer selecionado na interface.

A apresentação de todos os campos opcionais e os seus valores por omissão podem ser consultados na Figura 52.

Este *widget* possui também um conjunto de métodos específicos. O método *endRange(callback,car,reRenderedWindowCSSValues, sliders,steeringWheelStyle)* é o método da API que recorre ao método *on()* da biblioteca *jQuery*[12], que anexa a função que será invocada quando o utilizador mover esse *slider* para a direita no *slider End*, o que representa o fim da customização. Os argumentos deste método dizem respeito à função *callback*, que comunica com o modelo *PVS*, ao *array* dos novos estilos a aplicar (*Markup CSS*) de todos os elementos cuja aparência visual se pretende alterar, ao *array* com o valor de cada *slider* e ao estilo do volante. O método *rangeEvents(sliders)* é responsável por atualizar o objeto *sliders*, com os valores atualmente selecionados nos *sliders* no menu *Customize*. O método *setInitRenderingDiv(initWindowCSSValues)* permite definir a aparência visual inicial das interfaces que compõem o *widget Customization* com base no *array* com o *Markup CSS* inicial de todos os elementos (*Markup HTML*) criados. O método *setLastRenderingDiv(gaugeClass)* adiciona o identificador *"last-gauge"* a todas *div* com *id "gauge"*. Este método permite distinguir as *div* com os *widgets* que já tinham sido instanciados dos que foram instanciados com as configurações dos menus de customização, por forma a que sejam removidas mais

10 `https://www.w3schools.com/jsref/dom_obj_color.asp` (*HTML DOM Input Color Object*)
11 `https://www.w3schools.com/howto/howto_js_rangeslider.asp` (*Learn how to create custom range sliders with CSS and JavaScript*)
12 `http://api.jquery.com/on/` (*Method .on()*)

constructor(id, coords, opt) → {Customization}

Constructor for the Customization widget.

Parameters:

Name	Type	Description
id	String	The id of the widget instance.
coords	Object	The four coordinates (top, left, width, height) of the display, specifying the left, top corner, and the width and height of the (rectangular) display. Default is { top: 0, left: 0, width: 250, height: 250 }.
opt	Object	Options: *Properties* Name / Type / Attributes / Description

Name	Type	Attributes	Description
parent	String	<optional>	the parent div (default is "body").
sliderColor	String	<optional>	the sliders circle color (default is "#4CAF50").
imagesSteeringWheels	Array	<optional>	array of objects with all the images to insert in the imagepicker, providing the path and value fields (default values are {"basic","ferrari","porsche","sparco"}+"_steering_wheel.svg" and default paths are "widgets/car/steering_wheels/"+{"basic","ferrari","porsche","sparco"}+"_steering_wheel.svg").
sliderRanges	Array	<optional>	array of objects with all the sliders to be created, giving the name (to put in the div), the values min, max (slider intervals) and value (default value) (default is [{name: "speedometer",min: 0,max: 400,value: 340},{name: "tachometer",min: 0,max: 20,value: 16},{name: "lanes",min: 0,max: 3,value: 0},{name: "hills",min: 0,max: 10,value: 0},{name: "obstacles",min: 0,max: 10,value: 0},{name: "other-cars",min: 0,max: 10,value: 0}]).
controlsText	Array	<optional>	array of objects with information about widget control (default is ["Car controls:","[left/right arrow keys] Turn Left/Right","[up/down arrow keys] Accelerate/Brake"]).
gauges	Array	<optional>	array of objects with information about gauge widgets to create (default is [{name:"speedometer-gauge",styleId:"",style:""},{name:"tachometer-gauge",styleId:"float",style:"right"}]).
gaugesStyles	Array	<optional>	array of objects with CSS styles to add to the above gauge widgets (default is [{zoom: "45%",marginLeft: "370px",marginTop: "430px"}]).

Figura 52.: **Construtor do *Widget Customization***

facilmente pelos métodos privados *removeChild(id)* e *removeParentAllChilds(id)*, obtendo um *Markup HTML* sempre atualizado e sem elementos desnecessários.

Este *widget* possui um conjunto de métodos privados, apenas invocados pelo próprio *widget*, que permitem obter a funcionalidade desejada. O método privado *getSteeringWheelImage()* é responsável por selecionar a *div*, que contém o nome da última imagem do volante selecionada no *image picker*, que será fornecida ao *widget SteeringWheel*. O método privado *removeChild(id)* remove a *div* cujo identificador é *id*. O método privado *removeParentAllChilds(id)* remove todas as *divs*, cuja *div* pai possui o identificador *id*. Estes últimos métodos permitem eliminar *divs* que deixaram de ser necessárias num dado momento, por exemplo, as *divs* dos *widgets Speedometer* e *Tachometer* com a aparência visual antiga. O método privado *reRenderedWindowCSS(reRenderedWindowCSSValues)* é responsável por atualizar todo o *Markup CSS* dos *widgets* após o utilizador terminar a customização. O método privado *setImagePicker(aux)* é responsável por invocar o método *imagepicker()* da biblioteca *image-picker* que permite obter o nome da imagem selecionada no *image picker* ao fornecer uma função, que será invocada quando uma nova imagem for selecionada, no campo *selected* desse método.

A utilização do *widget Customization* está restrita ao mecanismo manual de *copy/paste* dos ambientes de condução criados com base nas informações fornecidas nos menus de custo-

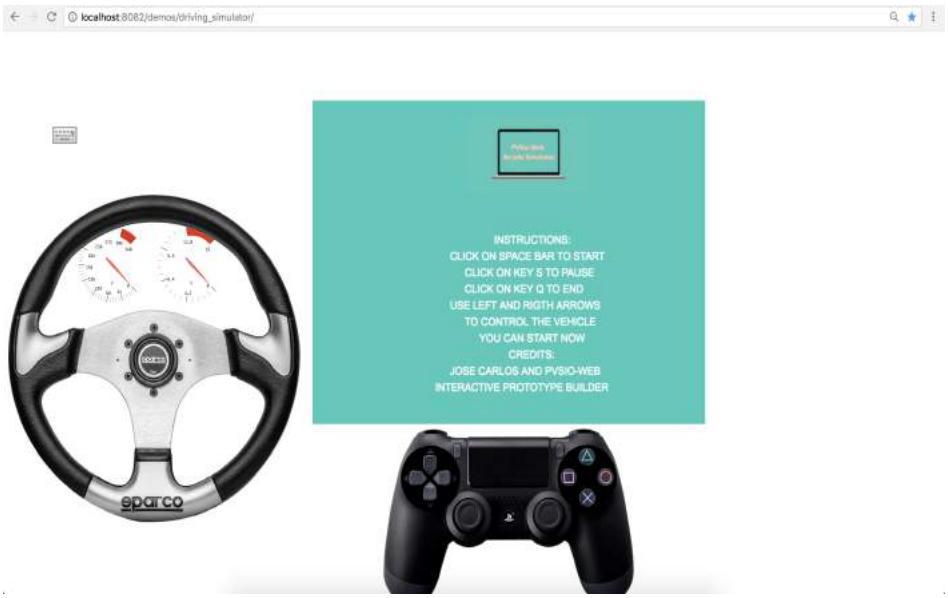


Figura 53.: **UI Resultado do *Widget Customization***

mização para um ficheiro de configuração *JSON* uma vez que a plataforma *PVSio-web* não possui nenhuma API de escrita de ficheiros, no contexto de *widgets*, o que impossibilita a escrita automática. A Figura 53, apresenta um possível resultado final de uma customização realizada com o *widget Customization*.

4.4.3 `TrackGenerator`

O *Widget TrackGenerator* é um dos componentes que compõe o mecanismo de simulação de ambientes de condução totalmente configurável. Este *widget* é responsável pela criação de todos os segmentos, que compõem os ambientes que se aproximam dos ambientes de condução reais, e que serão interpretados e renderizados pelo *widget Arcade*. Esta divisão, entre a criação dos ambientes e a renderização dos mesmos permite que o simulador de condução seja mais dinâmico e organizado ao separar as parametrizações que permitem criar os ambientes de condução desejados e as parametrizações que permitem visualizar esses ambientes. Por exemplo, será possível simular os mesmos ambientes de condução com diferentes tipos de veículos em simultâneo. Tal é conseguido com a instanciação de um *widget TrackGenerator* e com diferentes instanciações do *widget Arcade*.

A implementação deste *widget* depende da implementação do *widget Arcade*, uma vez que os segmentos a criar, que compõem os ambientes de condução, dependem do motor que irá interpretar os mesmos por forma a que seja possível visualizar a sequência de segmentos animados que constitui o jogo de corridas. Para tal foi necessário decidir qual a tecnologia que melhor se adapta a esta divisão e que pode ser facilmente integrada na plataforma

PVSio-web. Decidiu-se que se utilizaria *Canvas HTML* uma vez que esta tecnologia fornece um conjunto de funções que permitem desenhar e colocar imagens em determinadas posições que em conjunto com técnicas de renderização permitem obter uma perspectiva 3D utilizando gráficos 2D.

As posições horizontais numa *Canvas HTML* tomam valores positivos, negativos e zero. O valor zero representa o centro do referencial (ponto central do limite inferior da *Canvas*), que será o centro do simulador de ambientes de condução e onde o veículo será colocado. Os valores positivos representam o lado esquerdo e os valores negativos representam o lado direito.

As posições verticais numa *Canvas HTML* são todas positivas para cima. A extensão da estrada a criar será dada pelo número de blocos que a compõem, em que cada bloco possuirá o mesmo comprimento constante. Então, a distância total a percorrer na estrada será dado pela multiplicação entre o número de blocos e o comprimento de cada bloco.

O simulador de condução aproveitou não só o mecanismo do jogo simples *arcade*, desenvolvido em *racerJS*[13], que permite criar e visualizar o jogo numa *Canvas HTML5*, mas também o mecanismo de animação, que permite visualizar o mesmo. Foram desenvolvidos métodos por cima desses mecanismos e outros mecanismos que permitiram a integração na plataforma *PVSio-web* e que utiliza configurações e parametrizações definidas por utilizadores nos campos opcionais e em ficheiros de configuração *JSON*.

Os campos opcionais mais relevantes fornecidos como terceiro argumento do construtor deste *widget* apresentam-se de seguida

1. ***spritesFilename*** define o ficheiro de mapeamento de coordenadas das imagens na *spritesheet* que possui as imagens que serão utilizadas para construir os ambientes de condução no simulador de ambientes de condução.

2. ***render*** define as opções de renderização do simulador de ambientes de condução, como o posicionamento da câmara e a profundidade do campo de visão.

3. ***numLanes*** define o número de faixas que serão desenhadas na estrada a construir.

4. ***trackParam*** define o número total de zonas que compõem a estrada e o tamanho constante de cada uma dessas zonas.

5. ***objects*** define o nome, a posição horizontal e a escala a aplicar a cada uma das imagens que serão colocadas na paisagem do simulador de condução. Essas imagens terão de estar presentes na *spritesheet* e no ficheiro de mapeamento de coordenadas correspondente.

6. ***obstacle*** define o nome, a posição horizontal e a escala a aplicar a cada uma das imagens que serão colocadas dentro da pista do simulador de condução, como obstáculos.

13 `https://github.com/onaluf/RacerJS` (*A simple racing game inspired by the classic Lotus series on Amiga*)

Essas imagens terão de estar presentes na *spritesheet* e no ficheiro de mapeamento de coordenadas correspondente.

7. ***trackLayout*** define a topografia, perfis de elevação e sinais de trânsito da estrada que em conjunto com a paisagem compõem os ambientes de condução a criar. Essa definição assenta em valores predefinidos e especifícos para cada um dos blocos que constituem a estrada a criar. Cada bloco tem as propriedades *topography*, *profile*, *numZones* e *trafficSignals*. A propriedade *topography* possui as informações acerca da topografia do bloco. A topografia de cada bloco pode ser uma secção rectilínea ou curvilínea à esquerda ou curvilínea à direita. Os valores possíveis são *left*, *right* e *straight*. A propriedade *profile* possui a informação do perfil de elevação do bloco. O bloco poderá ser ascendente (subida), descendente (descida) ou plano. Os valores possíveis são *up*, *down* ou *flat*. A propriedade *numZones* define o número de réplicas deste bloco a criar. A propriedade *trafficSignals* diz respeito às imagens de sinais de trânsito que terão de ser colocadas nesse bloco. Então, define as posições horizontais, *posX*, dentro dos limites desse bloco, *zone*, onde um ou mais sinais de trânsito, com o *filename* serão colocados com uma dada escala, *scale*. Define, ainda, a posição relativa dentro do bloco onde serão colocados os sinais de trânsito, com base no comprimento do bloco, *zoneSize*, e define a distância a que cada sinal deve ser colocado, *zoneDistance*. Por exemplo, se o bloco possuir um comprimento total de 250 (valor por omissão) e a propriedade *zoneDistance* possuir o valor 40, então esse sinal de trânsito deverá colocado a cerca de 16% desde o início do bloco (40/250).

8. ***trackColors*** define as cores presentes no simulador de condução. Por exemplo, a cor verde representa relva.

9. ***obstaclePerIteration*** define a frequência com que se coloca um obstáculo, dos fornecidos no campo opcional *obstacle*, na estrada a criar.

A apresentação de todos os campos opcionais e os seus valores por omissão podem ser consultados no Anexo D.

O *widget TrackGenerator* define a imagem na propriedade *sprite* que toma um de dois valores possíveis, ou *false* caso não exista imagem, ou um objeto com as propriedades *type*, *pos*, *obstacle* e *scale* caso exista imagem nesse segmento.

A propriedade *type* possui as coordenadas x, y, w e h que correspondem às posições x, y, à largura e à altura da imagem original presente na *spritesheet* e no seu ficheiro de mapeamento de coordenadas *JSON*. Os valores dessas coordenadas são preenchidos com base num filtro que compara se a propriedade *filename* fornecida nos campos opcionais *objects* ou *obstacle* é igual à propriedade *filename* presente nesse ficheiro de mapeamento.

A propriedade *pos* diz respeito à posição horizontal onde essa imagem será colocada no simulador de ambientes de condução com base nas posições horizontais fornecidas no campo opcional *positionsX*.

A propriedade *obstacle* define se a imagem é um objecto da paisagem, valor zero, ou se é um obstáculo a colocar dentro da estrada, valor um.

A propriedade *scale* define a escala a aplicar individualmente a essa imagem no momento da sua renderização pelo *widget Arcade*.


```
{"laneWidth":0.02,"numLanes":2,"numberOfSegmentPerColor":4,"render":{"
    depthOfField":150,"camera_distance":30,"camera_height":320},"track":[{"
    height":0,"curve":0,"sprite":false},...,{"height":0,"curve"
    :-1576.7018229566875,"sprite":{"type":{"x":0,"y":506,"w":178,"h":115},"pos
    ":1.7,"obstacle":0,"scale":6}}],"trackParam":{"numZones":1000,"zoneSize"
    :250},"trackSegmentSize":5,"trackColors":{"grass1":"#344C32","border1":"#
    ffa500","border2":"#ffffff","outborder1":"#7F967D","outborder_end1":"
    #474747","track_segment1":"#777777","lane1":"#ffffff","lane2":"#777777","
    laneArrow1":"#ffff00","track_segment_end":"#000000","lane_end":"#ffffff"}}
```


Excerto de Código 12: **Resultado da Instanciação do *Widget TrackGenerator***

Este *widget* possui também um conjunto de métodos específicos que confere a funcionalidade desejada ao *widget*. O método *render()* difere um pouco da definição genérica apresentada no início da secção 4.1 do capítulo 4. Neste *widget* este método permite visualizar o estado da criação dos ambientes de condução através de um valor *boolean* que é apresentado na sua interface. Esse valor *boolean* passa a *true* quando a consola do *browser* apresentar o ficheiro de configuração *JSON* produzido. Esse ficheiro contém todos os segmentos que compõem os ambientes de condução desejados. Um exemplo de um ficheiro de configuração *JSON* produzido pelo *widget TrackGenerator* encontra-se no Excerto de Código 12. Atualmente, esses segmentos têm de ser colocados num ficheiro *JSON*, manualmente devido à inexistência de uma API de escrita de ficheiros na plataforma *PVSio-web*, no contexto dos *widgets*. Futuramente, quando essa API existir este processo será automatizado. O método *generateStraightTrack()* permite criar uma estrada rectilínea, sem perfis de elevação e sem sinais de trânsito, e criar os ambientes de condução na sua vizinhança aleatoriamente. O método *generateTrackCurvesSlopes()* permite criar uma estrada com diferentes topografias, com curvas e rectas, com diferentes perfis de elevação, com planos, subidas, descidas, e sem sinais de trânsito, e criar os ambientes de condução na sua vizinhança aleatoriamente. A utilização destas últimos dois métodos implica que o utilizador não precisa de preencher o campo opcional *trackLayout*, visto que esse não será utilizado durante o processo de criação supracitado. O método *generateTrackBasedOnTrackLayoutOptField()* permite criar uma estrada com diferentes topografias, com curvas e rectas, com diferentes perfis de elevação,

com planos, subidas, descidas, e com sinais de trânsito, e criar os ambientes de condução na sua vizinhança com base no conteúdo do campo opcional *trackLayout*.

4.5 WIDGET DE ANIMAÇÃO

Nesta secção apresenta-se o *widget* desenvolvido com o intuito de animar e visualizar os ambientes de condução criados com base em parametrizações do utilizador.

4.5.1 *Arcade*

O *Widget Arcade* é o último componente que compõe o mecanismo de simulação de condução que se desenvolveu para a plataforma *PVSio-web*. Este *widget* é responsável pela visualização de todos os segmentos, que compõem os ambientes de condução criados pelo *widget TrackGenerator*.

Este *widget* receberá esses segmentos no ficheiro de configuração *JSON*, com o formato apresentado no Excerto de Código 12. Receberá também algumas opções de renderização, como o posicionamento da câmara, a profundidade do campo de visão e as cores a colocar no simulador (delimitadores, paisagem, estrada, etc) que permitem configurar a visualização desses ambientes de condução.

Para a implementação deste *widget* recorreu-se à biblioteca *JavaScript jchronometer.js*[14] que permite definir cronómetros. Esta biblioteca não tem dependências e é muito leve, uma vez que apenas possui cerca de 100 linhas de código. A integração de um cronómetro no simulador de condução permite que o utilizador tenha a percepção de quanto tempo demorou a simular os ambientes que criou, particularmente útil se respeitar os limites de velocidade aplicados nos sinais de trânsito. Esta biblioteca permite invocar o modo pausa, sempre que o utilizador parar a simulação e permite reiniciar o cronómetro quando uma nova simulação ocorre.

Os campos opcionais mais relevantes fornecidos como terceiro argumento do construtor deste *widget* apresentam-se de seguida

1. ***trackFilename*** define o nome do ficheiro de configuração dos ambientes de condução, cujo conteúdo é resultante da invocação de um dos métodos da API do *widget TrackGenerator*. Esse ficheiro possui os segmentos e algumas configurações dos ambientes de condução a renderizar.

2. ***spritesFilename*** define o ficheiro de mapeamento de coordenadas *JSON* da *spritesheet* utilizado na construção desses segmentos pelo *widget TrackGenerator*.

14 `https://github.com/lingtalfi/jChronometer` (*A javascript chronometer*)

3. ***spritesFiles*** define a *spritesheet*, cujo mapeamento de coordenadas é fornecido no campo opcional *spritesFilename*, e a *spritesheet* com a fonte a usar nos elementos textuais do simulador de condução, no caso de ser uma visualização *arcade*. Esta imagem deverá possuir letras, números, entre outros.

4. ***realisticImgs*** define se os ambientes de condução utilizam imagens com mais ou menos qualidade, respectivamente, uma visualização do tipo *realista* ou do tipo *arcade*.

5. ***useVehicle*** define se o simulador de ambientes de condução renderizará uma imagem de um veículo.

6. ***vehicle*** define o veículo a renderizar e que será controlado no simulador de condução. Existem cinco tipos de veículos, "*airplane*","*bicycle*","*car*","*helicopter*" e "*motorbike*". Para qualquer um destes veículos o utilizador deverá certificar que possui três imagens para o veículo selecionado na *spritesheet*, fornecida no campo opcional *spritesFiles*, e no ficheiro de mapeamento, fornecido no campo opcional *spritesFilename*. Essas três imagens dizem respeito às imagens com o veículo virado para a esquerda, virado para a direita e virado para a frente por forma a que o simulador de condução altere a imagem do veículo consoante a direção presente no estado atual do modelo *PVS*. Caso não existam as três imagens para o veículo selecionado ou o utilizador não escolha nenhum desses veículos o simulador apresentará o veículo "*car*", que é o valor por omissão presente na *spritesheet* e no ficheiro de mapeamento correspondente por omissão. Salienta-se que nestes ficheiros por omissão existem três imagens de um carro (carro virado para a esquerda, virado para a direita e virado para a frente).

7. ***vehicleImgIndex*** define o sufixo da imagem (*sprite*) do veículo presente na propriedade *filename* do ficheiro de mapeamento, fornecido no campo opcional *spritesFilename*.

8. ***logoImgIndex*** define o sufixo da imagem (*sprite*) do logo presente na propriedade *filename* do ficheiro de mapeamento, fornecido no campo opcional *spritesFilename*.

9. ***backgroundImgIndex*** define o sufixo da imagem (*sprite*) do *background* presente na propriedade *filename* do ficheiro de mapeamento, fornecido no campo opcional *spritesFilename*.

10. ***stripePositions*** define as posições onde começam e onde acabam os elementos que compõem uma faixa horizontal inteira da *Canvas HTML5*. Uma faixa horizontal, na *Canvas*, é composta por *paisagem-berma-border exterior-border-border interior-estrada-border interior-border-border exterior-berma-paisagem*. Esta propriedade permite ter diferentes ambientes de condução quando a topografia, o perfil de elevação e os sinais de trânsito são os mesmos. Por exemplo, modificar o valor da largura do delimitador,

que é uma das propriedades deste objeto, permitirá obter uma visualização com delimitadores (*borders*) mais ou menos largos, cuja estrada possui a mesma topografia e perfis de elevação.

Existem ainda mais campos opcionais que dizem respeito à definição dos nomes das variáveis do estado, definidas nas especificações formais, que determinam o comportamento de cada uma das ações.

A apresentação de todos os campos opcionais e os seus valores por omissão podem ser consultados no Anexo E.

Este *widget* possui também um conjunto de métodos específicos. O método *startSimulation()* dá início à simulação após realizar a leitura do ficheiro de configuração dos ambientes *JSON* e do ficheiro de mapeamento de coordenadas *JSON* das imagens da *spritesheet*. O método *render(res)* é a variação do método *render()* já discutido na instanciação e utilização genérica de um *widget* de *PVSio-web* na secção 4.1 no capítulo 4. Neste *widget* este método é responsável por verificar que o estado não é nulo nem *undefined*, e por atualizar o estado interno do simulador de condução com o conteúdo desse estado do modelo formal *PVS*. Este método é invocado sempre que é detectada uma interação na simulação interativa ou é invocada de *x* em *x* milissegundos por uma função *tick* definida onde este *widget* foi instanciado. Esse estado do modelo formal *PVS* contém as informações acerca da posição atual do veículo no simulador de ambientes de condução, sobre a sua velocidade, sobre a direção do veículo, entre outras. É com base no conteúdo do estado do modelo formal *PVS* que o simulador de condução atualiza a posição do veículo, os ambientes a renderizar no campo de visão parametrizado e apresenta os menus de pausa, fim e início, quando o utilizador retoma a simulação.

Este *widget* possui um conjunto de métodos privados, que são invocados dentro do *widget*. A lista completa pode ser consultada no Anexo E. Existem métodos privados para ler, interpretar e filtrar o conteúdo dos ficheiros recebidos como campos opcionais, para apresentar os menus de fim e de pausa da simulação, para apresentar texto no simulador de condução, para desenhar uma seta simples, seta completa ou linha guia que indica a direção atual do veículo, para alterar as cores do simulador de condução, para apresentar o menu de início da simulação e desenhar os segmentos com a topografia e o perfil de elevação da estrada, criados no *widget TrackGenerator*, para desenhar faixas na estrada, para desenhar imagens quer na paisagem, quer na pista (obstáculos), quer no horizonte (*background* do simulador) e para definir as novas posições do veículo, com base ou não no conteúdo do modelo *PVS*.

As diferenças na qualidade dos ambientes de condução renderizados, nas cores do simulador de condução e nas fontes, que fornecem informações sobre o estado da simulação (tempo decorrido, velocidade atual, etc), numa simulação do tipo *arcade* e do tipo *realista* podem ser observadas nas Figuras 54 e 55.

Figura 54.: **Aparência Visual do *Widget Arcade* do Tipo *Realista***

Este *widget* em conjunto com os *widgets Speedometer*, *Tachometer*, *SteeringWheel* e *LincolnMKCDashboard* permitem obter uma simulação de condução mais realista, com uma vista "pára-brisas", que permite que o utilizador interaja com os diferentes componentes como se estivesse dentro do veículo, imediatamente atrás do volante (*widget SteeringWheel*) e com os ambientes de condução à sua frente e na sua vizinhança. A integração desses *widgets* para o painel de instrumentos do caso de estudo pode ser observada na Figura 56.

Na próxima secção será abordado com mais detalhe o modo como funciona a visualização dos segmentos criados, tendo em conta o posicionamento da câmara, a profundidade do campo de visão, entre outros parâmetros.

4.6 MECANISMO DE RENDERIZAÇÃO

O mecanismo de renderização de ambientes utilizado no *widget Arcade* tem por base toda a mecânica de renderização implementada em *racerJS*[15], sobre a licença *Massachusetts Institute of Technology (MIT)*, que permite representar um ambiente 3D em 2D.

A Figura 57 apresenta um diagrama simplificado com a sequência de invocações de métodos do *widget Arcade* que permitem obter a animação gráfica do simulador de condução. Essa sequência começa com a invocação do método público, *startSimulation()*, pelo utilizador e continua com a invocação de métodos privados. A variável *WIDGETSTATE* contém sempre o estado mais recente fornecido pelo servidor. Esse estado possui todos os atributos e valores especificados e calculados pelo modelo formal *PVS*, que é responsável por determinar o comportamento do *widget* com base nas interações do utilizador capturadas.

A iteração dos *frames*, no método *renderEachSimulationFrame()*, ocorre com base na profundidade do campo de visão definida pelo utilizador na propriedade *depthOfField* no objeto *render* fornecido como campo opcional ao *widget TrackGenerator*. Os ambientes a renderizar

15 `https://github.com/onaluf/RacerJS` (*A simple racing game inspired by the classic Lotus series on Amiga*)

Figura 55.: **Outra Aparência Visual do *Widget Arcade* do Tipo *Arcade***

Figura 56.: **Aparência Visual do *Widget Arcade* do Tipo *Realista* numa Simulação com a Versão Antes da Correção do Painel de Instrumentos do Veículo do Caso de Estudo (*Widget LincolnMKCDashboard*)**

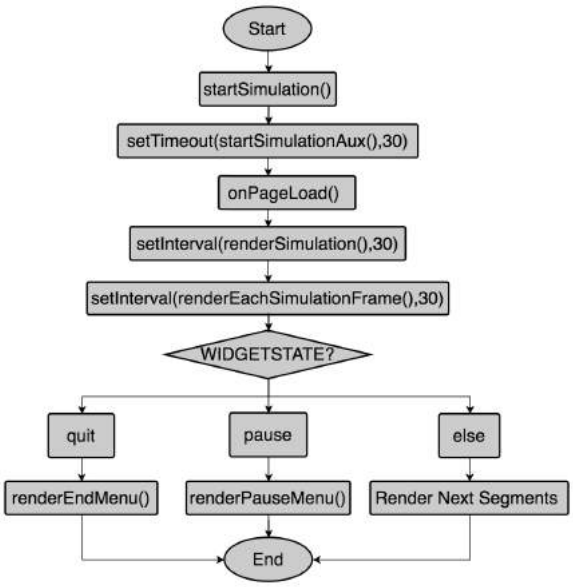


Figura 57.: **Diagrama Simplificado com o Mecanismo de Renderização**

estão configurados no ficheiro fornecido no campo opcional *trackFilename* ao *widget Arcade*. O *widget Arcade* utiliza uma variável *absoluteIndex* que determina o índice da estrada e dos seus ambientes envolventes a renderizar. Esta variável baseia-se no rácio entre a posição atual do veículo no jogo de corridas e o comprimento do segmento da pista. Quando esta tomar o valor correspondente à profundidade do campo de visão o simulador renderiza a última *frame*, que equivale ao final da estrada e dos seus ambientes. Consoante as configurações do *widget Arcade*, o veículo poderá ser colocado na posição inicial e recomeçar a simulação um número finito de vezes, que introduz a noção de voltas na pista, ou um número infinito, que corresponde a uma simulação infinita que só terminará quando o utilizador desejar. Esta ação é controlada pelo modelo *PVS*, cuja especificação formal utilizada se encontra parametrizada no campo opcional *newLap_functionNamePVS*, e que, por sua vez, faz o *reset* das posições horizontal e vertical do veículo na simulação.

O método *startSimulation()* invoca o método *startSimulationAux()* após 30 milissegundos. Este atraso na invocação é conseguido com a utilização do método *setTimeout()* e surge da necessidade de garantir que todos os ficheiros de configuração são lidos. O método *startSimulationAux()* invoca o método *onPageLoad()* após terminar o processo de interpretação dos ficheiros de configuração lidos. O método *onPageLoad()* é responsável pela leitura das imagens *spritesheet* que possuem as imagens utilizadas no processo de criação de ambientes de condução e invoca o método *renderSimulation()* a cada 30 milissegundos, utilizando o método *setInterval()*. O método *renderSimulation()* é responsável por renderizar o menu inicial do jogo de corridas e por iniciar a simulação quando o valor do atributo *action_attribute* no estado *WIDGETSTATE* possuir o mesmo valor que o atributo *resume_attribute*. A simu-

lação inicia com invocação do método *clearInterval()* para terminar a invocação do método *renderSimulation()* a cada 30 milissegundos. E termina com a invocação do método *renderEachSimulationFrame()* a cada 30 milissegundos, utilizando o método *setInterval()*. O método *renderEachSimulationFrame()* é responsável por limpar a tela, ao colorir a tela toda com a cor hexadecimal *#dc9*, e por desenhar a nova *frame* do jogo. É também responsável por renderizar o menu de pausa quando o valor do atributo *action_attribute* no estado *WIDGETSTATE* possuir o mesmo valor que o atributo *pause_attribute*. O método *renderPauseMenu()* é invocado a cada 30 milissegundos, utilizando o método *setInterval()*, e é responsável por renderizar o menu de pausa. É ainda responsável por renderizar o menu de fim quando o valor do atributo *action_attribute* no estado *WIDGETSTATE* possuir o mesmo valor que o atributo *quit_attribute*. O método *renderEndMenu()* é invocado a cada 30 milissegundos, utilizando o método *setInterval()*, e é responsável por renderizar o menu de fim. O método *clearInterval()* é invocado, sempre que os métodos *renderPauseMenu()* e *renderEndMenu()* são invocados, para terminar a invocação do método *renderEachSimulationFrame()* a cada 30 milissegundos.

O atributo *action_attribute* é atualizado com base nas interações do utilizador com o simulador de condução pelo modelo *PVS*, no servidor. As interações do utilizador são apresentadas nos menus inicial, de pausa e de fim, que coincidem com os atributos *quit_attribute*, *pause_attribute* e *resume_attribute*. Os atributos supracitados podem tomar um qualquer valor no modelo de *PVS*. Para tal apenas será necessário indicar esses valores nos respectivos campos opcionais na instanciação do *widget Arcade*. A criação de campos opcionais reservados para os valores dos atributos do modelo formal *PVS* confere uma dinâmica maior ao *widget Arcade*, uma vez que este *widget* nunca compara valores estáticos para determinar a resposta a uma dada ação do utilizador, mas sim compara o conteúdo dos atributos.

4.6.1 *Detalhes de Implementação do Mecanismo de Renderização*

A fórmula matemática apresentada no Excerto de Código 13 define a altura e a curvatura de cada segmento, que serão utilizados para determinar a altura da projeção na *Canvas* e a curvatura do segmento que será desenhado pelo *widget Arcade*. Esses valores recorrem à posição do próximo segmento, dado pela variável *i*, e pelos valores da altura e curvatura do último segmento desenhado, da zona atual. Com esta fórmula, é possível criar uma zona com perfil de elevação ascendente, cujos segmentos possuirão alturas contínuas e suaves, em vez de mudanças abruptas de altura e curvatura.

```
// topography name
case "straight": intendedCurveForCurrentZone=0; break;
case "left": intendedCurveForCurrentZone = (parseInt(topographyRead[tmpIter].
    topography.curvature) * 10) * self.randomPos(); break;
case "right": intendedCurveForCurrentZone = (parseInt(topographyRead[tmpIter].
    topography.curvature) * 10) * self.randomPos();break;

// profile
case "flat": intendedHeightForCurrentZone=0; break;
case "up": intendedHeightForCurrentZone = 900 * self.randomPos(); break;
case "down": intendedHeightForCurrentZone = - 900 * self.randomPos(); break;

// Iterate trafficSignals and fill object sprite with landscape object or
    track obstacle (based on \textit{obstaclePerIteration})
sprite = {type: {x:,y:,w:,h:}, pos: , obstacle: , scale:};

// Math formula to create continuous segments
self.generatedTrack.push({
height: currentZone.height+intendedHeightForCurrentZone / 2 * (1 + Math.sin(i/
    self.trackParam.zoneSize * Math.PI-Math.PI/2)),
curve: currentZone.curve+intendedCurveForCurrentZone / 2 * (1 + Math.sin(i/
    self.trackParam.zoneSize * Math.PI-Math.PI/2)),
sprite: sprite
})

// Updating zone height and curve for next iteration
currentZone.height += intendedHeightForCurrentZone;
currentZone.curve += intendedCurveForCurrentZone
```

Excerto de Código 13: **Mecanismo de Criação de Segmentos no *Widget TrackGenerator***

Em suma, o *widget TrackGenerator* cria segmentos de faixa a faixa, horizontal, da *Canvas* e define a topografia, o perfil de elevação e as imagens que em conjunto formam os ambientes de condução desejados a renderizar posteriormente pelo *widget Arcade*. Estas parametrizações são fornecidas pelo utilizador nos campos opcionais respectivos já apresentados anteriormente.

Cada segmento, dado pela variável *nextSegment*, é lido do *array track*, presente no ficheiro de configuração dos ambientes de condução, cujo índice é dado pela variável *nextSegmentIndex*, que, por sua vez, é calculado com base no valor do resto da divisão da variável *currentSegmentIndex* pelo comprimento desse *array track*. Para cada segmento a desenhar é necessário determinar a altura de projeção inicial e final com base na altura definida no segmento, presente no Excerto de Código 13, na altura e na distância da câmara, definida nas propriedades *camera_height* e *camera_distance* no objeto *render* fornecido como campo opcional ao *widget TrackGenerator* e na posição do segmento atual, dado por *currentSegment-*

Position, que é calculada com base no tamanho do segmento da estrada. É ainda necessário calcular a escala inicial e final a aplicar na projeção anterior. A altura atual da simulação é obtida com base no valor mínimo entre a última altura projectada e a altura de projeção inicial do novo segmento a desenhar. Após estes cálculos o *widget* invoca os métodos *setColorsCanvas()*, para colorir o jogo de corridas e para delimitar os componentes de cada faixa horizontal a desenhar na *Canvas*, e *drawSegment()*, para desenhar o novo segmento na *Canvas* e adiciona as imagens que compõem os ambientes de condução (*sprites*), a desenhar neste segmento na variável *spriteBuffer*. Após desenhar o segmento dado pela variável *nextSegment* o *widget* atualiza o valor da variável *currentSegmentIndex* com o valor da variável *nextSegmentIndex*. É ainda atualizado o valor da última altura projectada com a altura de projeção do segmento que acabou de ser desenhado e incrementa-se o valor de comprimento do segmento ao valor da variável *currentSegmentPosition*, por forma a atualizar a próxima posição do segmento a renderizar.

4.7 SÍNTESE

Neste capítulo abordou-se a implementação do todos os *widgets* desenvolvidos para a plataforma *PVSio-web*, que irão permitir construir a solução proposta. Alguns desses *widgets* recorrem a bibliotecas externas *JavaScript*, que permitiram criar os menus de seleção de cores e de imagens, *sliders HTML*, incorporar cronómetros para contabilizar o tempo decorrido desde o início da simulação e controlar a simulação com controladores externos. Foi descrita a implementação dos onze *widgets* (*ButtonExternalController*, *DrawGamepad*, *Sound*, *SteeringWheel*, *VirtualKeypadController*, *LincolnMKCDashboard*, *GyroscopeController*, *GamepadController*, *Customization*, *TrackGenerator* e *Arcade*) desenvolvidos para que seja possível criar protótipos de ambientes de condução e para criar simulações interativas com esses protótipos e com protótipos de painéis de instrumentos.

5

PROTÓTIPOS DESENVOLVIDOS

A criação de protótipos, de painéis de instrumentos e dos seus contextos de condução, tornou-se possível com a biblioteca de *widgets* desenvolvidos, e já apresentados no capítulo 4. Neste capítulo ilustramos a sua utilização.

5.1 PROTÓTIPOS DO CASO DE ESTUDO A DESENVOLVER

Os protótipos a desenvolver dizem respeito ao painel de instrumentos do veículo do caso de estudo (*Lincoln MKC de 2015*) e aos ambientes de condução, que servem como contextos de utilização. Os protótipos do painel de instrumentos são os protótipos com as interfaces antes e após a correção do fabricante, Figuras 58 e 59.

O protótipo dos ambientes de condução utiliza o simulador de condução para configurar a estrada (topografia, perfil de elevação), definir os sinais de trânsito a colocar, definir as imagens a colocar na paisagem, entre outras parametrizações.

5.2 CARACTERÍSTICAS DOS PROTÓTIPOS

Ambos os protótipos deverão possuir os mesmos ambientes de condução por forma a que futuros testes e análises às duas interfaces supracitadas permitam verificar o impacto que esses ambientes têm nessas interfaces. Esses testes e análises poderão ser utilizados para estudar a ativação do modo desportivo com diferentes ambientes de condução realistas. Esse processo deverá promover a utilização dessa funcionalidade sem olhar diretamente para o botão *S*, uma vez que tal ação implicaria o desviar do olhar da estrada e dos seus meios envolventes por uns milisegundos.

O protótipo dos ambientes de condução deverá construir uma estrada plana composta por quatro curvas à esquerda e com alguns sinais de trânsito que indicam, por exemplo, velocidade máxima numa dada zona da estrada e a existência de uma curva à esquerda perigosa. Em relação à paisagem, o protótipo dos ambientes de condução deverá colocar algumas árvores e edifícios realistas diferentes. A estrada deverá ter duas faixas para que

Figura 58.: **Versão Antes da Correção do Painel de Instrumentos do Veículo do Caso de Estudo**

Figura 59.: **Versão Depois da Correção do Painel de Instrumentos do Veículo do Caso de Estudo**

seja possível simular a mudança de faixa. As linhas que separam as faixas serão linhas descontínuas para permitir essa mudança de faixa. O protótipo dos ambientes de condução deverá ainda colocar uma zona montanhosa no horizonte.

O protótipo dos ambientes de condução deverá fornecer a informação relacionada com a percentagem da distância percorrida na estrada. Também os valores de velocidade e do número de rotações por minuto deverão ser indicados textualmente e deverão coincidir, respectivamente, com os valores apresentados no velocímetro e no tacómetro.

Para terminar será necessário identificar os vários componentes presentes nas duas interfaces do painel de instrumentos do veículo do caso de estudo, nas Figuras 58 e 59, e para cada um dos componentes indicar qual o *widget* seleccionado para o representar.

O componente volante, indicado pelas setas vermelhas, será representado pelo *widget SteeringWheel*. Os componentes velocímetro e tacómetro, indicados pelas setas azuis, serão representados pelo *widget Gauge*. O componente painel de instrumentos, indicado pelas setas verdes, será representado pelo *widget LincolnMKCDashboard*. Este componente inclui a consola central, em particular os *Start/Stop* e *S*, e todas as restantes interfaces do painel de instrumentos. Este *widget* deverá colocar este componente, para as versões antes e depois da correção supracitada, numa perspectiva que permita fornecer a vista "pára-brisas". No protótipo dos ambientes de condução, estes são representados pelos *widgets TrackGenerator* e *Arcade*. Respectivamente, permitem criar e visualizar ambientes de condução realistas, construídos com imagens de boa qualidade. Para controlar o veículo no simulador de condução serão utilizados os *widgets* de controlo *VirtualKeypadController*, *GamepadController* e *GyroscopeController*. O controlo do veículo passará pela aceleração e travagem do veículo (*accelerate* e *brake*), que implica alterações nos valores da velocidade e do número de rotações por minuto e pelo estado da simulação (*pause*, *quit* e *resume*). O *widget VirtualKeypadController* é particularmente útil caso a simulação esteja a ocorrer num aparelho móvel sem sensor giroscópio. Este *widget* permitirá controlar o veículo no simulador de condução através das teclas apresentadas no teclado de setas virtual. O *widget GamepadController* permitirá controlar o veículo através dos botões do controlador externo da *PlayStation 4*. O *widget GyroscopeController* é útil caso a simulação esteja a ocorrer num aparelho móvel com sensor giroscópio. Este *widget* permitirá controlar o veículo através da mudança de orientação desse aparelho móvel.

As Figuras 56 e 60 são os *screenshots* dos protótipos *arcade*[1] de painéis de instrumentos com as interfaces antes e depois da correção do fabricante.

Os ambientes de condução foram construídos e configurados no protótipo *track*[2]. Este é um protótipo auxiliar que apenas utiliza o *widget* auxiliar *TrackGenerator*.

1 *arcade_game_simulator_full_screen_for_paper/*
2 *track_generator_simulator_for_paper/*

Figura 60.: **Aparência Visual do Protótipo do Simulador de Condução Desenvolvido com a Versão Depois da Correção do Painel de Instrumentos do Veículo do Caso de Estudo**

5.3 ESPECIFICAÇÃO FORMAL DE SUPORTE

Para que estes protótipos possam ser simulados interativamente foi necessário desenvolver a especificação formal, que determina o comportamento e o estado dos *widgets* instanciados.

Para o utilizador interagir com os protótipos é necessário criar um modelo formal *PVS*. Este modelo formal define o comportamento dos *widgets* instanciados e foi estendido a partir do modelo[14], que já tinha um estado com informações relacionadas com o controlo da velocidade e das rotações por minuto, que são visualizadas pelos *widgets Gauge*, *Speedometer* e *Tachometer*.

Uma vez que a solução proposta nesta dissertação assenta num simulador de ambientes de condução, cujo controlo pode ser realizado através de vários mecanismos, é necessário que o modelo formal possua um estado com informações relacionadas com o controlo da velocidade, controlo da direcção do veículo no simulador de condução, controlo da posição do veículo no simulador de condução, controlo do estado de reprodução dos ficheiros áudio no simulador de condução. É ainda necessário ter informações acerca do estado dos botões *Start/Stop* e *S*.

O estado do modelo formal deve ter estas informações para que os *widgets* alterem as suas interfaces em resposta a uma dada interação entre o utilizador e os protótipos. Por exemplo, após o utilizador interagir com o protótipo por forma a silenciar a reprodução dos ficheiros áudio no simulador de condução é necessário que exista um estado no modelo formal que seja fornecido ao *widget Sound*, que por sua vez, seja utilizado para alterar a interface criada por esse *widget* e para silenciar a reprodução de todos os ficheiros áudio do simulador de condução. A alteração da interface irá contemplar esse novo estado do som,

para que o utilizador visualize também o resultado dessa ação. Neste exemplo, a interface irá apresentar uma imagem com o volume em silêncio (*muted*).

Em suma, o controlo do comportamento de cada *widget* será realizado com base no conteúdo do estado do modelo formal. De seguida, apresenta-se a lista das informações supracitadas.

- **Position** - a posição vertical atual do veículo no simulador de condução.
- **PosX** - a posição horizontal atual do veículo no simulador de condução.
- **Action** - o conjunto de ações possíveis (*idle*, *acc*, *brake*, *pause*, *resume* e *quit*).
- **Direction** - o conjunto de direções do veículo no simulador de condução possíveis (*left*, *right* e *straight*).
- **Sound** - o conjunto de estados do som no simulador de condução possíveis (*unmute* e *mute*).
- **Startstopbutton** - o conjunto de estados do motor do veículo (*off* e *on*).
- **Sportmodebutton** - o conjunto de estados do modo desportivo do veículo (*off* e *on*).

Os tipos *Position* e *PosX* fornecem as informações relacionadas com o controlo da direcção do veículo no simulador de condução. O tipo *Action* indica o estado e o movimento do veículo no simulador de condução. Esse estado permite que o *widget Arcade* renderize ou os menus de início, pausa e fim, ou os ambientes de condução. O tipo *Direction* está relacionado com a direção do veículo no simulador de condução. O tipo *Sound* fornece as informações relacionadas com o controlo do som e da reprodução dos ficheiros áudio no simulador de condução. Os tipos *Startstopbutton* e *Sportmodebutton* indicam se o botão *Start/Stop* ou o botão *S* foi clicado no protótipo pelo utilizador. O Excerto de Código 14 apresenta a especificação formal que define o estado inicial da simulação. O estado formal *PVS* possui ainda a informação *steering* que é responsável pelo estado atual do volante, mais concretamente, o valor do ângulo de rotação do volante que será utilizado pelo *widget SteeringWheel*.

```
POSITION_INIT: real = 10.0
Position: TYPE = [# val: real #]
POSX_INIT: real = 0.0
PosX: TYPE = [# val: real #]
Action: TYPE = { idle, acc, brake, pause, resume, quit }
Direction: TYPE = { left, right, straight }
Sound: TYPE = { unmute, mute }
Startstopbutton: TYPE = { off, on }
Sportmodebutton: TYPE = { off, on }
state: TYPE = [#
  speed: Speed, % Km/h
  gear: Gear,
  rpm: Rpm, % x1000/min
  odo: Odo, % Km
  temp: Temp,
  time: Time,
  steering: real,
  position: Position,
  posx: PosX,
  action: Action,
  direction: Direction,
  sound: Sound,
  startstopbutton: Startstopbutton,
  sportmodebutton: Sportmodebutton
#]
init(x: real): state = (#
  speed := (# val:= IF x < MAX_SPEED THEN x ELSE MAX_SPEED ENDIF, units := kpm
     #),
  gear := N,
  rpm := 0,
  odo := 0,
  temp := (# val := TEMP_AMB, units := C #),
  time := get_current_time,
  steering := 0,
  position := (# val := POSITION_INIT  #),
  posx := (# val := POSX_INIT #),
  action := idle,
  direction := straight,
  sound := unmute,
  startstopbutton := off,
  sportmodebutton := off
#)
```

Excerto de Código 14: **Especificação do Tipo *State* na Teoria *PVS* dos Protótipos Desenvolvidos**

Então, será necessário que o modelo formal possua métodos responsáveis por atualizar as diferentes informações que o seu estado possui. Para o tipo *Action* é necessário ter métodos que atualizem o seu valor para cada um dos estados possíveis de acordo com o definido em *Action* (*idle*, *acc*, *brake*, *pause*, *resume* e *quit*). De seguida, apresentam-se as assinaturas desses métodos.

- **click_accelerate(st: state)**
- **click_brake(st: state)**
- **press_accelerate(st: state)**
- **release_accelerate(st: state)**
- **press_brake(st: state)**
- **release_brake(st: state)**
- **press_quit(st: state)**
- **release_quit(st: state)**
- **press_pause(st: state)**
- **release_pause(st: state)**
- **press_resume(st: state)**
- **release_resume(st: state)**

Uma vez que esta especificação não foi criada de raíz, mas sim estendida a partir do modelo[14], decidiu-se manter a lógica que todos os métodos são declarados como *"press/release"* e que todos os métodos *"release"* são responsáveis por atualizar o atributo *action*, do tipo *Action*, do estado formal para *idle*. Desta forma, o protótipo possui o estado *idle* sempre que o utilizador deixar de interagir com as suas interfaces. Para o tipo *Action*, os métodos para as ações *pause*, *resume* e *quit* são declarados como *"press/release"*. No entanto, para as ações *acc* e *brake* é necessário declarar métodos para o tipo *"click"* e para o tipo *"press/release"*, uma vez que o utilizador poderá acelerar de forma contínua ou simples.

Para o tipo *Sound* é necessário ter métodos que atualizem o seu valor quer para *mute*, quer para *unmute*. De seguida, apresentam-se as assinaturas desses métodos.

- **press_mute(st: state)**
- **release_mute(st: state)**
- **press_unmute(st: state)**
- **release_unmute(st: state)**

Para o tipo *Direction* é necessário ter métodos que atualizem o seu valor quer para *right*, quer para *left*, quer para *straight*. Esta ação está diretamente relacionada com a informação *steering* que indica a direção do volante (*widget SteeringWheel*). De seguida, apresentam-se as assinaturas desses métodos.

- **steering_wheel_right(st: state)**
- **steering_wheel_left(st: state)**
- **steering_wheel_straight(st: state)**
- **steering_wheel_rotate(x: real)(st: state)**

Uma vez que deverá ser possível registar a rotação do volante (*widget SteeringWheel*) com base num ângulo de rotação é necessário criar o método *steering_wheel_rotate* que recebe como argumento o ângulo, em graus. Este ângulo será, depois, fornecido e utilizado pelo método *rotate(val)* do *widget SteeringWheel* para rodar a imagem do volante no protótipo.

Para os tipos *Startstopbutton* e *Sportmodebutton* é necessário ter métodos que atualizem o seu valor quer para *on*, quer para *off*. De seguida, apresentam-se as assinaturas desses métodos.

- **press_startAndStop(st: state)**
- **release_startAndStop(st: state)**
- **press_activateSportMode(st: state)**
- **release_activateSportMode(st: state)**

Para simular o problema de desligar o veículo em andamento decidiu-se que o método *press_startAndStop* (botão *Start/Stop*) deveria também alterar o atributo *action* do estado para *quit*. Esta alteração implicará que seja apresentado o menu de fim, no simulador de condução, quando o utilizador pressionar este botão no protótipo. E para simular a ativação do modo desportivo decidiu-se que o método *press_activateSportMode* (botão *S*) deveria alterar os valores da velocidade e do número de rotações por minuto de forma mais veloz e com intervalos maiores.

No futuro, caso a plataforma *PVSio-web* pretenda realizar testes para analisar a interface, do veículo do caso de estudo, poderá utilizar esta especificação formal como ponto de partida para estudar o problema de clicar o botão *Start/Stop* quando se pretendia ativar o modo desportivo.

Este modelo formal permite controlar os *widgets* nos protótipos a desenvolver, contudo pode ser melhorado. As melhorias passam pela definição dos métodos dos tipos *Sound*, *Startstopbutton* e *Sportmodebutton* como *"click"*, por forma a representar melhor os eventos que são instantâneos. Caso fossem declarados como *"click"*, o estado após uma interação terminar passaria a ser o estado da última interação e não o estado *idle*. Para tomar o estado *idle* o modelo formal *PVS* teria de ter os métodos *"click_idle"* ou *"press_idle"*/*"release_idle"* implementados.

5.4 Configurações Necessárias dos *widgets* dos Protótipos

Nesta secção serão apresentadas todas as configurações necessárias para que a instanciação dos *widgets* descritos na secção 5.2 produza os protótipos apresentados nas Figuras 56 e 60, páginas 77 e 85.

5.4.1 *Configurações dos Widgets de Painéis de Instrumentos*

A configuração dos botões lógicos, que permitem que o utilizador interaja com os protótipos e que estabelecem a ligação com o modelo formal *PVS*, passou pela declaração de diferentes instâncias do *widget ButtonExternalController*. Para cada interação, no protótipo, é necessário criar um botão lógico, cujo primeiro argumento a identifica e implica a implementação dos respetivos métodos, apresentados na secção 5.3, no modelo formal *PVS*. Os campos opcionais destes botões lógicos apenas diferem na tecla física que está associada à interação (campo opcional *KeyCode*). O Excerto de Código 15 corresponde à instanciação do botão lógico que permite acelerar o veículo no simulador de condução. Para as interações que permitem travar o veículo, pausar/terminar/retomar a simulação e silenciar/reproduzir os sons no simulador de condução a instanciação ocorre de forma igual.

```
1 arcade.up = new ButtonExternalController("accelerate", {width: 0, height: 0},
    {callback: onMessageReceived, evts: ['press/release'], keyCode: 38})
```

Excerto de Código 15: **Instanciação do *Widget ButtonExternalController* para Simular a Aceleração**

Os Excertos de Código 16 e 17 apresentam as configurações do *widget Gauge* que permite representar um velocímetro e um tacómetro que se assemelham ao velocímetro e tacómetro do veículo do caso de estudo, Figura 61, no protótipo. Os campos opcionais *redZones* e

Figura 61.: **Aparência Visual dos *Widgets* Velocímetro e do Tacómetro Instanciados**

innerFillColor representam as configurações que têm o maior impacto na interface a representar. O campo opcional *redZones* permite definir as áreas a vermelho na Figura 61. O campo opcional *innerFillColor* permite definir a cor de fundo (*#ob0605*) dos indicadores.

```
arcade.speedometerGauge = new Gauge('speedometer-gauge', {width: 136, height:
    136, top: 0, left: 0}, { max: 260, min: 0, label: "kmh", redZones: [{from:
     240, to: 260}], innerFillColor: "#0b0605"})
```

Excerto de Código 16: **Instanciação do *Widget Gauge* para Representar o Velocímetro**

```
arcade.tachometerGauge = new Gauge('tachometer-gauge', {width: 100, height:
    100, top: 0, left: 0}, {max: 9, min: 0, label: "x1000/min", redZones: [{
    from: 7, to: 9}], innerFillColor: "#0b0605"})
```

Excerto de Código 17: **Instanciação do *Widget Gauge* para Representar o Tacómetro**

O Excerto de Código 18 apresenta a configuração do *widget SteeringWheel* que permite representar o volante do veículo do caso de estudo, Figura 62, no protótipo. Essa configuração passou pelas definições da propriedade *z-index*, que permite colocar a imagem num plano diferente (à frente da interface do *widget LincolnMKCDashboard*), e do estilo (*style*), que permite renderizar a imagem desejada.

```
arcade.steeringWheel = new SteeringWheel("steering_wheel", {top: 470, left:
    430, width: 350, height: 350}, {style: "lincoln_mkc_2015_2", "z-index": 1,
     callback: onMessageReceived})
```

Excerto de Código 18: **Instanciação do *Widget SteeringWheel* para Representar o Volante**

Os Excertos de Códigos 19 e 20 apresentam as configurações do *widget LincolnMKCDashboard* para Representar as Versões Antes e Depois do Painel de Instrumentos Completo no protótipo. As configurações apenas diferem no conteúdo do campo opcional *design*. Este campo permite escolher a imagem com a versão antes (*before*) ou com a versão depois (*after*) da correção do fabricante. Com estas configurações este *widget* coloca as interfaces

Figura 62.: **Aparência Visual do *Widget* Volante Instanciado**

apresentadas nas Figuras 37, página 46, e 38, página 46, nos protótipos. O campo opcional *buttonsPVS* define os identificadores dos botões lógicos que correspondem aos botões *Start/Stop* e *S*. O conteúdo deste campo implicará que o modelo formal *PVS* possua a implementação dos métodos apresentados na secção 5.3, por forma a que o utilizador possa interagir com estes. A grande vantagem de definir os identificadores de forma dinâmica (lista) é que os *widgets* não precisam de saber estaticamente esses identificadores para estabelecer a ligação com o modelo formal *PVS*. Desta forma o utilizador pode facilmente visualizar comportamentos distintos (métodos no modelo formal *PVS* diferentes) ao instanciar os nomes dos métodos como identificadores. Por exemplo, o modelo formal pode ter três métodos diferentes (*press_activateSportMode/release_activateSportMode*, *press_b/release_b* e *press_c/release_c*), para definir o comportamento do modo desportivo (botão *S*), e para os testar apenas terá de fornecer no campo opcional, neste caso *buttonsPVS*, o identificador desse método (*activateSportMode*, *b* ou *c*). Para os campos opcionais idênticos, nos restantes *widgets*, a lógica é igual.

```
1 arcade.lincolnMKCDashboard = new LincolnMKCDashboard('lincolnMKCDashboard', {
    top: 5, left: 0, width: 450, height: 140},{parent: "content", dashIndex:
    1,design: "before", buttonsPVS: ["startAndStop", "activateSportMode"],
    callback: onMessageReceived})
```

Excerto de Código 19: **Instanciação do *Widget LincolnMKCDashboard* para Representar a Versão Antes do Painel de Instrumentos Completo**

```
arcade.lincolnMKCDashboard = new LincolnMKCDashboard('lincolnMKCDashboard', {
    top: 5, left: 0, width: 450, height: 140},{parent: "content", dashIndex:
    1,design: "after", buttonsPVS: ["startAndStop", "activateSportMode"],
    callback: onMessageReceived})
```

Excerto de Código 20: **Instanciação do *Widget LincolnMKCDashboard* para Representar a Versão Depois do Painel de Instrumentos Completo**

5.4.2 *Configurações dos Widgets de Controlo*

As configurações dos *widgets* de controlo genericamente passam pela associação dos índices *standard* dos controladores a identificadores de métodos do modelo formal *PVS*. Esta associação permitirá que os *handlers* dos eventos capturados nesses controladores estabeleçam a ligação com o modelo formal *PVS*.

O Excerto de Código 21 apresenta a configuração do *widget VirtualKeypadController* que permite representar o teclado de setas virtual. As configurações passam pela definição do *icon* teclado e da sua posição no ecrã, das classes *jQuery UI* que permitem visualizar os *icons* das teclas virtuais e dos identificadores de cada uma dessas teclas. Estes identificadores implicarão que o modelo formal *PVS* possua a implementação dos métodos apresentados na secção 5.3.

```
arcade.virtualKeypadController = new VirtualKeypadController("
    virtualKeypad_controller", {top: 580, left: 1280, width: 750, height:
    750}, {keyboardImgDiv: "mobileDevicesController", keyboardClass: "icon
    keyboard", keyboardLeftDesktop: 1370, keyboardTopDesktop: 710,
    keyboardHoverInitialTitle: "Click to open virtual keypad controller",
    keyboardHoverSecondTitle: "Click to close virtual keypad controller",
    parent: "content", buttonsDiv: "virtualKeyPad", buttonClass: "ui-button ui
    -corner-all ui-widget ui-button-icon-only", arrowKeysPVS: ["accelerate", "
    brake", "steering_wheel_left", "steering_wheel_right"], otherKeysPVS: ["
    quit", "pause", "resume"], callback: onMessageReceived})
```

Excerto de Código 21: **Instanciação do *Widget VirtualKeypadController* para Representar o Teclado de Setas Virtual**

Com esta configuração, este *widget* representará as interfaces apresentadas na Figura 42, página 51, nos protótipos.

As configurações apresentadas nos Excertos de Código 22 e 23 passam pela definição da associação supracitada, pela definição de índices *standard* do controlador externo a utilizar (definido pelo fabricante) e por fornecer o *widget* volante instanciado anteriormente e os botões lógicos *acelerar/travar*. Estes *widgets* não representam nenhuma interface, uma vez

que ficam apenas à espera dos eventos inerentes à conexão de controladores externos (via *USB/Bluetooth*) e à mudança de orientação do dispositivo (caso possua sensor giroscópio).

Os campos opcionais que possuem *Index* dizem respeito aos índices do mapeamento *standard* definido pelo fabricante do controlador externo. Por exemplo, para o *gamepad PlayStation 4* o botão *"cross"* possui o índice zero. Caso o utilizador pretenda utilizar o botão *"cross"* para acelerar apenas terá de fornecer o valor zero no campo opcional *accelerationIndex*. Desta forma, o utilizador pode rapidamente mudar a forma como controla a simulação no mesmo *gamepad*. Por exemplo, se o utilizador desejar utilizar o botão *"circle"* para acelerar apenas terá de fornecer o valor um no campo opcional *accelerationIndex*. No entanto, antes de fazer esta alteração deve verificar se o botão *"circle"* está a ser utilizado para outra funcionalidade, neste caso estava a ser utilizado para travar (campo opcional *brakeIndex*). Caso estivesse a ser utilizado, deverá certificar-se no fim que não existem duas funcionalidades para o mesmo botão. A solução simples para ocorrer esta alteração seria utilizar o botão *"circle"* para acelerar e o botão *"cross"* para travar. Esta solução consiste na troca de valores dos campos opcionais *accelerationIndex* e *brakeIndex*. Para os restantes índices o processo é igual.

Os campos opcionais *instructionPVS* dizem respeito aos identificadores dos botões e como já foi mencionado anteriormente esta configuração permite facilmente alterar a associação entre o botão do controlador externo e a interação desejada. Por exemplo, para pausar a simulação (*pauseAction*) esta configuração define o índice nove (botão *options* no *gamepad PlayStation 4*) e implica a implementação do método *press_pause/release_pause* no modelo formal *PVS*, uma vez que fornece o identificador *pause*. Caso o utilizador pretenda utilizar o método *press_pause2/release_pause2*, apenas terá de colocar o identificador *"pause2"* no campo opcional *instructionPVS* em *pauseAction*.

```
1 arcade.gamepadController = new GamepadController("gamepad_controller", {top:
    1000, left: 100, width: 750, height: 750}, {carAccelerate: arcade.up,
    carBrake: arcade.down, carSteeringWheel: arcade.steeringWheel,
    accelerateInstructionPVS: "accelerate", brakeInstructionPVS: "brake",
    steeringWheelInstructionPVS: "steering_wheel", type: "gamepad",
    accelerationIndex: 0, brakeIndex: 1, leftArrowIndex: 14, rightArrowIndex:
    15, leftAnalogueIndex: 0, rightAnalogueIndex: 2, pauseAction: {pauseIndex:
     9, instructionPVS: "pause"}, quitAction: {quitIndex: 8, instructionPVS: "
    quit"}, resumeAction: {resumeIndex: 16, instructionPVS: "resume"},
    muteAction: {muteIndex: 4, instructionPVS: "mute"}, unmuteAction: {
    unmuteIndex: 5, instructionPVS: "unmute"}, useSensitivity: false, callback
    : onMessageReceived})
```

Excerto de Código 22: **Instanciação do *Widget GamepadController***

```
arcade.gyroscopeController = new GyroscopeController("Gyroscope_Controller", {
    top: 100, left: 700, width: 750, height: 750},{parent: "gyroscope",
    carSteeringWheel: arcade.steeringWheel, carAccelerate: arcade.up, carBrake
    : arcade.down, useSensitivity: false, callback: onMessageReceived})
```

Excerto de Código 23: **Instanciação do *Widget GyroscopeController***

5.4.3 *Configurações dos Widgets Auxiliares*

O Excerto de Código 24 apresenta a configuração do *widget Sound* instanciada pelo construtor do *widget Arcade* para reproduzir sons no simulador de condução. Esta instanciação ocorre no construtor por forma a garantir que múltiplas instanciações do *widget Arcade* possuem múltiplas instanciações do *widget Sound*. Desta forma, cada simulador de condução terá o seu próprio mecanismo de reprodução de sons.

As configurações do *widget Sound*, nestes protótipos, passam pela definição da posição relativa ao *widget Arcade*, pelos identificadores dos botões responsáveis por silenciar e reproduzir sons e pelos ficheiros áudio a reproduzir (campo opcional *songs*). Para cada ficheiro áudio é necessário fornecer a diretoria onde se encontra e definir se será reproduzido em *loop* ou se é reproduzido uma vez de início ao fim.

```
this.soundWidget = new Sound("soundWidget_"+this.id, {top: (this.top+this.
    height-30), left: (this.left-30), width: 750, height: 750}, {
parent: this.parent.slice(1), callback: opt.callback, invokePVS: true,
    mute_functionNamePVS: "mute", unmute_functionNamePVS: "unmute", soundOff:
    "false",
songs: [{url: "../../client/app/widgets/car/configurations/song/sound.mp3",
    loop: false}, {url: "../../client/app/widgets/car/configurations/song/loop
    .mp3", loop: true}, {url: "../../client/app/widgets/car/configurations/
    song/car_startup.mp3", loop: false}, {url: "../../client/app/widgets/car/
    configurations/song/car_accelerating.mp3", loop: false}]})
```

Excerto de Código 24: **Instanciação do *Widget Sound* para Reproduzir Sons no Simulador de Condução**

Este *widget* é responsável por adicionar as interfaces apresentadas na Figura 47, página 61, nos protótipos supracitados.

O Excerto de Código 25 apresenta a configuração do *widget TrackGenerator* que permite criar os ambientes de condução com as características apresentadas na secção 5.2. As configurações passam pela definição do ficheiro *JSON* com o mapeamento de coordenadas das imagens presentes na *spritesheet* que se pretende utilizar, das opções de renderização (distância da câmara, etc), do número de faixas, pela configuração da estrada, dos objetos

(paisagem) e dos obstáculos. Caso se pretenda configurar obstáculos será necessário definir também a frequência com que estes são colocados na estrada.

Para definir uma estrada plana composta por quatro curvas à esquerda e com alguns sinais de trânsito é necessário definir o comprimento da estrada no campo opcional *trackParam*. Neste caso, a propriedade *numZones* toma o valor quatro, uma vez que são quatro curvas e a propriedade *zoneSize* toma o valor 250, que é o valor por omissão (comprimento de cada uma dessas quatro zonas). É também necessário descrever essas quatro zonas com mais detalhe no campo opcional *trackLayout* com a topografia esquerda (propriedade *name*), com uma curvatura de -90 graus (propriedade *curvature*), com perfil de elevação plano (propriedade *profile*), com quatro zonas consecutivas iguais (propriedade *numZones*) e com uma lista de sinais de trânsito na propriedade *trafficSignals*. Cada posição da lista *trafficSignals* diz respeito a um dado sinal de trânsito, cujo nome da imagem na *spritesheet* é dada pela propriedade *filename*, que se pretende colocar numa dessas quatro zonas (propriedade *zone*), com uma dada escala (propriedade *scale*), numa dada posição horizontal no simulador de condução (propriedade *posX*) a uma distância em relação ao campo de visão (propriedade *zoneDistance*). Para terminar a configuração da estrada é necessário definir o número de faixas a representar no campo opcional *numLanes*, com uma dada largura (campo opcional *laneWidth*). Neste protótipo a estrada possui duas faixas, então é necessário atribuir o valor dois a *numLanes*.

Para configurar a paisagem composta por algumas árvores e edifícios realistas diferentes é necessário configurar o campo opcional *objects*. Este campo é uma lista com todas as imagens que se pretendem colocar. Cada posição da lista *objects* diz respeito a uma dada imagem (árvore, etc), cujo nome da imagem na *spritesheet* é dada pela propriedade *filename*. Essa imagem será colocada com uma escala fornecida na propriedade *scale* numa dada posição horizontal (*positionsX*).

Uma vez que os ambientes de condução nos protótipos não possuem obstáculos o utilizador não precisa de preencher o campo opcional *obstacle*. No entanto, a configuração destes realiza-se da mesma forma que a configuração das imagens que compõem a paisagem.

O conteúdo das propriedades *filename* é necessário para filtrar as coordenadas da imagem desejada no ficheiro *JSON* com o mapeamento de coordenadas das imagens presentes na *spritesheet* (campo opcional *spritesFilename*).

Para terminar a configuração dos ambientes de condução é apenas necessário definir as cores da estrada, da linha início/fim, dos delimitadores, entre outras, que podem ser observadas nas Figuras 56 e 60, páginas 77 e 85. Esta configuração ocorre no campo opcional *trackColors*. Por exemplo, a propriedade *grass1* permite definir a cor da "relva", na paisagem no simulador de condução. Caso o utilizador deseje visualizar os ambientes de condução num cenário que se aproxima de um deserto será apenas necessário modificar a cor verde desta propriedade (atual) para uma cor amarelo escuro.

```
trackGenerator.trackGeneratorWidget = new TrackGenerator("trackGeneratorWidget
    ", {top: 80, left: 650, width: 780, height: 650}, {parent: "content",
    spritesFilename: "spritesheet4", render: {depthOfField: 150,
    camera_distance: 30, camera_height: 320}, numLanes: 2, laneWidth: 0.02,
 trackParam: {numZones: 4, zoneSize: 250},
objects: [{filename:"real_tree", scale: 3.5, positionsX: [-2.4, 2.3]}, {
    filename:"real_tree2", scale: 3.5, positionsX: [-2.9, 4.2]}, {filename:"
    real_tree3", scale: 3.5, positionsX: [-1.8, 1.6]}, {filename:"real_tree4",
     scale: 3.5, positionsX: [-1.6, 1.8]}, {filename:"real_building", scale:
    6, positionsX: [-1.7, 1.9]}, {filename:"real_building2", scale: 6,
    positionsX: [-1.9, 1.7]}, {filename:"real_skyscraper", scale: 7,
    positionsX: [2.9, -2.7]}], trackColors: {grass1: "#344C32", border1: "#
    ffa500", border2: "#ffffff", outborder1: "#7F967D", outborder_end1: "
    #474747", track_segment1: "#777777", lane1: "#ffffff", lane2: "#777777",
    laneArrow1: "#ffff00", track_segment_end:"#000000", lane_end: "#ffffff"},
    trackLayout: [{topography: {name:"left", curvature: -90}, profile: "flat",
     numZones: 4, trafficSignals: [{filename:"traffic_light_green", zone: 1,
    scale: 4, posX: -0.5, zoneDistance: 5}, {filename:"dangerous_curve_left",
    zone: 1, scale: 3, posX: -0.4, zoneDistance: 20}, {filename:"50kmh_limit",
     zone: 1, scale: 3, posX: -0.4, zoneDistance: 90}, {filename:"
    vehicle_surpass_forbidden", zone: 1, scale: 3, posX: -0.4, zoneDistance:
    130}, {filename:"dangerous_curve_left", zone: 2, scale: 3, posX: -0.4,
    zoneDistance: 20}, {filename:"30kmh_limit", zone: 2, scale: 3, posX: -0.4,
     zoneDistance: 90}, {filename:"dangerous_curve_left", zone: 3, scale: 3,
    posX: -0.4, zoneDistance: 20}, {filename:"50kmh_limit", zone: 3, scale: 3,
     posX: -0.4, zoneDistance: 90}, {filename:"dangerous_curve_left", zone: 4,
     scale: 3, posX: -0.4, zoneDistance: 20}, {filename:"30kmh_limit", zone:
    4, scale: 3, posX: -0.4, zoneDistance: 90}, {filename:"traffic_light_red",
     zone: 4, scale: 4, posX: -0.5, zoneDistance: 100}]}], callback:
    onMessageReceived})
```

Excerto de Código 25: **Instanciação do *Widget TrackGenerator* para Configurar e Criar os Ambientes de Condução**

Após a configuração dos ambientes de condução será apenas necessário invocar o método *generateTrackBasedOnTrackLayoutOptField()* para que esses sejam criados com base nesses parâmetros. Atualmente, o resultado apenas pode ser observado na consola do *browser* devido à inexistência de uma API de escrita de ficheiros no contexto de *widgets* na plataforma *PVSio-web* e, como tal, a criação do ficheiro *JSON* com estes ambientes de condução terá de ser realizada manualmente.

5.4.4 *Configuração do Widget de Animação*

O Excerto de Código 26 apresenta a configuração do *widget Arcade* que permite visualizar os ambientes de condução criados com o *widget TrackGenerator*. As configurações

Figura 63.: **Propriedades do Campo Opcional *stripePositions* na Faixa Horizontal do Simulador de Condução**

passam por definir as dimensões da *Canvas* (*width: 1440, height: 650*), os ficheiros de configuração *JSON* com os ambientes a renderizar (*trackFilename*) e com o mapeamento das coordenadas da *spritesheet* (*spritesFilename*), as imagens *spritesheet* com as imagens e com a fonte *arcade* (texto) a utilizar, o tipo de veículo, se o veículo é renderizado, as posições de uma faixa horizontal da pista completa e os nomes dos atributos e propriedades do estado do modelo formal *PVS*, que permitem de forma dinâmica controlar a simulação.

O campo opcional *spritesFilename* terá de receber o mesmo ficheiro *JSON* que foi utilizado na instanciação do *widget TrackGenerator*, por forma a garantir que as coordenadas presentes no ficheiro de configuração fornecido ao campo opcional *trackFilename* coincidam com as coordenadas presentes no ficheiro *JSON* com o mapeamento das coordenadas da *spritesheet* a utilizar. Salienta-se que o ficheiro de configuração fornecido ao campo opcional *trackFilename* resulta da escrita manual do conteúdo gerado pela invocação do método *generateTrackBasedOnTrackLayoutOptField()* na consola do *browser*. Para garantir que o simulador renderiza as imagens configuradas nos ambientes de condução pelo *widget TrackGenerator* é necessário que a primeira imagem fornecida ao campo opcional *spritesFiles* (posição zero da lista) seja a *spritesheet* a que o ficheiro *JSON* com o mapeamento de coordenadas se refere. A *spritesheet* fornecida nesta posição pode ser consultada no Anexo F. A segunda imagem fornecida ao campo opcional *spritesFiles* (posição um da lista) apenas será necessário fornecer caso se pretenda que o simulador de condução renderize fontes e imagens do tipo *arcade*, que são menos realistas (campo opcional *realisticImgs* a falso).

Os campos opcionais *useVehicle* e *vehicle* permitem definir se a imagem do veículo será renderizada no simulador e qual o tipo de veículo. Nestes protótipos, como se pretende ter a perspectiva "pára-brisas" o campo opcional *useVehicle* toma o valor falso.

O campo opcional *stripePositions* permite definir as posições horizontais onde começa/-termina e/ou a largura de cada porção da faixa horizontal no simulador de condução. Por exemplo, as propriedades *borderWidth*, *inOutBorderWidth* e *landscapeOutBorderWidth* permitem definir largura dos delimitadores e da berma, Figura 63. Desta forma, é possível ter os mesmos ambientes de condução (estrada e paisagem) e ter uma visualização ligeiramente diferente ao modificar as larguras dessas porções.

Para terminar as configurações deste *widget* é necessário fornecer os nomes dos atributos e das suas propriedades do estado do modelo formal *PVS*. Estas configurações ocorrem nos campos opcionais *_attribute* e *_value*. Por exemplo, o campo opcional *direction_attribute*

permite que o simulador utilize o seu conteúdo para saber a direção atual do veículo, de forma dinâmica, no estado no modelo formal *PVS*. Desta forma, é possível alterar o nome do atributo do estado do modelo formal *PVS* sem ter a necessidade de alterar manualmente o nome na implementação do *widget*.

O campo opcional *newLap_functionNamePVS* permite que o utilizador defina o nome do método no modelo formal *PVS* responsável por criar uma nova volta no simulador, caso o utilizador pretenda ter uma simulação com um número de voltas finito.

```
arcade.arcadeWidget = new Arcade("arcadeWidget", {top: 0, left: 0, width:
    1440, height: 650},{parent: "content", scaleWindow: 1, trackFilename: "
    trackLayout_real", spritesFilename: "spritesheet4", spritesFiles: ["
    spritesheet4","spritesheet.text"], realisticImgs: true, useVehicle: false,
     vehicle: "car", logoImgIndex: 3, stripePositions: {trackP1: -0.55,
    trackP2: 0.55, borderWidth: 0.08, inOutBorderWidth: 0.02,
    landscapeOutBorderWidth: 0.13, diffTrackBorder: 0.05, finishLineP1: -0.40,
     finishLineP2: 0.40, diffLanesFinishLine: 0.05}, loadPVSSpeedPositions:
    true, newLap_functionNamePVS: "set_positions_init", action_attribute: "
    action", direction_attribute: "direction", sound_attribute: "sound",
    speed_attribute: "speed", posx_attribute: "posx", position_attribute: "
    position", speed_value: "val", posx_value: "val", position_value: "val",
    left_attribute: "left", right_attribute: "right", straight_attribute: "
    straight", accelerate_attribute: "acc", brake_attribute: "brake",
    idle_attribute: "idle", quit_attribute: "quit", pause_attribute: "pause",
    resume_attribute: "resume", mute_attribute: "mute", unmute_attribute: "
    unmute", callback: onMessageReceived})
```

Excerto de Código 26: **Instanciação do *Widget Arcade* para Visualizar os Ambientes de Condução**

Após a configuração dos ambientes de condução será apenas necessário invocar o método *startSimulation()* para que o simulador realize a leitura dos ficheiros de configuração e que inicie o simulador de condução. A velocidade, número de rotações por minuto, a direção do veículo, entre outras informações são atualizadas com a invocação do método *render(res)* do próprio *widget Arcade*.

5.4.5 *Renderização dos Widgets Instanciados*

Para se visualizar as configurações acima será necessário definir uma função *render* que se usa em todos os protótipos e que invoca as devidas APIs dos *widgets* instanciados. O Excerto de Código 27 apresenta a implementação dessa função.

```
let firstResume = 0;
function render(res) { // res is the Current PVS State
  if(res.action==="resume"){
    firstResume = 1;
    $("#mobileDevicesController_virtualKeypad_controller").css("visibility", "
    visible");
  }
  if(res.action==="pause" || res.action==="quit"){
    $("#arcadeSimulator_arcadeWidget").css("height", "770px");
    firstResume = 0;
    // Hide Case Study Dashboard Image
    arcade.lincolnMKCDashboard.hide();
    $("#gauges").css("visibility", "hidden");
    arcade.steeringWheel.hide();
    arcade.virtualKeypadController.hide();
  }
  if(res.action!=="pause" && res.action!=="quit" && firstResume===1){
    $("#arcadeSimulator_arcadeWidget").css("height", "650px");
    // Render Case Study Dashboard Image
    arcade.lincolnMKCDashboard.render();
    $("#gauges").css("visibility", "visible");
    arcade.virtualKeypadController.reveal();
    // Overlapping VirtualKeypadController in relation to Arcade Simulator
    $("#mobileDevicesController_virtualKeypad_controller").css("z-index", "1")
    ;
    $("#virtualKeyPad_virtualKeypad_controller").css("z-index", "1");
    $("#simulatorArrows").css("margin-top", "-55px");
    $("#simulatorArrows").css("margin-left", "20px");
    arcade.speedometerGauge.render(evaluate(res.speed.val));
    arcade.tachometerGauge.render(evaluate(res.rpm));
    arcade.steeringWheel.render(evaluate(res.steering));
  }
  arcade.gamepadController.render();
  arcade.gyroscopeController.render();
  arcade.arcadeWidget.render(res);
}
```

Excerto de Código 27: **Renderização dos *Widgets* Instanciados com base no Estado Atual do Modelo Formal *PVS***

Esta função consiste em analisar os atributos do estado *PVS*, apresentado na secção 5.2, para decidir qual o método do *widget* a invocar. Salienta-se que esta implementação é mais complexa que o normal, uma vez que nos protótipos implementados decidiu-se que quando os menus de início, pausa e fim estivessem a ser visualizados no ecrã as interfaces dos restantes *widgets* deveriam estar invisíveis e que quando a simulação fosse retomada é que

essas interfaces estariam visíveis. A variável *firstResume* permite distinguir a renderização do menu de início do menu de pausa, uma vez que estes menus são "desenhados" pelo mesmo método, no *widget Arcade*. Esta decisão permite ter uma simulação mais parecida com os jogos de corridas atuais, onde as interfaces relacionadas com o acto de conduzir apenas são apresentadas durante a condução.

Os métodos dos *widgets* invocados pelo utilizador são os métodos *render()*, *render(res)*, *hide()* e *reveal()*, que correspondem com os métodos comuns apresentados na secção 4.1. Os métodos especifícos geralmente são invocados imediatamente a seguir à instanciação do *widget*.

O método *evaluate* permite interpretar o valor numérico do modelo formal *PVS* e retornar o valor numérico correspondente na linguagem *JavaScript*, uma vez que o modelo formal *PVS* envia os valores como *strings*.

Para comprovar a flexibilidade da solução proposta nesta dissertação foram desenvolvidos outros protótipos menos relevantes. A Figura 54, na página 76, é um *screenshot* do protótipo *old_arcade*[3], cujos ambientes de condução foram parametrizados e criados no protótipo *old_track*[4]. O protótipo *simple*[5] instancia o *widget Customization* para configurar, de forma dinâmica, uma simulação de condução sem conhecimentos de programação, sendo apenas necessário o preenchimento de campos de texto, seleção de cores e de imagens. No capítulo 6 estes protótipos *PVSio-web* serão discutidos mais um pouco.

5.5 SÍNTESE

Este capítulo apresentou os protótipos desenvolvidos, bem como as suas características e a especificação formal, que suporta as simulações interativas desses protótipos. Estes protótipos permitem simular as duas interfaces do painel de instrumentos do veículo do caso de estudo, que apenas diferem na localização dos botões *Start/Stop* e *S*. Demonstrou ainda como se utilizaram os *widgets* desenvolvidos e explicados no capítulo 4 nesses protótipos.

3 *old_appearance_arcade_game_simulator/*
4 *old_appearance_track_generator_simulator/*
5 *driving_simulator*

6

CONCLUSÃO

Neste capítulo são apresentadas conclusões e possíveis trabalhos futuros. As conclusões abordam conceitos como falhas de *design* de painéis de instrumentos de veículos, prototipagem de ambientes de condução, prototipagem rápida, bem como a abordagem da solução proposta e sua implementação.

Os protótipos podem ser analisados com ou sem os seus futuros contextos de utilização. Em alguns casos, os protótipos são mais úteis se forem analisados no seu contexto de utilização, uma vez que permitem detetar potenciais falhas ou lapsos de *design* na interface, que apenas seriam detetadas no sistema físico num dado contexto de utilização. Um simulador de condução possui algumas vantagens. Por exemplo, observar que às vezes existem detalhes nos *designs* de painéis de instrumentos que deviam ser alterados devido à ocorrência de problemas durante as interações entre os utilizadores e a interface em questão. E verificar a diferença das ações a realizar para uma dada tarefa em *designs* antes e após a deteção de problemas sérios. Nomeadamente, para o painel de instrumentos do caso de estudo do veículo Lincoln MKC de 2015, será útil para observar as diferenças de ativar o modo desportivo na interface antes de qualquer queixa ter sido apresentada ao fabricante, e na interface após o fabricante ter recolhido os veículos e ter modificado as suas interfaces, por forma a resolver o problema reportado.

A criação de um simulador de condução surge da necessidade de criar um mecanismo de avaliação de protótipos de painéis de instrumentos num contexto de condução. A análise de interfaces de protótipos de painéis de instrumentos poderá ser realizada com base em diversas interações homem-máquina numa simulação interativa virtual, em conjunto com protótipos de ambientes de condução. A solução proposta nesta dissertação permite criar e visualizar algum contexto no caso de protótipos de painéis de instrumentos, desenvolvidos em *PVSio-web*, com baixo custo de desenvolvimento e utilização em oposição a simuladores com capacidades muito maiores de imersividade/fidelidade.

O simulador de condução poderá também vir a ser utilizado para ajudar a demonstrar a importância de considerar os contextos de utilização no processo de criação de HMIs, onde se poderão realizar testes específicos por forma a que seja possível observar o comporta-

mento de uma dada interface de um painel de instrumentos em ambientes de condução diferentes.

Como ponto de partida, o *toolkit* de prototipagem rápida, *PVSio-web*, disponibilizou os protótipos de um velocímetro e de um tacómetro. Os comportamentos desses protótipos já estavam especificados com a linguagem formal *PVS*. De seguida apresenta-se a lista de objetivos propostos nesta dissertação, bem como esses foram atingidos.

1. **Dotar o *toolkit* de prototipagem rápida, *PVSio-web*, de um mecanismo que permite criar ambientes de condução.** Este objetivo foi atingido com a implementação de um *widget* indepedente, *TrackGenerator*, que permite criar os ambientes de condução, para um jogo de corridas 2D, com base em parâmetros fornecidos pelo utilizador que permitem configurar a topografia, o perfil de elevação da pista e configurar as imagens que formam os ambientes de condução. Uma pista poderá ser constituída por linhas rectas, por curvas à esquerda e/ou por curvas à direita, com inclinação ascendente ou descendente ou sem inclinação (plano). Este *widget* poderá ser melhorado para que a escrita do ficheiro de configuração, com esses ambientes, seja automático. Esta melhoria está relacionada com a inexistência de uma API de escrita de ficheiros, no contexto de *widgets* na plataforma *PVSio-web*. Quando tal API existir será necessário que esta devolva o nome do ficheiro de configuração *JSON*, com o conteúdo do objeto *generatedTrack*, para que possa ser fornecido no respectivo campo opcional do *widget Arcade*. Atualmente o ficheiro de configuração *JSON* tem de ser escrito manualmente com o conteúdo que está a ser escrito na consola do *browser*.

2. **Dotar o *toolkit* de prototipagem rápida, *PVSio-web*, de um mecanismo que permite visualizar ambientes de condução.** Este objetivo foi atingido com a implementação de um *widget* indepedente, *Arcade*, que permite visualizar os ambientes de condução parametrizados num ficheiro de configuração *JSON* num jogo de corridas. Para além das configurações presentes no ficheiro supracitado é possível parametrizar outras opções de visualização, como o tipo de veículo.

3. **Integração de controladores externos no *toolkit* de prototipagem rápida, *PVSio-web*.** Este objetivo foi atingido com a implementação do *widget GamepadController* que permite controlar o veículo no simulador de condução com controladores externos mais simples, como os *gamepads* da *Playstation 4* e *XBOX One*, e com controladores externos mais complexos, como o controlador externo *Logitech G29 Driving Force Racing Wheel and Pedals*, que é um controlador que possui um volante e pedais bastante realistas.

4. **Dotar o *toolkit* de prototipagem rápida, *PVSio-web*, de um teclado de setas virtual que permite controlar uma simulação interativa.** Este objetivo foi atingido com a implementação do *widget VirtualKeypadController* que permite controlar o veículo no

simulador de condução com um teclado de setas virtual, cujas interações com o utilizador se processam da mesma forma que com um teclado de setas físico. Este *widget* foi desenvolvido para dispositivos móveis sem sensor de giroscópio e cuja utilização de um controlador externo não é possível.

5. **Dotar o *toolkit* de prototipagem rápida, *PVSio-web*, de um mecanismo que permite obter uma "vista pára-brisas" simples.** Este objetivo foi atingido com a implementação do *widget LincolnMKCDashboard* que em conjunto com o *widget Arcade* permite que colocar o utilizador numa perspetiva que pareça com que esse esteja ao volante do veículo com o painel de instrumentos à sua frente e com a pista e os ambientes de condução à sua frente e à sua volta, que permite realizar uma simulação de uma forma mais realista.

6. **Dotar o *toolkit* de prototipagem rápida, *PVSio-web*, de um mecanismo que permite simular um qualquer *design* de um volante de um veículo.** Este objetivo foi atingido com a implementação do *widget SteeringWheel* que permite receber uma qualquer imagem de um volante de um veículo e visualizar o resultado de uma dada interação. Por exemplo, observar a rotação da imagem para a direita após o utilizador clicar numa tecla que é responsável por rodar o volante para a direita.

7. **Criação de uma simulação interativa com o painel de instrumentos do veículo do caso de estudo.** Este objetivo foi atingido com a criação do protótipo *arcade_game_simulator_full_screen_for_paper*. Este protótipo consiste na simulação interativa de um protótipo de um painel de instrumentos que recria o *design* e as funcionalidades *start/stop* e *activate sport mode* do painel de instrumentos do veículo do caso de estudo, antes e depois da recolha do fabricante, e do protótipo com os ambientes de condução desejados. Este protótipo, posteriormente, em processos de teste com utilizadores poderá permitir observar o impacto que os ambientes de condução têm na ativação do modo desportivo, nas duas interfaces do painel de instrumentos do veículo do caso de estudo, sendo que essas interfaces dizem respeito à interface com a falha de *design* reportada pelos proprietários e à interface com a posterior correção pelo próprio fabricante.

Foram ainda desenvolvidos os quatro *widgets* que se seguem

1. ***DrawGamepad*** Este *widget* dota o *toolkit* de prototipagem rápida, *PVSio-web*, de um controlador externo virtual para controlar o veículo no simulador de condução, num dispositivo móvel.

2. ***GyroscopeController*** Este *widget* dota o *toolkit* de prototipagem rápida, *PVSio-web*, de um controlador que utiliza o sensor de giroscópio para controlar o veículo no simulador de condução, num dispositivo móvel que possua tal sensor.

3. ***Sound*** Este *widget* dota o *toolkit* de prototipagem rápida, *PVSio-web*, de um leitor de ficheiros áudio que permite reproduzir e interagir com músicas no contexto de *widgets*.

4. ***Customization*** Este *widget* dota o *toolkit* de prototipagem rápida, *PVSio-web*, de um mecanismo que visa automatizar o processo de criação de uma simulação interativa de um protótipo de um simulador de condução com base no preenchimento de campos de texto e na seleção de cores e imagens.

Para além dos objetivos propostos, esta dissertação criou um conjunto de tutoriais e de documentações, em português e em inglês, que visam auxiliar futuras melhorias na solução proposta, nomeadamente nos processos de teste que visam comprovar o verdadeiro valor do simulador de condução desenvolvido. Outros protótipos foram criados, para testar a versatibilidade da solução. Esses protótipos apresentam-se de seguida

1. ***track_generator_simulator_for_paper*** Este protótipo permite criar uma simulação interativa que permite criar os ambientes de condução, com uma aparência visual mais realista, visualizados nos protótipos *arcade_game_simulator* e *arcade_game_simulator_-full_screen_for_paper*.

2. ***old_appearance_track_generator_simulator*** Este protótipo permite criar uma simulação interativa que permite criar os ambientes de condução, com uma aparência visual semelhante aos jogos de *arcade* antigos, visualizados no protótipo *old_appearance_arcade_game_simulator*.

3. ***old_appearance_arcade_game_simulator*** Este protótipo permite criar uma simulação interativa de um simulador de condução com uma aparência semelhante aos jogos de *arcade* antigos.

4. ***arcade_game_simulator*** Este protótipo permite criar uma simulação interativa de um simulador de condução, com um *gamepad* virtual, com um teclado virtual, com um volante e com um controlador externo, que será conectado ao computador via *USB*. Desta forma é possível testar todos os diferentes controladores no mesmo simulador de condução e observar que as posições do veículo no jogo de corridas altera com base nas interações realizadas em qualquer um desses controladores.

5. ***arcade_game_simulator_shared_across_instances*** Este protótipo permite criar uma simulação interativa com três instâncias diferentes do simulador de condução, para que seja possível verificar que não existe nenhum problema de variáveis partilhadas entre as diferentes instâncias do mesmo *widget*.

6. ***driving_simulator*** Este protótipo permite criar uma simulação interativa de um painel de instrumentos e seus componentes e de um protótipo de ambientes de condução

para utilizadores com poucos ou nenhuns conhecimentos de programação. Este processo consiste no preenchimento de campos de texto e na seleção de cores e imagens e que de forma automática cria a respectiva simulação interativa. Contudo, este protótipo pode ser melhorado. Esta melhoria está relacionada com a melhoria apresentada para o *widget TrackGenerator* e diz respeito à inexistência de uma API de escrita de ficheiros, no contexto de *widgets* na plataforma *PVSio-web*.

7. ***gamepads_simulator*** Este protótipo permite testar um protótipo construído com os *gamepads* virtuais da *PlayStation 4* e da *XBOX One*.

8. ***sound_simulator*** Este protótipo permite simular um leitor de ficheiros áudio.

Em suma, esta proposta confere ao *toolkit* de prototipagem *PVSio-web* a capacidade de criar e de testar diferentes protótipos de painéis de instrumentos com diferentes protótipos de ambientes de condução, cujas configurações incluem informações acerca da topografia e do perfil de elevação da estrada. O Anexo B apresenta o *link* da documentação *online* relacionada com a criação e utilização de um protótipo de simulador de condução.

6.1 TRABALHO FUTURO

Após a incorporação destes *widgets* na biblioteca oficial da plataforma *PVSio-web*, qualquer utilizador poderá utilizar a ferramenta *Prototype Builder* para configurar (*Builder View*) e para simular (*Simulator View*). Atualmente, a criação de protótipos com os *widgets* desenvolvidos nesta dissertação apenas pode ser realizada criando os ficheiros de configuração manualmente.

Um possível trabalho futuro é a realização de testes com utilizadores para determinar o verdadeiro valor dos protótipos desenvolvidos nesta dissertação. Um dos primeiros testes poderá passar pela validação da deteção da falha de *design*, reportada pelos proprietários do veículo do caso de estudo, nos protótipos apresentados no capítulo 5.

Um outro trabalho futuro possível é dotar a plataforma *PVSio-web* de uma API de escrita e manipulação de ficheiros automática no contexto de *widgets*.

Outra possibilidade de trabalho futuro poderá ser a criação de um *widget* que simplificará a instanciação dos campos opcionais *Widget TrackGenerator*. Este novo *widget* poderá disponibilizar uma interface gráfica onde o utilizador poderá desenhar a pista, com as diferentes topografias e perfis de elevação, e desenhar os ambientes de condução que pretendem visualizar. E que, posteriormente, deverá interpretar esses desenhos e instanciar o *Widget TrackGenerator* com as parametrizações interpretadas nos respectivos campos opcionais .

Para terminar, um possível trabalho futuro mais complexo poderá ser o desenvolvimento de uma versão 3D do simulador de condução. Essa versão consistiria na implementação de protótipos de ambientes de condução com maior fidelidade.

REFERÊNCIAS

[1] A. K. Dey; G. D. Abowd and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16:107–108, 8 2001.

[2] K. Cagle; A. Bellamy-Royds and D. Storey. *Using SVG with CSS3 and HTML5. Vector Graphics for Web Design*. O'Reilly Media, 2017.

[3] Committee CH/62/1. Medical devices. application of usability engineering to medical devices. Standard BS EN 62366:2008, 2008.

[4] P. Masci; P. Oladimeji; Y. Zhang; P. Jones; P. Curzon and H.Thimbleby. Pvsio-web 2.0: Joining pvs to hci. *Computer Aided Verification, pages 470-478, Springer International Publishing*, 2015.

[5] P. Oladimeji; P. Masci; P. Curzon and H.Thimbleby. Pvsio-web: a tool for rapid prototyping device user interfaces in pvs. *Proceedings of the 5th International Workshop on Formal Methods for Interactive Systems (FMIS 2013)*, 2013.

[6] H. Djirdeh. *Fullstack Vue: The Complete Guide to Vue.js Paperback*. 2018.

[7] E. Elliott. *Programming JavaScript Applications*. 2014.

[8] K. Hennessy and C. Arora. *Angular 6 by Example*. Packt Publishing, 2018.

[9] J. Hoc. From human–machine interaction to human–machine cooperation. *Ergonomics*, 43(7):833–843, 2000. PMID: 10929820.

[10] A. Osman; H. Baharin; M. H. Ismail and K. Jusoff. Paper prototyping as a rapid participatory design technique. *Computer and Information Science*, 2(3):53–57, 2009.

[11] P. Mallozzi. *Design and development of a co-simulation library for the PVSio-web prototyping tool, pages 3-6*. PhD thesis, 2015.

[12] P. Mallozzi. *Design and development of a co-simulation library for the PVSio-web prototyping tool, pages=86–87*. PhD thesis, 2015.

[13] A. Accomazzo; N. Murray and A. Lerner. *Fullstack React: The Complete Guide to ReactJS and Friends Paperback*. 2017.

[14] H. Pacheco. A library of user interface widgets prototypes for car dashboards, pages=32-35, 2017.

[15] D. Pogue. Automobile dashboard technology is simply awful. *Scientific American*, 318(4), 2018.

[16] A. Shankar. *Pro HTML5 Games: Learn to Build your Own Games using HTML5 and JavaScript*. Apress, 2017.

[17] S. Owre; J. M. Rushby; N. Shankar. Pvs: A prototype verification system, volume. 607, lncs. 2001.

[18] J. C. Campos; T. Abade; J. Silva and M. D. Harrison. Don't go in there! using the apex framework in the design of ambient assisted living systems. *Journal of Ambient Intelligence and Humanized Computing*, 8(4):551–566, 8 2017.

[19] B. Smus. *Web Audio API: Advanced Sound for Games and Interactive Apps*.

A

PROTÓTIPO PVSIO-WEB DO DISPOSITIVO MÉDICO RADICAL-7 PULSE CO-OXIMETER

Este apêndice apresenta a teoria do modelo *PVS* do protótipo *PVSio-web* do dispositivo médico *Radical-7 Pulse CO-Oximeter*.

```
main: THEORY
  BEGIN

  Alarm: TYPE = { off, mute, alarm }

  state: TYPE = [#
    id: string,
    spo2: real, % sensor data - blood oxigenation level (percentage)
    spo2_max: nonneg_real,
    spo2_min: nonneg_real,
    spo2_label: string,
    spo2_alarm: Alarm,
    spo2_fail: bool,
    rra: real, % sensor data - acoustic respiratory rate (breaths per minute,
   bpm)
    rra_max: nonneg_real,
    rra_min: nonneg_real,
    rra_label: string,
    rra_alarm: Alarm,
    rra_fail: bool,
    isOn: bool
  #]

  init(x: real): state = (#
    id := "Radical7",
    spo2 := 99,
    spo2_max := 0,
    spo2_min := 88,
    spo2_label := "SpO2",
    spo2_alarm := off,
    spo2_fail := false,
    rra := 18,
```

```
    rra_max := 30,
    rra_min := 6,
    rra_label := "RRa",
    rra_alarm := off,
    rra_fail := false,
    isOn := true
#)

init(x: real, id: string): state = (#
    id := id,
    spo2 := 99,
    spo2_max := 0,
    spo2_min := 88,
    spo2_label := "SpO2",
    spo2_alarm := off,
    spo2_fail := false,
    rra := 18,
    rra_max := 30,
    rra_min := 6,
    rra_label := "RRa",
    rra_alarm := off,
    rra_fail := false,
    isOn := true
#)

per_on(st: state): bool = true

on(st: (per_on)): state = st WITH [ isOn := NOT isOn(st) ]

click_btn_on(st: state): state =
    COND
    per_on(st) -> on(st),
    ELSE -> st
    ENDCOND

click_btn_mute(st: state): state =
    COND
    isOn(st) AND spo2_alarm(st) = alarm -> st WITH [ spo2_alarm := mute ],
    ELSE -> st
    ENDCOND

check_spo2(st: state): state =
    IF spo2_fail(st) = FALSE THEN
    IF ((spo2_max(st) > 0) AND (spo2(st) >= spo2_max(st))) OR
    ((spo2_min(st) > 0) AND (spo2(st) <= spo2_min(st)))
    THEN st WITH [ spo2_alarm := alarm ]
    ELSE st WITH [ spo2_alarm := off ] ENDIF
```

```
    ELSE st WITH [ spo2_alarm := alarm ] ENDIF
80
  check_rra(st: state): state =
82  IF rra_fail(st) = FALSE THEN
    IF ((rra_max(st) > 0) AND (rra(st) >= rra_max(st))) OR
84  ((rra_min(st) > 0) AND (rra(st) <= rra_min(st)))
    THEN st WITH [ rra_alarm := alarm ]
86  ELSE st WITH [ rra_alarm := off ] ENDIF
    ELSE st WITH [ rra_alarm := alarm ] ENDIF
88
  check_vitals(st: state): state =
90  LET st = check_spo2(st)
    IN check_rra(st)
92  tick(st: state): state =
    COND
94  isOn(st) -> check_vitals(st),
    ELSE -> st
96  ENDCOND

98  spo2_sensor_data(x: real)(st: state): state =
    st WITH [ spo2 := x, spo2_fail := FALSE ]
100
  rra_sensor_data(x: real)(st: state): state =
102  st WITH [ rra := x, rra_fail := FALSE ]
END main
```

Excerto de Código 28: **Teoria *PVS* da Demonstração *Radical7***

B

INSTALAÇÃO LOCAL DE PVSIO-WEB E UTILIZAÇÃO DO SIMULADOR DE CONDUÇÃO

B.1 INSTALAÇÃO LOCAL

Este apêndice pode ser consultado em 'https://zecarlos94.github.io/pvsio_web_2D_driving_simulator/car_tutorials_pt'.

B.2 CRIAÇÃO DO PROTÓTIPO DO SIMULADOR DE CONDUÇÃO

Este apêndice pode ser consultado em 'https://zecarlos94.github.io/pvsio_web_2D_driving_simulator/car_tutorials_pt/tutorial-Arcade.html'.

C

DOCUMENTAÇÃO DE TODOS CAMPOS OPCIONAIS DO *WIDGET GAMEPADCONTROLLER*

Este apêndice apresenta o ficheiro *PDF* com a tabela com todos os campos opcionais e com os valores por omissão referentes ao *widget GamepadController*.

`constructor(id, coords, opt)` → {GamepadController}

Constructor for the GamepadController widget.

Parameters:

Name	Type	Description
id	String	The id of the widget instance.
coords	Object	The four coordinates (top, left, width, height) of the display, specifying the left, top corner, and the width and height of the (rectangular) display. Default is { top: 1000, left: 100, width: 500, height: 500 }.
opt	Object	Options: *Properties* Name / Type / Attributes / Description carAccelerate / ButtonExternalController / <optional> / Button 'accelerate' to accelerate when a certain gamepad button is pressed. carBrake / ButtonExternalController / <optional> / Button 'brake' to brake when a certain gamepad button is pressed. carSteeringWheel / SteeringWheel / <optional> / SteeringWheel 'steering_wheel' to rotate the current steering wheel with gamepad axes values. accelerateInstructionPVS / String / <optional> / String with the name of the pvs instruction for 'accelerate' action (Default is "accelerate"). brakeInstructionPVS / String / <optional> / String with the name of the pvs instruction for 'brake' action (Default is "brake"). steeringWheelInstructionPVS / String / <optional> / String with the name of the pvs instruction prefix regarding the direction changes with steering wheel (Default is "steering_wheel"). useButtonActionsQueue / Bool / <optional> / is the variable that allows to change press()/release() object invocations and ButtonActionsQueue invocations (Default is false). usePressReleasePVS / Bool / <optional> / is the variable that allows to change press()/release() click() invocations (Default is true). type / String / <optional> / Field 'type' allows to differentiate the axes of the external controller, i.e. to differentiate between gamepad axes and a more complex controller such as steeringWheelAndPedals axes (Default is "gamepad". Other possible value is "steeringWheelAndPedals"). accelerationIndex {Int} Index 'accelerationIndex' is the external controller index where acceleration action will be invoked (Default is 0). brakeIndex / Int / <optional> / Index 'brakeIndex' is the external controller index where brake action will be invoked (Default is 1).

accelerationIndex	Int	<optional>	Index 'accelerationIndex' is the external controller index where acceleration action will be invoked (Default is 0).
leftArrowIndex	Int	<optional>	Index 'leftArrowIndex' is the external controller index where turn left action(steering wheel left rotation) will be invoked (Default is 14).
rightArrowIndex	Int	<optional>	Index 'rightArrowIndex' is the external controller index where turn right action(steering wheel right rotation) will be invoked (Default is 15).
accelerationPedalIndex	Int	<optional>	Index 'accelerationPedalIndex' is the external controller(pedals external controller) index where acceleration action will be invoked (Default is 1).
brakePedalIndex	Int	<optional>	Index 'brakePedalIndex' is the external controller(pedals external controller) index where brake action will be invoked (Default is 1).
steeringWheelIndex	Int	<optional>	Index 'steeringWheelIndex' is the external controller(steering wheel external controller) index where steeringWheel widget rotate method will be invoked (Default is 0).
analogueStickIndex	Int	<optional>	Index 'analogueStickIndex' is the external controller index where steeringWheel widget rotate method will be invoked, based on X axis calculated angle (Default is 9).
leftAnalogueIndex	Int	<optional>	Index 'leftAnalogueIndex' is the external controller index where steeringWheel widget rotate method will be invoked, based on X,Y axes calculated angle (Default is 0).

rightAnalogueIndex	Int	<optional>	Index 'rightAnalogueIndex' is the external controller index where steeringWheel widget rotate method will be invoked, based on X,Y axes calculated angle (Default is 2).
pauseAction	Object	<optional>	Object with index 'pauseIndex', the external controller index where pause menu pvs instruction will be invoked, and string instructionPVS, which is the name of the pvs instruction for this action (Default is 9 and "pause").
quitAction	Object	<optional>	Object with index 'quitIndex', the external controller index where quit menu pvs instruction will be invoked, and string instructionPVS, which is the name of the pvs instruction for this action (Default is 8 and "quit").
resumeAction	Object	<optional>	Object with index 'resumeIndex', the external controller index where resume menu pvs instruction will be invoked, and string instructionPVS, which is the name of the pvs instruction for this action (Default is 16 and "resume").
muteAction	Object	<optional>	Object with index 'muteIndex', the external controller index where mute pvs instruction will be invoked, and string instructionPVS, which is the name of the pvs instruction for this action (Default is 4 and "mute").
unmuteAction	Object	<optional>	Object with index 'unmuteIndex', the external controller index where unmute pvs instruction will be invoked, and string instructionPVS, which is the name of the pvs instruction for this action (Default is 5 and "unmute").
useSensitivity	Boolean	<optional>	boolean to determine which rotation API will be invoked, i.e., with or without sensitivity (default is false).
sensitivityValue	Int	<optional>	the sensivity value to be provided to the rotation API with sensitivity (default is "gyroscope").

D

DOCUMENTAÇÃO DE TODOS CAMPOS OPCIONAIS DO *WIDGET TRACKGENERATOR*

Este apêndice apresenta o ficheiro *PDF* com a tabela com todos os campos opcionais e com os valores por omissão referentes ao *widget TrackGenerator*.

`constructor(id, coords, opt)` → {TrackGenerator}

Constructor for the TrackGenerator widget.

Parameters:

Name	Type	Description
id	String	The id of the widget instance.
coords	Object	The four coordinates (top, left, width, height) of the display, specifying the left, top corner, and the width and height of the (rectangular) display. Default is { top: 1000, left: 100, width: 500, height: 500 }.
opt	Object	Options: *Properties* Name / Type / Attributes / Description parent — String — <optional> — the HTML element where the display will be appended (default is "body"). spritesFilename — String — <optional> — the spritesheet filename(json file) that will be loaded (default is "spritesheet"). render — Object — <optional> — the rendering configurations, i.e. width, height, etc. (default is {depthOfField: 150, camera_distance: 30, camera_height: 320}). trackSegmentSize — Int — <optional> — the size of the track segment (default is 5). numberOfSegmentPerColor — Int — <optional> — the number of segments per color, i.e. how many sequenced segments to alternate colors (default is 4). numLanes — Int — <optional> — the number of lanes the track will be draw (default is 3). laneWidth — Float — <optional> — the width of the lane separator (default is 0.02). trackParam — Object — <optional> — the track configurations, i.e. number of zones(track length), etc (default is {numZones: 12, zoneSize: 250}).

		controllable_vehicle	Object	<optional>	the vehicle configurations, i.e. initial position, acceleration and deceleration values, etc (default is {position: 10, speed: 0, acceleration: 0.05, deceleration: 0.3, breaking: 0.6, turning: 5.0, posx: 0, maxSpeed: 15}).
		objects	Array	<optional>	the sprite names to be drawed in the landscape (default is []).
		obstacle	Array	<optional>	the sprite names to be drawed within the track as obstacles (default is []).
		trackLayout	Array	<optional>	the track layout that will be used to create the corresponding segments. (default is []).
		trackColors	Array	<optional>	the track colors that will be used to color the segments in each stripe. (default is {grass1: "#699864", border1: "#e00", border2: "#fff", outborder1: "#496a46", outborder_end1: "#474747", track_segment1: "#777", lane1: "#fff", lane2: "#777", laneArrow1: "#00FF00", track_segment_end:"#000", lane_end: "#fff"}).
		obstaclePerIteration	Int	<optional>	the number of iterations where a new obstacle will be placed within the track (default is 50).
		filePath	String	<optional>	the image path with spritsheet to load. (default is "../../client/app/widgets/car/configurations/").

E

DOCUMENTAÇÃO DE TODOS CAMPOS OPCIONAIS DO *WIDGET ARCADE*

Este apêndice apresenta o ficheiro *PDF* com a tabela com todos os campos opcionais e com os valores por omissão referentes ao *widget Arcade*.

`constructor(id, coords, opt)` → {Arcade}

Constructor for the Arcade widget.

Parameters:

Name	Type	Description
id	String	The id of the widget instance.
coords	Object	The four coordinates (top, left, width, height) of the display, specifying the left, top corner, and the width and height of the (rectangular) display. Default is { top: 1000, left: 100, width: 500, height: 500 }.
opt	Object	Options: *Properties* Name / Type / Attributes / Description

Name	Type	Attributes	Description
parent	String	<optional>	the HTML element where the display will be appended (default is "body").
scaleWindow	Float	<optional>	the scale to be set on canvas (default is 1).
trackFilename	String	<optional>	the track configuration filename, i.e. JSON file with the track that will be drawed as well as the required sprite coordinates, etc (default is "track").
spritesFilename	String	<optional>	the spritesheet configuration filename, i.e. JSON file with the all available sprites, whose coordinates are the same in trackFilename, i.e. the track must have been generated with this JSON as well so the coordinates will match (default is "spritesheet").
spritesFiles	Array	<optional>	array with spritesheets(images) names (default is ["spritesheet","spritesheet.text"]).
vehicleImgIndex	Int	<optional>	number placed as suffix in the JSON file with the sprite of the vehicle image (front, left side, right side) (default is null).
logoImgIndex	Int	<optional>	number placed as suffix in the JSON file with the sprite of the logo image (default is null).
backgroundImgIndex	Int	<optional>	number placed as suffix in the JSON file with the sprite of the background image (default is null).
realisticImgs	Bool	<optional>	value that indicates if the sprite of the vehicle to be used is a realistic image or if it is a pixelated image as in arcade games (default is "false").
useVehicle	Bool	<optional>	value that indicates if arcade will display a sprite of the vehicle (default is true).
vehicle	String	<optional>	the type of vehicle to be used in the simulation. The types available are ["airplane", "bicycle", "car", "helicopter", "motorbike"]. It should be noted that these types must exist in the spritesheet if they are to be used. (default is "car").
stripePositions	Object	<optional>	position values and respective widths (borders, track and finish line) to be rendered on a stripe. (default is { trackP1: -0.55, trackP2: 0.55, borderWidth: 0.08, inOutBorderWidth: 0.02, landscapeOutBorderWidth: 0.13, diffTrackBorder: 0.05, finishLineP1: -0.40, finishLineP2: 0.40, diffLanesFinishLine: 0.05 }).
lapNumber	Int	<optional>	the number of desired laps in the simulation (default is 0 laps, i.e. infinite simulation).
showOfficialLogo	Bool	<optional>	the option to render extra image, on the bottom-left corner, which is the PVSio-web logo created in this thesis (default is false).

loadPVSSpeedPositions	Bool	<optional>	allows to use PVS calculated positions and speed in the simulation. (default is true).
predefinedTracks	Int	<optional>	allows to use predefined tracks, present on JSON files with filename "trackLayout"+predefined+".json", in car/configurations/ directory. (default is null).
newLap_functionNamePVS	String	<optional>	allows to set pvs function name for new lap. (default is "set_positions_init").
action_attribute	String	<optional>	allows to set pvs attribute name for action. (default is "action").
direction_attribute	String	<optional>	allows to set pvs attribute name for direction. (default is "direction").
sound_attribute	String	<optional>	allows to set pvs attribute name for sound. (default is "sound").
speed_attribute	String	<optional>	allows to set pvs attribute name for speed. (default is "speed").
posx_attribute	String	<optional>	allows to set pvs attribute name for posx. (default is "posx").
position_attribute	String	<optional>	allows to set pvs attribute name for position. (default is "position").
speed_value	String	<optional>	allows to set pvs val name for speed attribute. (default is "val").
posx_value	String	<optional>	allows to set pvs val name for posx attribute. (default is "val").
position_value	String	<optional>	allows to set pvs val name for position attribute. (default is "val").
left_attribute	String	<optional>	allows to set pvs attribute name for left direction. (default is "left").
right_attribute	String	<optional>	allows to set pvs attribute name for right direction. (default is "right").
accelerate_attribute	String	<optional>	allows to set pvs attribute name for accelerate action. (default is "acc").
brake_attribute	String	<optional>	allows to set pvs attribute name for brake action. (default is "brake").
idle_attribute	String	<optional>	allows to set pvs attribute name for idle action. (default is "idle").
quit_attribute	String	<optional>	allows to set pvs attribute name for quit action. (default is "quit").
pause_attribute	String	<optional>	allows to set pvs attribute name for pause action. (default is "pause").
resume_attribute	String	<optional>	allows to set pvs attribute name for resume action. (default is "resume").
mute_attribute	String	<optional>	allows to set pvs attribute name for mute sound. (default is "mute").
unmute_attribute	String	<optional>	allows to set pvs attribute name for unmute sound. (default is "unmute").
filesPath	String	<optional>	the path with spritsheet image and other JSON files to load. (default is "../../client/app/widgets/car/configurations/").

De seguida apresenta-se a lista completa dos métodos privados implementados no *widget Arcade*.

O método privado *init()* seleciona a *div* com a *Canvas HTML5* onde serão "desenhados"os ambientes de condução. O método privado *onPageLoad(spritesFiles)* lê as imagens (*spritesheets*) para objetos *JavaScript Image*. O método privado *startSimulationAux()* interpreta e filtra o conteúdo dos ficheiros recebidos como campos opcionais dinamicamente. O método privado *renderEndMenu()* apresenta o menu de fim de simulação que é uma interface com um conjunto de instruções a realizar para começar uma nova simulação. O método privado *renderPauseMenu()* apresenta o menu de pausa de simulação que é uma interface com um conjunto de instruções a realizar para retomar a simulação. O método privado *renderSimulation()* apresenta o menu de início de simulação e renderiza os ambientes criados com o *widget TrackGenerator*. O método privado *setColorsCanvas(alternate, border1, border2, outborder1, outborder_end1, track_segment1, lane1, lane2, laneArrow1)* define as cores do simulador de condução. O método privado *setColorsEndCanvas(track_segment, lane)* define as cores da linha de fim (*finishing line*) no simulador de condução. O método privado *calculateNewControllableVehiclePosition()* interpreta a nova posição e os novos valores de velocidade e do número de rotações por minuto presente no estado *PVS* atual. O método privado *setControllableVehiclePosition(vehicleCurrentDirection, newSpeed, newPosition, newPositionX, vehicleXPosition, vehicleYPosition)* define a nova posição e os novos valores de velocidade e do número de rotações por minuto interpretados pelo método privado *calculateNewControllableVehiclePosition()*. O método privado *updateControllableVehicle()* define a nova posição e os novos valores de velocidade e do número de rotações por minuto caso o utilizador não pretenda utilizar as especificações formais para controlar a simulação (campo opcional *loadPVSSpeedPositions*). O método privado *detectBrowserType()* detecta o *browser* onde a simulação interativa está a correr. O método privado *drawSimpleArrowDown(x, y, color)* desenha seta simples (*arrow head*) para baixo. O método privado *drawSimpleArrowFront(x, y, color)* desenha seta simples para cima. O método privado *drawSimpleArrowLeft(x, y, color, direction)* desenha seta simples para o lado esquerdo. O método privado *drawSimpleArrowRight(x, y, color, direction)* desenha seta simples para o lado direito. O método privado *drawSprite(sprite, image, x, y, scale)* desenha uma imagem (sprite) no simulador recorrendo à API *drawImage()* da tecnologia *Canvas HTML5*. O método privado *drawText(string, pos, imageIndex, ratioRealisticFont)* desenha texto no simulador (velocidade atual, tempo decorrido, etc). O método privado *drawArrowFront(x, y, width, height, color, withLine)* desenha seta completa para cima. O método privado *drawArrowLeft(x, y, width, height, color, withLine, direction)* desenha seta completa para o lado esquerdo. O método privado *drawArrowRight(x, y, width, height, color, withLine, direction)* desenha seta completa para o lado direito. O método privado *drawBackground(position)* desenha a imagem de *background* que varia consoante a posição horizontal do veículo. O método privado *drawGuidingLine(pos1, scale1, offset1, pos2, scale2, offset2, delta1,*

delta2, color) desenha linha guia em cima da estrada. O método privado *drawLanePos(pos1, scale1, offset1, pos2, scale2, offset2, color, indexPos, laneWidth)* desenha as faixas na estrada numa dada posição horizontal. O método privado *drawLanes(pos1, scale1, offset1, pos2, scale2, offset2, color, numLanes, laneWidth)* desenha as faixas na estrada consoante o número de faixas presentes no ficheiro de configuração *JSON* fornecido no campo opcional *trackFilename*. O método privado *drawSegment(position1, scale1, offset1, position2, scale2, offset2, finishStart)* desenha uma faixa horizontal completa desde o limite da *Canvas HTML5* do lado direito até ao limite da *Canvas HTML5* do lado esquerdo (*full width*). O método privado *drawSegmentPortion(pos1, scale1, offset1, pos2, scale2, offset2, delta1, delta2, color)* desenha uma porção do segmento entre duas posições. Por exemplo, um segmento entre a posição *a* e *d* é desenhado com base nas porções do segmento cujas posições são de *a* a *b*, *b* a *c* e *c* a *d*.

F

SPRITESHEET UTILIZADA NOS PROTÓTIPOS DESENVOLVIDOS

Este apêndice apresenta a *spritesheet* utilizada na implementação dos protótipos do simulador de condução apresentados no capítulo 5.